Revisiting the Efficiency–Fairness Tradeoff of Shortest-Job-First Scheduling in GPU Clusters

Yi Yang
Department of Computer Science
University of Toronto
Toronto, Canada
yangy180@cs.toronto.edu

Zhongze Zheng
Department of Computer Science
University of Toronto
Toronto, Canada
zhongze.zheng@mail.utoronto.ca

***Abstract*—Shortest-Job-First (SJF) scheduling is widely regarded as an efficient policy for reducing average job completion time in large-scale GPU clusters, yet its fairness properties under realistic workloads remain poorly understood. In this work, we revisit SJF scheduling through the lens of fairness and robustness, using production-scale GPU traces from Alibaba together with an extended cluster simulator.**

We conduct a systematic study of SJF's efficiency–fairness tradeoff across both user-level and temporal fairness metrics. Our results reveal that, contrary to common concerns, SJF can simultaneously achieve substantial reductions in waiting time while maintaining high fairness under realistic workloads, provided that lightweight robustness mechanisms such as aging can be applied when bounding worst-case waiting times is critical.

We further examine the impact of imperfect runtime information by integrating a lightweight machine learning-based runtime predictor into the scheduler. Despite nontrivial prediction errors, the ML-assisted SJF closely matches the performance of oracle SJF and preserves strong fairness guarantees. Our findings suggest that SJF operates in a favorable efficiency–fairness regime in practice, and that accurate ranking rather than precise runtime estimation is sufficient to enable fair and effective GPU scheduling.

***Index Terms*—GPU scheduling, fairness, shortest-job-first scheduling, cluster scheduling, machine learning, rank correlation**

I. Introduction

Modern machine learning workflows increasingly rely on shared GPU clusters as the default execution environment for both large-scale training and data-intensive experimentation [1], [2]. In such environments, job runtimes are highly heterogeneous, ranging from short exploratory jobs to long-running production training workloads. This extreme runtime heterogeneity amplifies queueing effects under resource contention, causing waiting time to dominate end-to-end performance and making scheduling policy a primary determinant of system efficiency [3].

Shortest-Job-First (SJF) scheduling is theoretically optimal for minimizing average job completion time (JCT) when job runtimes are known, and is therefore widely used as a benchmark for efficiency-oriented scheduling policies. Recent systems studies demonstrate that even approximate SJF-style schedulers, enabled by runtime prediction, can substantially outperform FIFO in production GPU clusters [1]. However, these efficiency gains have long been viewed as fundamentally at odds with fairness. By prioritizing short jobs, SJF deprioritizes long-running workloads, raising concerns about starvation and unpredictable worst-case performance under sustained load. This tension has motivated a line of fair scheduling systems such as Themis [4] and Quincy [5], that explicitly enforce fair resource sharing, often at the cost of significantly longer job completion times. As a result, SJF has long been perceived as fundamentally trading fairness for efficiency, shaping the design of many fairness-first schedulers.

Most prior GPU scheduling work emphasizes efficiency metrics such as average JCT or throughput, while treating fairness as a secondary objective or enforcing it only indirectly through aging heuristics. Consequently, fairness is often evaluated implicitly or under instantaneous resource-sharing assumptions, rather than being examined as a first-class, user-facing performance property. As a result, how SJF-style policies behave across different fairness dimensions under realistic GPU workloads, and how these behaviors interact with efficiency remains poorly understood.

In this workshop, we deliberately re-examine SJF scheduling through a fairness-centric lens using realistic GPU workload traces. Lever aging an extended GPU cluster simulator and production traces from Alibaba, we conduct a comprehensive empirical study of SJF's behavior across multiple fairness dimensions, including user-level fairness, temporal fairness, and starvation risk. Rather than framing fairness as a binary constraint, we analyze how fairness degrades, recovers, and interacts with efficiency across workload conditions and scheduling assumptions.

Our empirical study yields several findings that directly challenge prevailing assumptions about SJF scheduling. First, we find that SJF substantially reduces average waiting time compared to FIFO, while maintaining high levels of fairness under realistic GPU workloads when combined with lightweight aging mechanisms. Contrary to widespread expectations, the resulting fairness degradation is markedly smaller than anticipated despite SJF's aggressive prioritization of short jobs. Second, we observe that fairness outcomes are strongly correlated with the quality of job ordering rather than the accuracy of absolute runtime estimates, suggesting that GPU cluster scheduling is fundamentally a ranking-oriented prob-

lem.

Motivated by this observation, we examine the impact of imperfect runtime information by integrating machine learning–based runtime predictors into SJF scheduling. Such predictors are increasingly feasible in practice, as prior work has shown that runtimes or resource demands can be approximated from historical traces and lightweight learning models [1], [6]–[8]. Concretely, we train multiple predictors (MLP, XGBoost, LightGBM, and a CRR++ variant) under different training objectives (MAE vs. MSE, including log-transformed targets), and compare them against history-based heuristics commonly used in production traces (e.g., user- and group-level duration estimates). Rather than evaluating predictors solely by absolute error, we additionally quantify how well each method preserves the relative ordering of job runtimes using rank-alignment metrics (Spearman's ρ and Kendall's τ), and then validate these findings end-to-end by running the resulting ML-SJF policies inside a GPU-cluster simulator.

