The Cost of Accurate Predictions in Learning-Augmented Scheduling

Zhiyun Jiang, Tianming Zhao, Chunqiu Xia, Wei Li, Albert Y. Zomaya
School of Computer Science, The University of Sydney, Australia
zjia5638@uni.sydney.edu.au, tzha2101@uni.sydney.edu.au, chunqiu.xia@sydney.edu.au,
weiwilson.li@sydney.edu.au, albert.zomaya@sydney.edu.au

***Abstract*—This paper considers non-clairvoyant scheduling with predictions to minimize total completion time, with a focus on the often-overlooked impact of prediction costs. Prior works assume ideal settings where predictions are cost-free at inference, but quality predictions require time, which is critical when optimizing a time-sensitive metric. We investigate the trade-off between prediction accuracy and inference time in learning-augmented scheduling by simulating datasets with models of varying accuracy and inference times. Using Gaussian Processes to model this relationship, we find that highly accurate predictions with long inference times are suboptimal, whereas moderately accurate models with fast inference times yield empirically optimal performance.**

I. Introduction

Learning-augmented scheduling algorithms surpass the pessimistic assumptions in non-clairvoyant scheduling, where the scheduler has no information about job sizes until the job is completed. In the learning-augmented setting, the algorithm can access a machine learning (ML) model that predicts the unknown job sizes [1], [2]. With job size predictions, one can design a scheduler with theoretical guarantees, including consistency and robustness [1], [2]. These properties ensure the algorithm performs well under perfect predictions and degrades gracefully in the less predictable cases.

An often-overlooked aspect is that predictions come with no costs, i.e., the framework assumes a free job size prediction per job size readily available at job arrival. In reality, making predictions requires time and computing resources [3]–[6]. The ML model must gather data and use computing resources to make inferences. The time to complete all tasks increases substantially due to the cumulative effect of these costs.

To illustrate this impact concretely, consider single-machine static scheduling to minimize total completion time, shown in Fig. 1. If inference costs were negligible ($I_{\text{cost}} \approx 0$ for all jobs), the total completion time $\sum_{j=1}^{n} C_j$ is computed solely with processing times, as in the first two schedules. The quantity $\sum_{j=1}^{n} I_{\text{cost}} = n \cdot I_{\text{cost}}$ precisely captures the cumulative inference overhead; the third scheduled task's completion time includes inference delays from itself and two preceding tasks. The example shows that all jobs are available at time 0 with unknown processing times. The first schedule depicts a poor job sequence, where jobs are scheduled from longest to shortest. This results in a total completion time of 42. The second schedule represents the Round Robin approach, the optimal non-clairvoyant scheduler, which equally shares processing time among active jobs, achieving a total completion time of 32. Schedules 3 and 4 incorporate predictions: the third uses moderate predictions, while the fourth has perfect predictions. Suppose inference requires the processing unit and will delay the job execution, with schedule 3 taking 1 and schedule 4 taking 6 units. Although scheduler 4 performs optimal sequencing by accurately predicting job processing times, its total completion time is worse than the first schedule's.

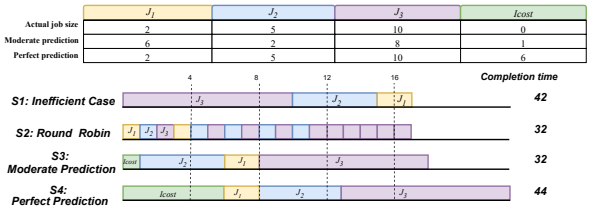

Fig. 1. Four scheduling executions illustrate how prediction accuracy and inference delay impact the total completion time. The table displays job's processing times (for J_1, J_2, and J_3) and the prediction cost (I_{cost}) in S3 and S4. The total completion time is the sum of individual job completion times, defined as $C_{total} = \sum_{j=1}^{3} C_j$, where each job's completion time (C_j) includes both processing and inference delays. Perfect predictions (S4) result in a completion time of 44 units, while moderate predictions yield 32 units, indicating that prediction costs can offset the benefits of ideal job sequencing.

Job size predictions can improve scheduling performance guarantees, but inference costs introduce delays that may counteract benefits, particularly in time-sensitive optimization. This trade-off points to a broader challenge:

How do prediction costs impact the performance of learning-augmented algorithms?

We take an empirical approach to this question by simulating datasets of job characteristics such that job features have predictability with job sizes [7]. In the simplest setting, we generate features and job sizes from a multivariate normal distribution [8]. We then use Gaussian Process Regression (GPR) as the ML model to predict job sizes. With more observations in the dataset, GPR predictions become more accurate, although inference time increases. This trade-off is captured by assuming that all predictions for single- or multi-machine scenarios are performed on the same hardware [9], thus introducing the effect of prediction cost. We apply learning-augmented scheduling algorithms in the experimental setting

2325-1301/25/$31.00 ©2025 IEEE
DOI 10.1109/RTCSA66114.2025.00031

to measure performance (e.g., total completion time competitive ratio), and consistently observe a U-shaped trend: modestly accurate yet cheaper predictions often outperform near-perfect inference when overhead grows too large. Extending beyond the single-machine case, we investigate multi-machine scheduling, where parallelization reveals that inference costs can dominate more quickly than in single-CPU environments. We also vary the job size distribution: comparing normal, lognormal, and exponential models to highlight how tail-heavy distributions affect both predictive accuracy and the overall scheduling outcome. Further, we incorporate job release times, showing that scenarios with sequential arrivals often reduce queue buildup and thus partially mitigate inference delays.

A. Main Contributions

In this work, we challenge the common assumption in learning-augmented scheduling that job-size predictions incur no cost. Our primary contributions are as follows:

- **Modeling Prediction Costs:** We introduce the concept of *prediction cost* into scheduling, demonstrating that the time required to infer job sizes is not negligible. Our approach reveals that inference delays, experienced by every job, compound to impact the overall total completion time significantly.
- **Empirical Analysis of the Accuracy–Cost Trade-off:** Through extensive simulations, we analyze how observation numbers affect both prediction accuracy and inference time. Our results uncover a distinct U-shaped relationship in the performance ratio, where initial improvements in prediction accuracy eventually give way to inference cost dominance, thereby degrading scheduling performance.
- **Algorithmic Advancement with DPRR:** We propose Dynamic Preferential Round Robin (DPRR), a novel scheduling algorithm that dynamically adjusts its processing power allocation, in contrast to user pre-defined schemes such as PRR [1].
- **Comprehensive Experimental Evaluation:** Our study considers various scenarios—including different job size distributions, multiple machine settings, and varying job release times—to show that the impact of prediction cost is multifaceted.