Empirically, we find that ranking quality is a stronger predictor of scheduling performance than absolute accuracy: models that better preserve job ordering achieve JCT close to (and under certain placement dynamics, occasionally better than) the oracle-SJF baseline while maintaining strong fairness properties. Overall, these results suggest that accurate *relative* ordering is sufficient to capture most of SJF's efficiency benefits in shared GPU clusters, and that high-precision runtime prediction is not a prerequisite for deploying fair and effective SJF-style scheduling in practice.

Taken together, our results demonstrate that SJF can operate in a favorable efficiency–fairness regime in practice, even under realistic uncertainty in runtime information. This study directly challenges the conventional view of SJF as inherently unfair, and provides empirical guidance for deploying SJF-style scheduling policies in shared GPU clusters without sacrificing fairness.

II. A Fairness-Centric View of SJF

This section introduces the conceptual lens used throughout this study and clarifies how fairness is interpreted and examined in shared GPU clusters. Our goal is not to redefine fairness abstractly, but to establish a practical, user-facing perspective that guides the empirical analysis that follows.

In multi-tenant GPU clusters, users experience fairness primarily through job completion behavior rather than instantaneous resource allocation [1]. From this perspective, fairness is reflected in how long jobs wait under contention, how completion times scale with execution demands, and whether workloads are systematically disadvantaged over time.

Accordingly, we adopt a completion-time–oriented view of fairness. Under this framing, fairness is assessed based on observable completion outcomes, treating excessive waiting, disproportionate slowdowns, and unbounded delays as indicators of unfair behavior [9]. This view aligns fairness evaluation with user-perceived performance rather than internal scheduling mechanics [10].

Fairness in practice spans multiple dimensions. At the job level, fairness concerns whether individual jobs incur disproportionate delays due to queueing effects. At the user level, it captures whether jobs submitted by different users receive consistently unequal treatment. Fairness also has an inherent temporal dimension, as a scheduler may appear balanced on average while still exposing a subset of jobs to excessive or unbounded waiting, giving rise to starvation risk. These dimensions provide a structured lens for examining fairness behavior empirically.

III. Fairness Metrics and Implementation

We evaluate scheduling fairness using metrics derived directly from job execution traces and simulator logs. Consistent with the fairness-centric perspective introduced in Section II, we treat fairness as a user-facing, completion-time–oriented performance property rather than enforcing it as an explicit scheduling constraint. All metrics are computed post hoc from observable execution outcomes.

A. Logged Signals

For each job j, the simulator records its submit time a_j, completion time c_j, and execution duration d_j. From these signals, we derive the job completion time (JCT), defined as $T_j = c_j - a_j$, and the waiting time $W_j = T_j - d_j$. These quantities form the basis of all fairness metrics considered in this study.

B. Per-Job Fairness

We quantify per-job fairness using *slowdown*, defined as

$$\text{Slowdown}_j = \frac{T_j}{\max(d_j, 1)}.$$

Slowdown captures the relative performance degradation experienced by a job due to contention, normalizing completion time by execution demand [9]. To characterize both average and worst-case fairness behavior, we report aggregate statistics of slowdown, including the mean, standard deviation, and tail percentiles (P50, P95, and P99).

C. Per-User Fairness

To assess fairness across users, we group jobs according to their associated user identifiers provided in the trace. For each user, we compute the average slowdown across all submitted jobs. Inter-user fairness is summarized using the ratio between the maximum and minimum average slowdown across users. A ratio close to one indicates uniform treatment, while larger values reflect increasing disparity across users.

D. Temporal Fairness and Starvation Risk

We further evaluate temporal fairness by measuring starvation risk. A job is considered starved if its waiting time exceeds a fixed threshold, which we set to 3600 seconds by default. We report both the absolute number of starved jobs and the fraction of jobs that experience starvation under each scheduling policy. These metrics capture the risk of excessive or unbounded waiting without assuming any idealized fair-sharing baseline.

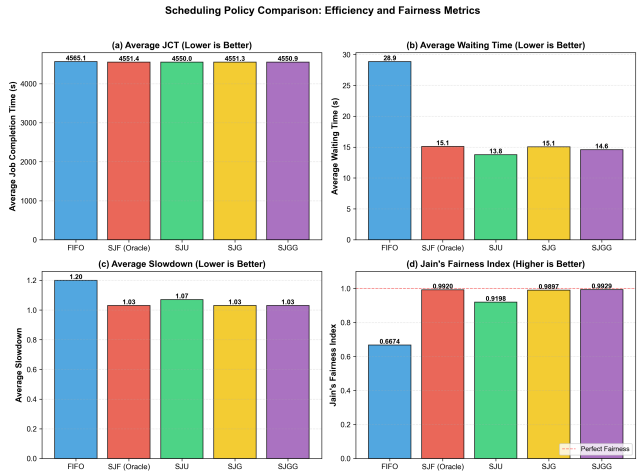

Fig. 1. Efficiency and fairness comparison between FIFO and SJF-style scheduling policies under different runtime estimation signals. We report (a) average job completion time (JCT), (b) average waiting time, (c) average slowdown, and (d) Jain's fairness index for FIFO, oracle SJF, and SJF variants using user-level (SJU), group-level (SJG), and group+GPU-level (SJGG) runtime estimators.

E. Discussion of Scope and Limitations

Our fairness evaluation focuses on observable completion-time metrics and does not simulate an idealized fair-sharing scheduler for comparison. As a result, starvation is detected and quantified but not actively prevented.

IV. Empirical Evaluation: Efficiency vs Fairness

We now empirically evaluate the efficiency fairness tradeoff of SJF-style scheduling under realistic GPU workloads. Using production-scale traces and our simulator, using production-scale traces from large multi-tenant GPU clusters [1]. we compare FIFO with a family of SJF-style policies, including oracle SJF and variants that rely on increasingly coarse runtime estimation signals. Unless otherwise specified, all results are aggregated over completed jobs.

A. Efficiency Results

We first examine efficiency in terms of average job completion time (JCT) and average waiting time. Figure 1 compares FIFO with a family of SJF-style scheduling policies, including oracle SJF and variants that rely on different runtime estimation signals.

Across all workloads, SJF consistently reduces average JCT compared to FIFO. However, the absolute improvement in average JCT is modest (below 1%). This behavior is expected in shared GPU clusters, where long-running training jobs dominate overall completion time statistics and inherently limit achievable reductions in average JCT [1].

In contrast, SJF yields substantial reductions in average waiting time. By prioritizing shorter jobs, SJF significantly reduces queueing delays and accelerates job turnover, particularly under high contention. These results indicate that SJF's primary efficiency benefit manifests in reduced waiting time rather than in average JCT alone. SJF-style policies consistently yield substantial reductions in average waiting time across all estimation variants. While the magnitude of improvement varies with estimation quality, the overall trend remains robust, indicating that the efficiency benefits of SJF persist even under imperfect runtime estimates.

B. Fairness Across Multiple Dimensions

We next evaluate fairness from a completion-time–oriented perspective. Rather than relying on a single metric, we examine fairness across user-level, temporal, and tail dimensions to capture different aspects of user experience under contention.

a) User-level fairness.: We measure user-level fairness by aggregating job slowdowns per user. FIFO exhibits substantial disparity across users, with some users consistently experiencing much higher slowdowns than others.

In contrast, SJF significantly improves user-level fairness, yielding more uniform slowdown distributions across users. Among all evaluated policies, SJF achieves the highest Jain's fairness index, indicating the strongest completion-time–oriented fairness. Notably, despite its aggressive prioritization of short jobs, SJF does not exacerbate inter-user disparity under realistic workloads.

b) Temporal fairness and tail behavior.: Beyond average behavior, we examine temporal fairness by analyzing tail waiting times and starvation risk [11].

FIFO exhibits heavy-tailed waiting time distributions, indicating unpredictable worst-case behavior under contention. SJF substantially reduces tail waiting times at high percentiles, improving predictability while maintaining strong average fairness guarantees. Across all evaluated workloads, we observe no starvation events under SJF, suggesting that, in practice, SJF does not induce the severe starvation behavior often associated with worst-case theoretical analyses.

C. The Role of Aging Mechanisms

We also evaluated an aging-based variant of SJF to mitigate extreme waiting times. Aging-style priority boosting has been widely used to improve robustness and mitigate starvation in scheduling systems [12]. However, aging does not improve completion-time–oriented fairness; compared to SJF without aging, it slightly reduces inter-user fairness as measured by Jain's fairness index, while exhibiting similar average slowdown.

These results suggest that aging is best viewed as a robustness mechanism rather than a primary fairness driver. In practice, aging may be desirable in highly bursty or adversarial workloads where bounding worst-case delays is critical, even at the cost of a small reduction in inter-user fairness.

V. Exploration of ML runtime predictor

The Alibaba cluster trace release and its original preprocessing code [13] contain on the order of 100,000 job records with engineered features. Each record includes cluster and resource specifications (e.g., GPU/CPU types and counts), hashed identifiers for jobs and user groups, and runtime estimates inferred from historical executions. Using this dataset, Weng et al. [1]

evaluate several scheduling baselines, including First-In-First-Out (FIFO) and size-aware policies such as Shortest Job First (SJF), as well as heuristics that order jobs by a predicted runtime. SJF is a classical optimal policy for minimizing total waiting time in the single-machine setting when true processing times are known [14]. This "oracle SJF" therefore intuitively provides an lower bound on job completion time, while FIFO serves as a simple baseline.