II. Preliminaries

A. Problem definition

We consider an online scheduling problem where n independent jobs released at time 0 or incrementally over time on a single or multiple machines. Every job J_j $(1 \leq j \leq n)$ is with an unknown processing time p_j^* (interchangeably referred to as *job size* in this paper) until the job is finished [10]–[13]. The literature calls such a setting as *non-clairvoyant scheduling*. The jobs are preemptive, meaning that one can stop processing a job and later resume it [14]–[17]. Given a schedule, let C_j denote the completion time of job J_j. Our objective is to minimize the total completion time $\sum_{j=1}^{n} C_j$ (abbreviated as $\sum C_j$); it captures the cumulative delay experienced by all jobs due to scheduling decisions such us scheduling early arriving job in later order and inference time directly delaying job execution. Our study aims to solve two conflicting goals: correct job execution order to approximate the optimal offline scheduling and minimization of prediction computation. In Graham notation, our scheduling problem can be defined as $1 \mid \textit{online-time-nclv}, \textit{pmtn} \mid \sum C_j$ [18]. Instead of knowing the exact job sizes, the algorithm has access to a job size prediction $\hat{p}_j$ for each job J_j. These predictions may not be perfect and may have errors, which we will define shortly. Our study quantifies the impact of prediction costs on minimizing total completion time, incorporating the computational cost of generating predictions. This cost, referred to as the *inference cost* (denoted as I_{cost}), represents the total time spent producing job size predictions.

B. Gaussian process regressions (GPR)

Recent studies have advanced job size prediction, using either empirical measurements in controlled environments [19]–[21] or statistical estimations based on historical data and job characteristics [22]–[24]. In ML terms, job characteristics are considered features indicative of job size. In our setting, job sizes are unknown at the decision-making point and must be predicted online [25], [26]. We model each job J_j by a feature vector $\mathbf{x}_j \in \mathbb{R}^p$, where p is the number of features. We assume an underlying function $f : \mathbb{R}^p \to \mathbb{R}$ that maps every job feature vector $\mathbf{x}_j$ to its job size p_j^*. While job sizes in queuing systems traditionally follow inverse Gaussian distributions—arising naturally as first passage times of Brownian motion with drift [60], [61]—we model f using a Gaussian Process (GP) prior [27]–[31]:

$$f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

where $m(\mathbf{x})$ is the mean function and $k(\mathbf{x}, \mathbf{x}')$ is the covariance function defining the correlation between any two feature vectors $\mathbf{x}$ and $\mathbf{x}'$. We use GPR for two main reasons. First, leveraging a Bayesian framework, GPR has been widely used for predictive tasks: given a new job feature vector $\mathbf{x}^*$, it predicts the expected job size $\mathbb{E}[f(\mathbf{x}^*)]$ along with an uncertainty estimate $\mathrm{Var}(f(\mathbf{x}^*))$ [30]. In scheduling contexts, GPR has proven effective in job-runtime forecasting [23] and load prediction [28] due to its kernel-based ability to capture non-linearity and correlation in the data. In addition, GPR naturally encodes the trade-off between inference costs and accuracy: increasing training observations enables better precise prediction but comes with a higher inference cost.

C. Performance evaluation

a) Prediction error: Suppose the ML model generates a series of job size predictions $(\hat{p}_1, ..., \hat{p}_n)$ for the true job sizes $(p_1^*, ..., p_n^*)$. We consider two prediction error metrics: Mean Absolute Error (MAE) and Root Mean Square Error (RMSE):

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n} |\hat{p}_j - p_j^*|, \quad \text{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n} (\hat{p}_j - p_j^*)^2}$$

MAE is commonly used in theoretical analysis [1], [2], because it directly relates to theoretical bounds. RMSE is preferred in empirical studies as it penalizes larger errors more heavily, better reflecting real-world performance where large mis-predictions are particularly costly.

b) Competitive ratio: The competitive ratio compares an online algorithm's performance to the offline optimal. For minimizing completion time, Shortest Remaining Processing Time (SRPT) is optimal when job sizes are known [32]. For algorithm A, the competitive ratio is: $R(A) = \sup_\sigma \frac{C_A(\sigma)}{C_{\text{SRPT}}(\sigma)}$ where σ is any job instance, sup (supremum) is the worst-case ratio over all instances, and $C_A(\sigma)$, $C_{\text{SRPT}}(\sigma)$ are total completion times. Lower ratios are better as they are closer to optimal.

c) Consistency and robustness: Learning-augmented algorithms are evaluated on two key metrics [1]. 1) Consistency measures performance under perfect predictions. An algorithm has consistency β if its competitive ratio equals β when predictions exactly match actual job sizes ($\eta = 0$). Ideally, $\beta = 1$, matching the offline optimal. 2) Robustness measures worst-case performance under arbitrary prediction errors. An algorithm has robustness γ if its competitive ratio never exceeds γ regardless of prediction quality. For non-clairvoyant scheduling, the optimal robustness is 2, achieved by Round Robin.