Based on their implementation, we first reprocess the trace-derived features and then evaluate multiple machine learning models for runtime prediction. Our baseline predictor is a multilayer perceptron (MLP). We also experiment with tree-based gradient boosting (XGBoost and LightGBM), and inspired by [15], [16], we implemented a more expressive mixture-of-experts (MOE) design that clusters jobs using k-means and trains multiple LightGBM experts, with a random forest classifier selecting the appropriate expert per job. The model is called CRR++ in the paper, so we reuse this term here. The goal is to test whether higher-capacity predictors can better capture the mapping from job features to runtime in large-scale, heterogeneous GPU clusters.

A. Data Preprocessing

Weng et al. [1] argue that mean absolute error (MAE) is preferable to mean squared error (MSE) because runtimes are extremely right-skewed and contain heavy tails. In our experiments, after applying an explicit variance-stabilizing transformation to the target runtime, MSE-trained models consistently achieved lower error. We therefore report both MAE and MSE in Section IV.

Directly regressing on raw durations is challenging because the target distribution is highly right-skewed since most jobs have short to moderate runtimes, while a small fraction run orders of magnitude longer. When training with mean squared error (MSE), these rare far-right samples can dominate optimization because squared loss penalizes the outliers much heavier. In addition, many classical regression diagnostics and inferential procedures rely on normality and constant-variance assumptions. The NIST Engineering Statistics Handbook notes that an appropriate transformation can often yield data that more closely follows a normal distribution [17]. The Box–Cox family formalizes this idea by selecting a power transform under which a normal, homoscedastic linear model is more appropriate [18].

Motivated by these observations, we compared several transformations to reduce skewness [19]. One method is the Box–Cox transform:

$$\mathrm{BC}_\lambda(x) = \begin{cases} \dfrac{x^\lambda - 1}{\lambda}, & \lambda \neq 0, \\ \ln(x), & \lambda = 0, \end{cases} \qquad x > 0.$$

When $\lambda \to 0$, Box–Cox approaches a log transform; when $\lambda = 1$, it reduces to the identity map. In practice, we observed that choosing λ near 0 produced the strongest reduction in skewness, effectively recovering a log-like transform. We therefore adopt a standard log transformation, which is widely

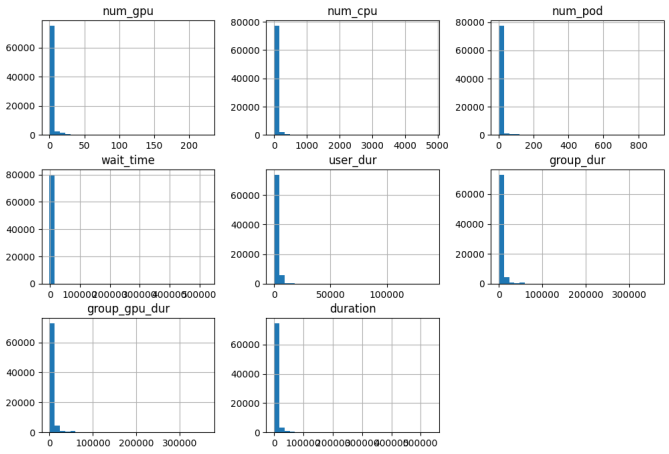

Fig. 2. Raw features

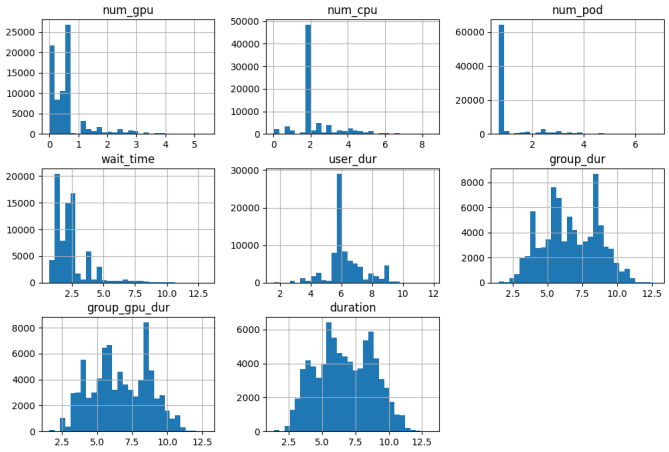

Fig. 3. Features after log transformation

used for right-skewed data because it compresses large values and yields distributions closer to normal [20]. Concretely, we transform durations using $f(x) = \log(x + 1)$, which safely handles zero-valued durations. Figures 2 and 3 visualize the effect of this transformation.

B. Model Implementation

In the original paper, the authors leverage task meta-information that is typically stable across repeated submissions–such as entry scripts, command-line arguments, and data sources/sinks–to form a grouping signal. They hash this meta–information to obtain a unique group identifier, and use features including the task's username (User), resource requests (Resource; e.g., GPU and other resources), and the resulting group tag (Group) [1]. This approach depends on historical coverage of these hashes: if a previously unseen hash appears (e.g., a new task or a small code change), the model cannot benefit from prior statistics, which can degrade performance.

Weng et al. [1] uses the raw, skewed data to schedule the tasks based on different types of infered runtime. However,

as argued in the previous section, we found the data to be extremely right–skewed so decide to try to process it first. To test out the capabilities of ML models, we decide to train on both settings. The first is to training using MAE metrics on raw data, similar to Weng et al.'s approach, the second is to training using MSE metrics on log transformed data. We want to try the second method because MSE fit much better than MAE on normally distributed data [find some reference].

We adopted 4 models, a simple two layer MLP with a ReLU activation function, an XGBoost model, an LGBM model and the CRR++ model. The CRR++ model first uses k-means to create 5 clusters, then a random forest classifier is trained with the input data and cluster centers as labels to later decide the expert to use. Since we set the number of clusters to 5, this means we will have 5 experts, or equivalently, LGBM models to train.

Before we train the model, we split the data to 80,000 data points of training and validation and 20,000 for testing. Since the original code [13] uses 20,000 random tasks to run the simulation, for the ease of comparison later, we also set the test set to 20,000 data points.

C. Comparison of Models

We trained all models with early stopping on a held-out validation set and saved the checkpoint with the best validation performance. We consider two training configurations: (i) optimizing an MSE-style objective on a log-transformed target, reported using RMSE in the transformed space; and (ii) optimizing an MAE objective on the raw target. Because squared-error objectives are sensitive to extreme values, using a transformed target can reduce the influence of heavy-tailed outliers and yield more stable fits on skewed runtime distributions [21].

TABLE I
PREDICTION ERROR COMPARISON (TRAINED WITH RMSE LOSS WITH LOG TRANSFORMATION).

Model	**RMSE**	**MAE**	p_{50}	p_{75}	p_{90}	p_{95}
MLP	8998.5	1611.9	15.8	33.8	63.9	92.1
XGBoost	9462.5	1487.2	15.6	32.9	64.0	93.2
LGBM	9352.8	1477.1	15.4	32.8	63.6	91.8
CRR++	9129.9	1494.2	15.6	33.8	64.3	94.3

RMSE, MAE, and APE percentiles on the test set.

TABLE II
PREDICTION ERROR COMPARISON (TRAINED WITH MAE LOSS WITHOUT LOG TRANSFORMATION).

Model	**RMSE**	**MAE**	p_{50}	p_{75}	p_{90}	p_{95}
MLP	9072.4	1513.6	15.3	35.3	69.5	107
XGBoost	9260.0	1632.8	29.6	74.3	200	388
LGBM	8831.4	1606.6	30.7	75.7	190	494
CRR++	8800.6	1620.6	34.4	85.9	218	512

RMSE, MAE, and APE percentiles on the test set.

Training losses from these two configurations are not directly comparable because they are computed under different

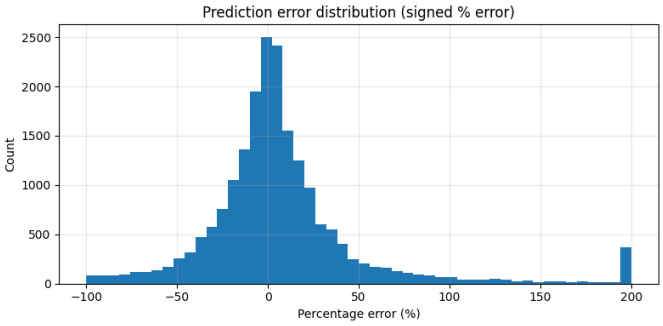

Fig. 4. XGBoost trained using a log-transformed target with an MSE-style objective (reported as RMSE).

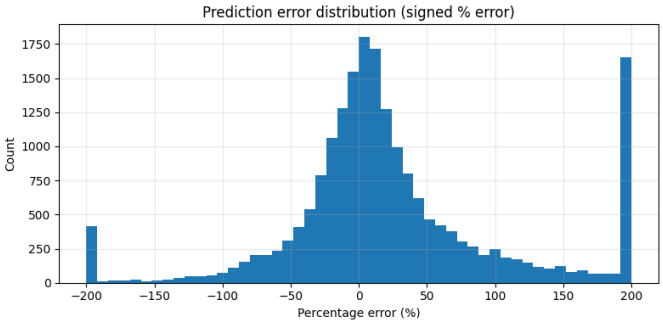

Fig. 5. XGBoost trained on the raw target using an MAE objective.

scalings and metrics. We therefore evaluate all models using a unified set of metrics on the test set in the *original* target scale: RMSE, MAE, and absolute percentage error (APE) percentiles. For each test sample with true value y and prediction $\hat{y}$, we define

$$\mathrm{APE} = 100 \cdot \left| \frac{\hat{y} - y}{y} \right|,$$

excluding samples with $|y| < \varepsilon$ (for a small ε) to avoid division by values close to zero. Tables I and II summarize the results.