D. Related work

a) Non-clairvoyant scheduling: Numerous studies have explored non-clairvoyant scheduling, where job sizes are unknown to the scheduler [10], [33], [34]. For single-machine scheduling to minimize total completion time, the Round-Robin (RR) algorithm, which cycles through unfinished jobs in equal time slices, achieves the optimal 2-competitive ratio [35]. Becchetti and Leonardi [36] consider scheduling parallel machines to minimize total flow time as non-clairvoyant. Their work shows that the Randomized Multi-level Feedback algorithm achieves a competitive ratio of $O\left(\log n \log \frac{n}{m}\right)$, where m is the number of machines.

b) Learning-augmented algorithms: Learning-augmented algorithms have shown promise in scheduling jobs involving numerous constraints, objectives, and sequence-dependent setups. The work in [37] proposes an ML approach for multi-target regression to enhance scheduling, which improves computational efficiency and solution quality. Recent studies [1], [2] have advanced the field by addressing the balance between consistency and robustness for learning-augmented algorithms. These works reveal a robustness-consistency trade-off and provide tight performance bounds for non-clairvoyant job scheduling and ski rental problems [1], [2].

c) Embedded real-time scheduling: Real-time and embedded systems also benefit from ML predictions, given their strict deadline and resource constraints. Classical real-time scheduling relies on well-known algorithms such as EDF or RM scheduling [14]; however, incorporating ML-based predictions can enhance responsiveness without violating temporal guarantees. The work [38] explores real-time scheduling and power allocation in wireless networks using deep neural networks, emphasizing the feasibility of integrating ML-based schedulers to manage resource allocation. Further theoretical work [39] formalizes scheduling with learned predictions in time-critical systems, underscoring the potential for improved performance when precise estimates are available.

d) Meta-reasoning and prediction cost: Beyond machine-learning-based scheduling, prior work in temporal planning and meta-reasoning [40], [41] emphasizes the cost of making predictions or decisions under time constraints. Planners must balance the time spent reasoning before dispatching actions against tight deadlines [40], often referred to as a prediction cost. In some models, the cost of allocating t units of time to process i appears as $C_i(t) = -\log\big(1 - s_i(t, tb)\big)$, where $s_i(\cdot)$ captures the log-probability of success within time t. Formal meta-reasoning approaches rely on cost predictions to optimize resource usage and action scheduling [40], [41], sometimes formulating a total decision-making cost as $C_{\text{total}} = \sum_i \big(C_i(t) + P(t)\big)$, where $P(t)$ represents time-based overheads. Incorporating these estimates into the planning loop helps planners adapt to deadlines [41], indicating that prediction overhead is not confined to machine-learning-based scheduling but constitutes a broader challenge in time-critical domains.

e) Job completion time minimization: Minimizing completion time remains a key objective in scheduling, especially for systems that manage parallel and batch processing [42], [43]. In the single-machine case, the SRPT algorithm provides the optimal schedule when the exact job sizes are known [32]. For non-clairvoyant scheduling, Round Robin (RR) achieves an optimal 2-competitive ratio [35]. Kumar et al. [1] propose Preferential Round Robin (PRR), the first learning-augmented algorithm for this problem. PRR runs SPJF (Shortest Predicted Job First) and RR in parallel with rates λ and $(1-\lambda)$ respectively, where $\lambda \in (0, 1)$. With k active jobs, the predicted shortest job gets rate $\lambda + \frac{1-\lambda}{k}$ while others get $\frac{1-\lambda}{k}$, balancing prediction benefits with robustness. PRR achieves competitive ratio $\min\left\{\frac{1}{\lambda}\left(1 + \frac{2\eta}{n}\right), \frac{2}{1-\lambda}\right\}$ where $\eta = n \cdot$MAE, yielding $\frac{1}{\lambda}$-consistency and $\frac{2}{1-\lambda}$-robustness [1]. Subsequent work proved optimal trade-offs [2], explored alternative metrics [44], and extended to parallel machines [45].

III. Modeling prediction costs

GPR has been used effectively in data-driven systems and load forecasting [27], [28], [46]. We justify its application in scheduling: the heart of scheduling is decision-making under uncertainty, models that produce probabilistic output, such as a mean and uncertainty estimate, are useful. If we consider the job size prediction for job j with features $\mathbf{x}_j$, a probabilistic model like GPR can represent this as follows: $\hat{p}_j \sim \mathcal{N}(\mu_j, \sigma_j^2)$, where $\hat{p}_j$ is the job size prediction for j, and $\mu_j = \mathbb{E}[\hat{p}_j|\mathbf{x}_j]$ represents the mean of the prediction. Term $\sigma_j^2 = \text{Var}(\hat{p}_j|\mathbf{x}_j)$ denotes the variance in the prediction for job j. Incorporating the expected job size and the confidence interval around that

prediction enables more robust and adaptive decision-making under uncertainty.

A. Modeling assumption

We assume that true job sizes, denoted p_j^*, follow multivariate normal, lognormal, and exponential distributions. While this may not reflect real-world workloads, it provides a practical framework that enables non-parametric ML models, such as GPR, to perform effective inference [47]. The assumption is not unfounded. Empirical studies show that real-world job characteristics, including job sizes, often exhibit heavy-tailed distributions, such as Pareto distributions [48]. With statistical techniques like the Box-Cox transformation, these distributions can be approximated to a normal form, allowing effective learning and predictive performance in ML tasks [49].

B. Job size distribution

a) Multi-normal jobs sizes: Our study, under the assumptions of the GPR model, simulates job sizes and features under a multi-normal distribution: the feature set $\mathbf{X}$ and job size $\mathbf{y}$ is defined by: $\begin{bmatrix}\mathbf{X}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\mathbf{X} \in \mathbb{R}^{n\times p}$ represents the matrix of job features for n jobs, each characterized by a vector of p attributes, and $\mathbf{y} \in \mathbb{R}^n$ represents the vector of actual job sizes. Each element is denoted as p_j^* for each job J_j. Here, $\mathbf{x}_j \in \mathbb{R}^p$ represents the complete feature set for the j-th job. The mean vector $\boldsymbol{\mu} = \begin{bmatrix}\mathbf{0}_p\\ \mu_y\end{bmatrix}$ supports the multi-normal distribution by defining $\mathbf{0}_p \in \mathbb{R}^p$ as a zero vector for each feature [28], implying no initial mean shift within the feature set, and μ_y establishes the central tendency for job sizes. The processed job sizes p_j^* are ensured to be non-negative.

The covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{(p+1)\times(p+1)}$ models variability within the feature set $\mathbf{X}$ and relationships between features and the job size vector $\mathbf{y}$ [29], [30]: $\boldsymbol{\Sigma} = \begin{bmatrix}\boldsymbol{\Sigma}_{\mathbf{X}} & \sigma_{\mathbf{X},y}\\ \sigma_{\mathbf{X},y}^\top & \sigma_{yy}\end{bmatrix}$, where $\boldsymbol{\Sigma}_{\mathbf{X}} \in \mathbb{R}^{p\times p}$ captures the covariances among the p features within $\mathbf{X}$, $\sigma_{\mathbf{X},y} \in \mathbb{R}^p$ represents the covariance vector between each feature in $\mathbf{X}$ and $\mathbf{y}$, and σ_{yy} denotes the variance of $\mathbf{y}$, quantifying the spread in job sizes around μ_y. This distributional framework simulates realistic scheduling scenarios, capturing interactions between job sizes and job features in a controlled, measurable way.

b) Lognormal job sizes: We also investigate lognormal job sizes to capture heavy-tailed or multiplicative effects [51]. Concretely, a positive random variable p_j^* follows a lognormal distribution if $\ln(p_j^*) \sim \mathcal{N}(\mu_\ell, \sigma_\ell^2)$ where μ_ℓ and σ_ℓ^2 are the mean and variance of the underlying normal distribution [47]. In such scenarios, most jobs are relatively short, but occasional tasks can be significantly longer than average [48], thus generating a right-skewed distribution. To simulate these lognormal sizes, one typically samples a normal deviate $v \sim \mathcal{N}(\mu_\ell, \sigma_\ell^2)$ and then exponentiates it to obtain p_j^*. This approach provides a clear link to the normal framework while allowing for heavy tails. The resulting skewed distribution can significantly impact scheduling decisions: a small fraction of "long" tasks may dominate total workload, testing the adaptability of learning-augmented schedulers.

Algorithm 1 Data generation for multi-normal distribution

Input: n, p, μ, Σ, N_{sets}
initialize mean vector $\mu = [0, 0, \ldots, 0, \mu_y]$
initialize $\Sigma \in \mathbb{R}^{(p+1)\times(p+1)}$
initialize noise levels: σ_X^2, $\sigma_{p^*}^2$
for each job set k from 1 to N_{sets} **do**
  sample $X^{(k)}$ and $p^{*(k)}$ from the multivariate normal distribution:

$$\left(X^{(k)}, p^{*(k)}\right) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

  for each job j from 1 to n **do**
    $X_j^{(k)} = X_j^{(k)} + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_X^2)$
    $p_j^{*(k)} = p_j^{*(k)} + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma_{p^*}^2)$
  end for
end for
return $\{X^{(1)}, \ldots, X^{(N_{\text{sets}})}\}$, $\{p^{*(1)}, \ldots, p^{*(N_{\text{sets}})}\}$

c) Exponential job sizes: Finally, we consider exponential job sizes to model memory-less, highly variable task durations [49], [52]. An exponentially distributed random variable p_j^* with rate λ has probability density $f(p_j^*) = \lambda \exp(-\lambda p_j^*), \quad p_j^* \geq 0$ This distribution is often employed in queuing theory when inter-arrival or service times exhibit no correlation between past and future [49]. Exponential job sizes serve as a stress test: if an algorithm can handle abrupt, unpredictable job lengths, it is likely robust in other volatile scenarios [46]. To generate exponential samples, we draw $u \sim \text{Uniform}(0, 1)$ and set $p_j^* = -\frac{1}{\lambda}\ln(u)$. As with lognormal, a small subset of jobs may still take disproportionately long times, making scheduling outcomes sensitive to the accuracy of job-size predictions. The multi-normal, lognormal, and exponential models collectively span a broad range of workload characteristics, offering a comprehensive environment to assess the performance of learning-augmented schedulers.

d) Justification for GPR with non-Gaussian distribution: Standard GPR assumes the target outputs follow a joint Gaussian distribution. However, empirical data often exhibit heavy-tailed or skewed patterns (e.g. Pareto, lognormal, exponential) [1]. GPR is still valid for job sizes drawn from lognormal or exponential distributions, as a GP with a universal kernel, RBF, can approximate any continuous function. Output transformations-log-scale or frameworks such as the Warped Gaussian Process, can further align skewed observations with Gaussian assumptions, preserving GPR's predictive capacity under these distributions.

C. GPR predictions

We employ GPR to predict job sizes based on the job features. We generate multi-normal distributed data: $\mathbf{X} \in \mathbb{R}^{n\times p}$ and $y \in \mathbb{R}^n$. GPR models the unknown function f mapping a feature vector $X \in \mathbb{R}^p$ to the job size $y \in \mathbb{R}^n$, assuming that

f follows a Gaussian process: $f(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$, where $m(\mathbf{x})$ is the mean function, and $k(\mathbf{x}, \mathbf{x}')$ is the covariance function, often modeled as the squared exponential kernel (or RBF kernel) [28], [30]: $k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp(-\frac{1}{2\ell^2}\|\mathbf{x}-\mathbf{x}'\|^2)$, where ℓ is the length scale hyper-parameter, controlling how quickly the correlation between points decays with distance. If ℓ is small, points must be very close to have a high correlation. $|\mathbf{x}-\mathbf{x}'|$ is the Euclidean distance between two feature vectors $\mathbf{x}$ and $\mathbf{x}'$. σ_f^2 is the signal variance determining its variance from its mean. The length scale can be in a diagonal matrix Λ, particularly when different features have different prediction levels: $k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x}')^\top \Lambda^{-1}(\mathbf{x} - \mathbf{x}'))$, where Λ is a diagonal matrix containing length scale values. This allows the kernel to account for different rates of change in each dimension, making the model more flexible and capable of capturing complex relationships. Then, we further assume that the observed job sizes are generated from the function $f(\mathbf{x})$ with additive Gaussian noise ϵ, i.e., $p_j^* = f(\mathbf{x}_j) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ and σ_n^2 is a pre-defined noise variance. The noise models real-world imperfections, such as variability in job execution environments or fluctuations in job processing requirements. In this formulation, $f(\mathbf{x})$ represents the underlying smooth function we wish to learn.

a) Prediction framework: The GPR model predicts job sizes by inferring the posterior distribution of the latent function f at a test point $\mathbf{x}_*$. Assuming Gaussianity and a fixed kernel structure (e.g., RBF with hyper-parameters $\ell, \sigma_f^2, \sigma_n^2$), the predictive distribution is given by: $f_* \mid \mathbf{X}, \mathbf{y}, \mathbf{x}_* \sim \mathcal{N}\left(\mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{y},\, k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*\right)$, where $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}_1), \ldots, k(\mathbf{x}_*, \mathbf{x}_n)]^\top$ is the covariance vector between the test point and the training points, and $\mathbf{K}$ is the covariance matrix of the training data defined as:

$\mathbf{K} = [k(\mathbf{x}, \mathbf{x}')]_{i,j=1}^n + \sigma_n^2 \mathbf{I}$ where $\mathbf{I}$ is the identity matrix. Term $\mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{y}$ corresponds to the contribution of the training data in shaping the prediction, capturing the relationship between the training inputs $\mathbf{X}$ and observed outputs $\mathbf{y}$. Term $k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{k}_*^\top \mathbf{K}^{-1}\mathbf{k}_*$ expresses the uncertainty associated with the prediction; as the quantity of observations increases, the influence of this term diminishes, reflecting reduced uncertainty in predictions. Consequently, error metrics such as MAE and RMSE improve. The prediction error diminishes as the number of training points increases, which leads to better generalization.

b) Hyperparameter tuning: The RBF length scale ℓ is a key hyper-parameter that can be tuned to optimize the model's performance. Using a simple grid search or Bayesian optimization over different values of ℓ, we ensure that the model can fit smooth and complex patterns in the data. Our methodology optimizes ℓ by evaluating model performance metrics on a validation set. Additionally, feature-specific length scales can also be learned; we test length scales ℓ by using Automatic Relevance Determination (ARD), which provides insights into which features are most influential and helps in further refining the model's complexity [53], [54].

c) Modeling procedure: For modeling the prediction cost, firstly, the generated data is split into training and test sets for each job set and across all job sets. Then, hyperparameter tuning is applied to adjust the RBF length scale, improving the model's performance and generalization to fit the underlying functions. In the controlled simulated environment, the number of training observations is iteratively increased to examine the influence of training set size on ML model accuracy. In contrast, the test set remains fixed across all job sets, allowing consistent model performance comparison. The trained GPR models are then used to predict job sizes. The job size predictions $\hat{\mathbf{y}}_{\text{test}} = \{\hat{p}_j\}_{j=1}^n$ are compared to the actual job sizes $\mathbf{y}_{\text{test}} = \{p_j^*\}_{j=1}^n$. This test set of jobs will also be used for scheduling shortly. The illustrative workflow for the prediction models is shown in Figure 2.

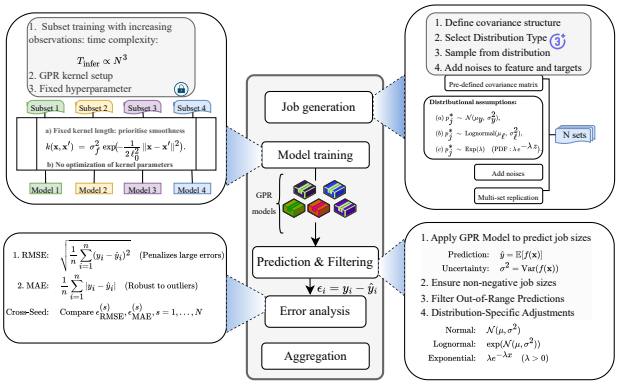

Fig. 2. Workflow for job size prediction: subset training with pre-assessed kernel hyperparameter, error analysis (RMSE/MAE), N job sets validation, adding noises via predefined distributions, and distribution-specific prediction adjustments.

D. Inference Time

Next, we quantify the inference time overhead of our GPR-based job size predictor. Inference time here refers to the cumulative latency to predict all test jobs after training, rather than a per-job prediction time. GPR training requires $O(n^3)$ time for n training points due to the Cholesky factorization (or inversion) of the $n \times n$ covariance matrix [30], and is performed offline during model fitting. After training, predicting the posterior mean for a new input has complexity $O(n)$, given precomputed training inverses. Although computing the predictive variance would cost up to $O(n^2)$ per test point, only the mean is required. The predictive mean for a test job x_* is given by: $\mu(x_*) = k(x_*, x_*)\,[k(x, x')]^{-1} y$ This computation reduces to a dot product

$$k(x_*, x_*) \cdot \Big([k(x, x')]^{-1} y\Big)$$

which is essentially a dot product between the n-dimensional covariance vector $k(x_*, x)$ and the weight vector. This dot product is the core computation for each prediction. For instance, if there are m test samples, the total inference time scales as $O(m, n)$ (linear in both the number of predictions and training points). In our study, the inference cost in GPR

concerning the number of observations follows a linear function: $T_{\text{inference}} = c \cdot N$ where N is the number of training points (observations) used to fit GPR, and c is a constant that depends on factors such as the number of features and hardware or software efficiencies.

IV. Optimization

We implement the SRPT, RR, SPJF, and Preferential Round Robin (PRR) [1]. We also propose a variant of PRR, named *dynamic preferential round robin (DPRR)*. Then, we design a scheduling simulator that runs these algorithms and considers inference costs for total completion time calculation.

A. Scheduling algorithms

a) SPJF: The SPJF [1] prioritizes jobs to implement scheduling policy according to the job size prediction for job j, $\hat{p}_j$, which is presented in Algorithm 2.