As a result, RMSE and MAE alone do not yield a consistent model ranking, so we rely on APE percentiles to compare the percentage errors. Across the log-transformed MSE-trained setting (Table I), the models exhibit similar central error, with the *50th* percentile (p_{50}) around 15–16%. LGBM achieves the best overall tail behavior among these models, with the lowest or near-lowest errors also at p_{75}, p_{90}, and p_{95}, consistent with its strong validation performance during training. In contrast, under the MAE-trained setting without log transformation (Table II), only the MLP retains a competitive median APE ($p_{50} = 15.3\%$), while the tree-based methods (XGBoost, LGBM, and CRR++) show substantially worse tail errors (e.g., p_{90} and p_{95}), indicating unstable behavior on extreme samples when trained directly on the raw heavy-tailed target.

To better understand why the MAE-trained XGBoost model exhibits worse tail errors, we visualize the distribution of percentage errors. Figures 4 and 5 show histograms of signed percentage error, clipped to the range $[-200\%, 200\%]$ for readability, with values outside this range aggregated into

the boundary bins. XGBoost trained on the log-transformed target shows a much higher concentration of predictions near 0% error than the raw-target MAE-trained model, consistent with the substantially smaller APE percentiles in Table I. In addition, the raw-target MAE-trained model displays a noticeably positively skewed error distribution, suggesting a tendency toward overprediction on this right-skewed dataset.

D. More Practical Evaluations: Ordering Alignment for SJF

Pointwise error metrics (RMSE/MAE/APE) quantify numerical accuracy, but they do not directly measure whether a predictor preserves the *relative ordering* of job runtimes. Since an SJF-style policy ultimately sorts jobs by duration, we additionally evaluate ordering alignment between predicted and true runtimes using Spearman's ρ and Kendall's τ over the test set.

a) Spearman's ρ (global monotonic alignment): Spearman's ρ is a nonparametric rank correlation that measures the monotonicity of the relationship between two variables. In our setting, this measures whether jobs that are truly longer also tend to receive larger predicted runtimes, regardless of the exact scale [22]. Its main benefit of Spearman's ρ is that it summarizes global ordering consistency while being insensitive to the outliers, because we only use the rank of data. This makes ρ a stable indicator of whether the model learns the overall runtime ordering signal [23].

b) Kendall's τ (pairwise inversion sensitivity): Kendall's τ is also a rank correlation, but it is defined through pairwise agreements: it quantifies the correspondence between two rankings and increases when more pairs are ordered consistently [24]. The benefit of τ is its direct *pairwise* interpretation: it reflects how frequently the model introduces inversions in predicted results. This is especially meaningful for scheduling, since inversions are precisely the local ranking errors that can degrade an ordering-based policy such as SJF [25].

c) Interpretation: All metrics quantify ordering quality, with higher values indicating better agreement with the true runtime ranking. Spearman's ρ captures *global* monotonic alignment, whereas Kendall's τ captures *pairwise* ordering correctness and is therefore more directly related to inversions that can harm SJF-style sorting. We report both for complementary views of ranking quality. Tables III and IV summarize models trained with RMSE on log-transformed targets and with MAE on raw targets, respectively. We additionally include the history-based features used by the original scheduler (`user_dur`, `group_dur`, and `gpu_group_dur`) and evaluate how well their induced ordering matches the ground-truth durations [1].

Overall, models trained with RMSE on log-transformed data achieve consistently stronger rank alignment than models trained with MAE on raw data. Among the RMSE/log models, LightGBM yields the best alignment, though the gap to other strong models is small. The learned models also outperform the scheduler's history-based heuristics: the strongest heuristic, `gpu_group_dur`, achieves Spearman's $\rho \approx 0.94$ and Kendall's $\tau \approx 0.84$, compared to $\rho \approx 0.96$ and $\tau \approx 0.87$ for the best learned model.

TABLE III
ORDERING ALIGNMENT BETWEEN PREDICTED AND TRUE RUNTIMES (TRAINED WITH RMSE LOSS WITH LOG TRANSFORMATION).

Model	**Spearman's ρ**	**Kendall's τ**
MLP	0.956	0.862
XGBoost	0.961	0.871
LGBM	0.962	0.871
CRR++	0.961	0.870
user_dur	0.32	0.23
group_dur	0.88	0.77
gpu_group_dur	0.94	0.84

Metrics computed on the test set using predicted and true runtimes.

TABLE IV
ORDERING ALIGNMENT BETWEEN PREDICTED AND TRUE RUNTIMES (TRAINED WITH MAE LOSS WITHOUT LOG TRANSFORMATION).

Model	**Spearman's ρ**	**Kendall's τ**
MLP	0.952	0.860
XGBoost	0.924	0.800
LGBM	0.921	0.799
CRR++	0.920	0.802
user_dur	0.32	0.23
group_dur	0.88	0.77
gpu_group_dur	0.94	0.84

Metrics computed on the test set using predicted and true runtimes.