Algorithm 2 Shortest Predicted Job First (SPJF)

Input: job size prediction $\hat{\mathbf{p}} = \{\hat{p}_j\}_{j=1}^n$
Output: total completion time C_{SPJF}
initialize $C_{\text{SPJF}} \leftarrow 0$
initialize $t \leftarrow 0$
sort jobs by job size predictions such that $\hat{p}_{(1)} \leq \hat{p}_{(2)} \leq \cdots \leq \hat{p}_{(n)}$, where $\hat{p}_{(i)}$ is the i-th smallest value in $\hat{\mathbf{p}}$.
for each job $J_{(i)}$ in the sorted list, $i = 1, 2, \ldots, n$ **do**
    $t \leftarrow t + p^*_j$, where p^*_j is the actual job size of $J_{(i)}$
    $C_{\text{SPJF}} \leftarrow C_{\text{SPJF}} + t$
end for
return C_{SPJF}

TABLE I
Notation Definitions for Dynamic PRR with Inference (DPRR)

Notation	Description
n	Total number of jobs
q	Scheduling quantum length (time interval for re-evaluation)
θ	Error threshold triggering increased inference allocation
δ	Step size for adjusting λ_{inf} (typically 0.05-0.1)
α	Base weight for RR allocation (typically 0.3-0.7)
β	Decay rate for RR priority as predictions improve
γ	Sensitivity factor for error-based RR adjustment
M	Pre-trained GPR model for job size prediction
r_j	Remaining processing time for job j
U	Set of jobs not yet predicted
P	Set of jobs with available predictions
λ_{inf}	Dynamic fraction of quantum allocated to inference
λ_{rr}	Dynamic fraction allocated to round-robin
λ_{spjf}	Dynamic fraction allocated to SPJF
$\eta(j)$	Relative prediction error: $\lvert p^*_j - \hat{p}_j \rvert / p^*_j$
$\Phi(E, \text{Var})$	Error variance function: $\min(1, E \cdot \sqrt{\text{Var}})$
k	Current number of unfinished jobs
p	Job with smallest predicted remaining time in P
ϵ	Small positive constant prevent λ_{inf} reach boundaries

b) PRR: PRR [1] uses a parameterized scheduling model with $\lambda \in [0, 1]$ that continuously allocates processing power between two strategies. At any time with k active jobs, PRR runs the job with smallest predicted remaining processing time at rate $\lambda + \frac{1-\lambda}{k}$, while each other job runs at rate $\frac{1-\lambda}{k}$. This can be viewed as running SPJF at rate λ and RR at rate $(1-\lambda)$ in parallel. The algorithm achieves competitive ratio $\min\left\{\frac{1}{\lambda}\left(1 + \frac{2\eta}{n}\right), \frac{2}{1-\lambda}\right\}$, where $\eta = \sum_{j=1}^n |\hat{p}_j - p^*_j|$ is the total ℓ_1 prediction error. This yields $\frac{2}{1-\lambda}$-robustness and $\frac{1}{\lambda}$-consistency, when $\eta = 0$. Note that PRR cannot achieve both optimal robustness (2) and optimal consistency (1) simultaneously: any $\lambda > 0$ sacrifices robustness ($\frac{2}{1-\lambda} > 2$), while any $\lambda < 1$ sacrifices consistency ($\frac{1}{\lambda} > 1$ even with perfect predictions). The parameter λ provides a smooth trade-off: small λ prioritizes robustness: approaching RR's optimal bound of 2 as λ is close to 0, while large λ prioritizes consistency: approaching optimal performance as λ is close to 1 with good predictions, but with unbounded worst-case ratio). For example, $\lambda = 0.5$ yields a 4-robust, 2-consistent algorithm. Wei and Zhang [2] prove this trade-off is essentially tight through matching lower bounds.

c) DPRR: Building on PRR's robustness-consistency tradeoff, the Dynamic PRR with Inference (DPRR) introduces inference time awareness and online adaptation. Unlike PRR which operates continuously, DPRR discretizes time into scheduling quanta of length q to enable periodic re-evaluation and adjustment. Each quantum is divided among three activities: a fraction λ_{inf} for GPR inference, and the remaining $(1 - \lambda_{\text{inf}})$ split between round-robin (fraction λ_{rr}) and SPJF (fraction λ_{spjf}), where $\lambda_{\text{inf}} + \lambda_{rr} + \lambda_{spjf} = 1$. The adaptation mechanism uses the mean relative prediction error $E = \frac{1}{|\mathcal{C}|}\sum_{j \in \mathcal{C}} \eta(j)$ from completed jobs. When $E > \theta$, DPRR increases λ_{inf} by step δ to allocate more time for inference, potentially improving future predictions. Conversely, when predictions are accurate ($E \leq \theta$), it reduces inference overhead. The round-robin fraction is computed as $\lambda_{rr} = \min\{\alpha(|U|/n)^\beta + \gamma\Phi(E, \text{Var}), 1 - \lambda_{\text{inf}}\}$, where $\Phi(E, \text{Var}) = \min(1, E \cdot \sqrt{\text{Var}})$ combines error magnitude and variance. Parameters α, β, and γ control how aggressively DPRR shifts between strategies based on the fraction of unpredicted jobs and prediction quality. To address practical concerns: (i) inference time is handled by allocating at most Δt_{inf} per quantum, with partial progress tracked if inference doesn't complete; (ii) when $P = \emptyset$, only round-robin executes; (iii) to prevent λ_{inf} getting stuck at boundaries, practical implementations should use $\lambda_{\text{inf}} \in [\epsilon, 1 - \epsilon]$ for small $\epsilon > 0$; (iv) sorting uses predicted remaining time rather than initial predictions to account for partial execution. DPRR maintains robustness through its adaptive design. In worst-case scenarios with consistently poor predictions, it converges toward round-robin behavior (as λ_{rr} increases and λ_{spjf} decreases), achieving competitive ratio approaching $2/(1 - \lambda_{\text{inf}})$. With accurate predictions, it approaches SPJF's near-optimal performance. The per-quantum overhead of $O(n)$ for parameter updates is comparable to modern OS schedulers and justified when

inference costs are significant. Table I and Algorithm 3 present the complete specification.

B. Scheduling simulator

The simulator evaluates different scheduling algorithms by incorporating inference costs. For algorithms like SPJF and PRR that use predictions, the simulator first runs the GPR model M to generate predictions $\hat{\mathbf{p}}$, then adds the total inference cost to the scheduling cost. For DPRR, which integrates inference into its scheduling decisions, the algorithm handles both prediction and execution internally. The simulator is presented in Algorithm 4.

C. Performance measurement

We introduce an empirical performance metric tailored for average-case analysis, complementing traditional worst-case competitive ratios. Specifically, we define the performance ratio as the normalized total completion time of the scheduling simulator compared to an offline optimal baseline. Let C_{sim} denote the total completion time obtained by the S, including prediction inference costs, and let C_{SRPT} represent the offline optimal total completion time using the SRPT algorithm. The performance ratio R is defined as:

$$R = \frac{C_{\text{sim}}}{C_{\text{SRPT}}} \tag{1}$$

The metric quantifies how closely the simulated algorithm approaches optimal scheduling under realistic settings, used for our subsequent empirical evaluation and comparisons across different experimental settings.