VI. EVALUATIONS

A. ML Runtime Prediction

1) Experiment Setup: We simulate a workload of 15,000 tasks, with 1,000 jobs submitted per client, and compare the available baselines against our ML-powered SJF variants. The baseline policies include oracle SJF, FIFO, and the history-based SJF heuristics from [1]: SJU, SJG, and SJGG. These heuristics order jobs using inferred runtimes derived from past executions. Concretely, SJU estimates duration from each user's historical jobs (User). SJG estimates duration based on a hashed *group tag* constructed from stable submission meta-information such as entry scripts, command-line arguments, and data sources/sinks. SJGG further add information to this group-based estimate on the requested GPU type and count, yielding a more fine-grained history bucket. Our ML models include a simple MLP, tree-based gradient boosting (XGBoost and LightGBM) and a MOE–style LightGBM.

The baseline simulator consumes preprocessed trace data and does not model the runtime cost of predicting job durations. To quantify inference overhead, we initially invoked the predictor online each time the scheduler assigned tasks to nodes. In this naive integration, repeated small-batch calls dominated runtime: end-to-end simulation time increased from $\sim$60 s (original code) to 120–280 s for tree/MLP models and $>$800 s for CRR++. Profiling indicated the overhead was primarily due to Python-level invocation and many small batches rather than the underlying model computation. We

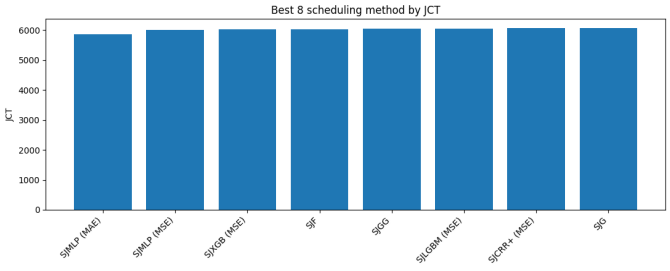

Fig. 6. Top 8 benchmark over existing methods.

therefore switch to batched inference, computing predictions for all tasks in one call before scheduling, representing a server-side batch inference service or parallelized client-side inference. Under batched inference, predicting all tasks took $< 1\,\mathrm{s}$ for all models, and we use this mode for the reported results.

B. Performance Comparison

Figure 6 shows the comparison of the best 8 methods. And surprisingly, the oracle SJF sometimes gives suboptimal performance in terms of JCT and is outperformed by MLP training with both MAE and MSE metric. We have ran the experiments several times and most of the time the SJF with MLP model outperforms oracle SJF. However, LGBM which performs the best in our previous evaluations never beats MLP models in this benchmark. Therefore, we take a look into the code to find what could cause this issue.

In the simulator, jobs are not pre-assigned to nodes. Instead, the system maintains a single global pending queueof tasks. At each scheduling step, the scheduler (i) sorts this global queue according to the chosen policy, and then (ii) greedily scans the sorted list and attempts to place each job onto the first feasible node according to an ordering policy per node, specific to different algorithms (e.g., by node id, idle GPUs, or utilization). As soon as a feasible placement is found, the job is committed to that node. This procedure makes the schedule order dependent, because small changes in job ordering can change which nodes have capacity at the moment each job is considered, which in turn changes subsequent feasibility and packing across different algorithms and introduces fluctuations.

As a result, even when job durations are known, oracle SJF is not necessarily optimal in this setting because placement decisions are coupled with ordering. Greedy first–fit placement can introduce resource fragmentation. For example, some tasks need multiple instances; thus, the greedy algorithm is suboptimal here. Feasibility constraints such as GPU-type matching can further restrict candidate nodes. These effects can lead to run–to–run variability and sometimes allow an ML–based ordering to outperform oracle SJF under the same placement policy in such variable setting.

REFERENCES

[1] Q. Weng, W. Xiao, Y. Yu, W. Wang, C. Wang, J. He, Y. Li, L. Zhang, W. Lin, and Y. Ding, "MLaaS in the wild: Workload analysis and scheduling in Large-Scale heterogeneous GPU clusters," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, 2022, pp. 945–960. [Online]. Available: https://www.usenix.org/system/files/nsdi22-paper-weng.pdf

[2] M. Jeon, S. Venkataraman, A. Phanishayee, J. Qian, W. Xiao, and F. Yang, "Analysis of large-scale multi-tenant GPU clusters for DNN training workloads," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. USENIX Association, 2019, pp. 947–960. [Online]. Available: https://www.usenix.org/system/files/atc19-jeon.pdf

[3] T. N. Le, X. Sun, M. Chowdhury, and Z. Liu, "Allox: Allocation across computing resources for hybrid CPU/GPU clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 46, no. 2, pp. 87–88, 2019. [Online]. Available: https://doi.org/10.1145/3305218.3305251

[4] K. Mahajan, A. Balasubramanian, A. Singhvi, S. Venkataraman, A. Akella, A. Phanishayee, and S. Chawla, "Themis: Fair and efficient GPU cluster scheduling," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. USENIX Association, 2020, pp. 289–304. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/mahajan

[5] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. V. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP '09)*. ACM, 2009, pp. 261–276. [Online]. Available: https://doi.org/10.1145/1629575.1629601

[6] P. Yu, J. Liu, and M. Chowdhury, "Fluid: Resource-aware hyperparameter tuning engine," in *Proceedings of Machine Learning and Systems (MLSys)*, vol. 3, 2021, pp. 502–516. [Online]. Available: https://proceedings.mlsys.org/paper_files/paper/2021/file/c0987e6b6da2428e8cd43efa74790ccb-Paper.pdf

[7] Y. Peng, Y. Bao, Y. Chen, C. Wu, and C. Guo, "Optimus: An efficient dynamic resource scheduler for deep learning clusters," in *Proceedings of the Thirteenth EuroSys Conference (EuroSys '18)*. ACM, 2018. [Online]. Available: https://doi.org/10.1145/3190508.3190517

[8] Q. Hu, M. Zhang, P. Sun, Y. Wen, and T. Zhang, "Lucid: A non-intrusive, scalable and interpretable scheduler for deep learning training jobs," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '23), Volume 2*. ACM, 2023, pp. 457–472. [Online]. Available: https://doi.org/10.1145/3575693.3575705

[9] N. Bansal and M. Harchol-Balter, "Analysis of srpt scheduling: Investigating unfairness," in *Proceedings of the 2001 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 2001, pp. 279–290.

[10] A. Wierman, "Fairness and scheduling in single server queues," https://users.cms.caltech.edu/~adamw/papers/survey.pdf, 2011, accessed 2026-01-04.

[11] J. Rasley, K. Karanasos, S. Kandula, R. Fonseca, M. Vojnovic, and S. Rao, "Efficient queue management for cluster scheduling," in *EuroSys '16: Proceedings of the Eleventh European Conference on Computer Systems*, London, United Kingdom, 2016.

[12] T. Becker and T. Schüle, "Evaluating dynamic task scheduling with priorities and adaptive aging in a task-based runtime system," in *Architecture of Computing Systems – ARCS 2020*, ser. Lecture Notes in Computer Science, vol. 12155. Springer, 2020, pp. 17–31. [Online]. Available: https://doi.org/10.1007/978-3-030-52794-5_2

[13] Alibaba Group, "Alibaba Cluster Trace Program (clusterdata)," GitHub repository, 2025. [Online]. Available: https://github.com/alibaba/clusterdata

[14] A. Miller and H. Hassanieh, "Scheduling jobs to minimize average waiting time (optimality of shortest job first)," CS 374: Algorithms and Models of Computation, University of Illinois Urbana-Champaign (lecture notes), 2020, accessed 2025-12-31. [Online]. Available: https://courses.grainger.illinois.edu/cs374/fa2020/lec_prerec/19/19_3_0_0.pdf

[15] Anonymous, "non-clairvoyant-with-predictions (anonymous repository)," Anonymous repository hosted on 4open.science, accessed 2025-12-31. [Online]. Available: https://anonymous.4open.science/r/non-clairvoyant-with-predictions-7BF8/README.md

[16] Anonymous, "ATLAS: Alibaba dataset and benchmark for learning-augmented scheduling," in *Submitted to The Fourteenth International Conference on Learning Representations*, 2025, under review. [Online]. Available: https://openreview.net/forum?id=QBvxXzHdZx

[17] NIST/SEMATECH, "Box-cox normality plot," NIST Engineering Statistics Handbook, accessed 2025-12-31. [Online]. Available: https://www.itl.nist.gov/div898/handbook/eda/section3/boxcox.htm

[18] G. E. P. Box and D. R. Cox, “An analysis of transformations,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 26, no. 2, pp. 211–243, 1964. [Online]. Available: https://doi.org/10.1111/j.2517-6161.1964.tb00553.x

[19] N. Jermain, “Transforming skewed data for machine learning,” Open Data Science (ODSC), accessed 2025-12-31. [Online]. Available: https://opendatascience.com/transforming-skewed-data-for-machine-learning/

[20] GeeksforGeeks, “Log transformation,” GeeksforGeeks, Jun. 2025, last updated 21 Jun 2025; accessed 2025-12-31. [Online]. Available: https://www.geeksforgeeks.org/data-science/log-transformation/

[21] M. Eppert, P. Fent, and T. Neumann, “A tailored regression for learned indexes: Logarithmic error regression,” in *aiDM'21*, 2021.

[22] SciPy Developers, “scipy.stats.spearmanr,” https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html, 2026, accessed 2026-01-02.

[23] C. Spearman, “The proof and measurement of association between two things,” *The American Journal of Psychology*, vol. 15, no. 1, pp. 72–101, 1904, commonly cited as the original introduction of rank correlation; PDF available online.

[24] SciPy Developers, “scipy.stats.kendalltau,” https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.kendalltau.html, 2026, accessed 2026-01-02.

[25] M. G. Kendall, “A new measure of rank correlation,” *Biometrika*, vol. 30, no. 1–2, pp. 81–93, 1938.