V. Experimental results

A. Experiment Setup

We empirically evaluate our extended PRR [1] scheduling algorithm with prediction cost and the proposed DPRR. The experiments aim to quantify how incorporating machine-learned predictions influences scheduling performance. We outline the data generation process, the training/test protocol for the predictive model, and the evaluation methodology. All experiments were conducted on an Intel Core i5-12600KF CPU with 16GB RAM and NVIDIA GeForce RTX 3070 GPU (8GB VRAM).

a) Parameters: We generate $n = 1500$ jobs with $p = 5$ features each, grouped into $N_{\text{sets}} = 100$ job sets with no overlap between sets for robust evaluation. For each job set, we sample features and job sizes from a $(p+1)$-variate normal distribution with a mean vector with μ set as 2.5 and a predefined covariance matrix $\mathbf{\Sigma}$ capturing feature correlations and their link to job size. After sampling, we add small Gaussian noise to features and job sizes, respectively, clipped to ensure non-negativity. We implemented the algorithms in Python 3.8 using scikit-learn 1.0 for GPR. For DPRR, we used parameters: $\alpha = 0.5$, $\beta = 0.5$, $\gamma = 0.3$, $\theta = 0.3$, $\delta = 0.05$, and $q = 1.0$ seconds. For PRR, we used $\lambda = 0.75$, considered initial λ for DPRR.

Algorithm 3 Dynamic PRR with Inference (DPRR)

Input: n jobs, quantum $q > 0$, threshold $\theta > 0$, step $\delta > 0$, constants $\alpha, \beta, \gamma > 0$, trained GPR model M
Output: Total completion time C
$r_j \leftarrow p_j^*,\ \forall 1 \le j \le n$
$U \leftarrow \{1, \ldots, n\}$; $P \leftarrow \emptyset$ {unpredicted/predicted jobs}
$t \leftarrow 0,\ C \leftarrow 0,\ \lambda_{\text{inf}} \leftarrow 0.5,\ \lambda_{\text{rr}} \leftarrow 0.5,\ \lambda_{\text{spjf}} \leftarrow 0$
$\eta(j) \leftarrow 0,\ \forall j,\ \mathcal{C} \leftarrow \emptyset$ {set of completed jobs}
while $\sum_{j=1}^{n} r_j > 0$ **do**
  if $|\mathcal{C}| = 0$ **then**
    $E \leftarrow 0$; $\text{Var} \leftarrow 0$
  else
    $E \leftarrow \frac{1}{|\mathcal{C}|} \sum_{j \in \mathcal{C}} \eta(j)$; $\text{Var} \leftarrow \frac{1}{|\mathcal{C}|} \sum_{j \in \mathcal{C}} (\eta(j) - E)^2$
  end if
  if $E > \theta$ **and** $\lambda_{\text{inf}} < 1$ **then**
    $\lambda_{\text{inf}} \leftarrow \min\{\lambda_{\text{inf}} + \delta,\ 1\}$
  else if $E \le \theta$ **and** $\lambda_{\text{inf}} > 0$ **then**
    $\lambda_{\text{inf}} \leftarrow \max\{\lambda_{\text{inf}} - \delta,\ 0\}$
  end if
  $\lambda_{\text{rr}} \leftarrow \min\{\alpha(\frac{|U|}{n})^{\beta} + \gamma\Phi(E, \text{Var}), 1 - \lambda_{\text{inf}}\}$
  $\lambda_{\text{spjf}} \leftarrow \max\{0,\ 1 - \lambda_{\text{inf}} - \lambda_{\text{rr}}\}$
  $\Delta t_{\text{inf}} \leftarrow \lambda_{\text{inf}} q$; $\Delta t_{\text{rr}} \leftarrow \lambda_{\text{rr}} q$; $\Delta t_{\text{spjf}} \leftarrow \lambda_{\text{spjf}} q$
  if $U \neq \emptyset$ **and** $\Delta t_{\text{inf}} > 0$ **then**
    choose any $j \in U$; $t_{\text{inf}} \leftarrow$ time to infer job j using M
    $t_{\text{used}} \leftarrow \min\{t_{\text{inf}}, \Delta t_{\text{inf}}\}$; $t \leftarrow t + t_{\text{used}}$
    if $t_{\text{used}} \ge t_{\text{inf}}$ **then**
      $U \leftarrow U \setminus \{j\}$; $P \leftarrow P \cup \{j\}$ {Inference done}
      sort P by increasing predicted remaining time $\hat{r}_j$
    end if
  end if
  $k \leftarrow |\{j \mid r_j > 0, j \in U \cup P\}|$
  if $P \neq \emptyset$ **then**
    $p \leftarrow \arg\min_{j \in P: r_j > 0} \hat{r}_j$
  else
    $p \leftarrow$ null
  end if
  for each $j \in (U \cup P)$ with $r_j > 0$ **do**
    if $j \in P$ **and** $j = p$ **then**
      $\Delta t_j \leftarrow \min\{\Delta t_{\text{spjf}} + \frac{\Delta t_{\text{rr}}}{k}, r_j\}$
    else
      $\Delta t_j \leftarrow \min\{\frac{\Delta t_{\text{rr}}}{k}, r_j\}$
    end if
    $t \leftarrow t + \Delta t_j$; $r_j \leftarrow r_j - \Delta t_j$
    if $r_j = 0$ **then**
      $C \leftarrow C + t$; $\mathcal{C} \leftarrow \mathcal{C} \cup \{j\}$
      if $j \in P$ **then**
        $\eta(j) \leftarrow |p_j^* - \hat{p}_j| / p_j^*$
      end if
    end if
  end for
end while
return C

Algorithm 4 Scheduling simulator

Input: scheduler S, GPR model M, job features $\mathbf{X} = \{\mathbf{x}_j\}_{j=1}^n$, actual sizes $\{p_j^*\}_{j=1}^n$
Output: total completion time C_{total}
if scheduler S integrates inference (e.g., DPRR) **then**
  $C_{\text{total}} \leftarrow S(M, \mathbf{X}, \{p_j^*\})$
else
  $\hat{\mathbf{p}} \leftarrow M(\mathbf{X})$ {Predict all job sizes}
  $t_{\text{inf}} \leftarrow$ total inference time for all predictions
  $C_{\text{sched}} \leftarrow S(\hat{\mathbf{p}}, \{p_j^*\})$ {Schedule with predictions}
  $C_{\text{total}} \leftarrow C_{\text{sched}} + n \cdot t_{\text{inf}}$ {Add inference overhead}
end if
return C_{total}

b) Training protocol: We partition the generated jobs into a training set and a hold-out test set: 100 jobs form the test set, and the remaining jobs are used to train the GPR model. We vary the training set sizes incrementally each time, examining how additional observations improve predictive accuracy, which leads to better decision-driven scheduling performance.

c) Offline Optimal Baseline (SRPT): As a benchmark for completion times, we compare against the Shortest Remaining Processing Time (SRPT) schedule [32], which is offline-optimal given exact knowledge of true job sizes. Our experiments measure how closely the various learning-augmented algorithms approximate SRPT under imperfect predictions.

B. Job distribution

Figure 3 visualizes job size distributions across 10 representative job sets sampled from our 100 total sets. These 10 sets demonstrate the consistency and minor variations, verifying that data generation produces stable distributional properties with realistic random variations across different seeds. The Lognormal distribution skews right, forming a prominent peak near smaller values with a heavy tail. The Exponential distribution declines quickly with increasing job size. Each distribution's parameters produce a mean of 2.5: we set $\mu = 0.8$ and $\sigma = 0.5$ for the lognormal, satisfying $e^{\mu+\frac{\sigma^2}{2}} \approx 2.5$, and use a scale of 2.5 for the exponential.

C. Prediction error

Figure 4 shows prediction error surfaces using normally distributed job sizes. Both 3D error surfaces, for RMSE and MAE, show that the prediction error decreases as the number of observations increases. The improvements stabilize for larger training sets; diminishing returns cause progressively smaller error reductions as the data volume grows. The seemingly large absolute errors relative to the mean reflect the substantial variability in the data, which spans a large range across all job sets. We report normalized error metrics alongside absolute values to provide proper context for model performance assessment. Overall, more training data improve the model's predictions such that p_j^* approaches $\hat{p}_j$, validated by diverse job sets. For lognormal and exponential

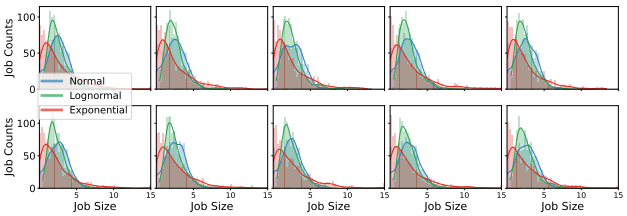

Fig. 3. Job size distributions for representative job sets (out of 100 total used in experiments), all sharing a mean of approximately 2.5: *(i)* normal $|\mathcal{N}(2.5, 1)|$ with light tails and near-symmetric shape, *(ii)* a lognormal $\text{Lognormal}(\mu = 0.8, \sigma = 0.5)$ that exhibits heavier right-skew while maintaining the same mean, and *(iii)* an exponential $\text{Exp}(\text{scale} = 2.5)$ with the memory-less property and variance 6.25. The similar trends confirm consistent generation of skewed versus symmetric distributions across all job sets used in our experiments.

job size distributions, while both distributions do show error reductions with more training data, the heavy-tailed nature leads to significantly higher error levels than typical normal data, particularly in smaller training sets.

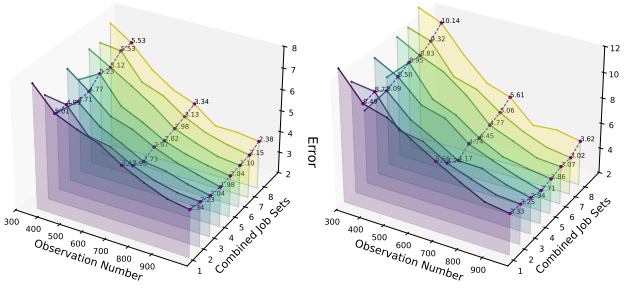

Fig. 4. Three-dimensional error surfaces showing RMSE (left) and MAE (right) as functions of training observation count and aggregated job set groups. Each of the 8 surface layers represents averaged results from 10 job sets out of 80 total. The convergence of surfaces to stable lower values at higher observation counts illustrates diminishing returns in prediction improvement, with the steepest error reductions occurring in the initial phase.

D. Inference cost

To evaluate our model's runtime overhead, we measure the prediction inference time of the Gaussian Process with fixed RBF kernel, where each reported value represents the mean of 10 repeated predictions to ensure timing stability. From Table II, the inference time increases approximately linearly from minimum n to maximum n for making predictions. Meanwhile, the standard deviation remains on the same millisecond scale, showing consistent performance across runs. These observed times confirm that the cost of computing predictions in GPR scales $O(n)$, making it tractable for moderate n in many real-world applications. As discussed, data has high variability. To quantify the prediction error and inference cost trade-off, we have utilized approximate mean normalized RMSE and MAE [56], denoted as M-NRMSE (Mean Normalized RMSE) and M-NMAE (Mean Normalized MAE), where $\bar{p}$ represents the estimated mean job size.

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(p_j^* - \hat{p}_j\right)^2}}{\bar{p}} \quad \text{NMAE} = \frac{\frac{1}{n}\sum_{i=1}^{n}\left|p_j^* - \hat{p}_j\right|}{\bar{p}}$$

TABLE II
INFERENCE TIME AND MEAN-NORMALIZED ERROR METRICS AT VARIOUS OBSERVATION SIZES.

Obs	Mean Inf. Time (s)	Inf. Std (s)	M-NRMSE	M-NMAE
200	0.0025	0.0021	3.4171	2.0492
300	0.0037	0.0030	3.5737	2.0687
400	0.0051	0.0036	2.7568	1.6326
500	0.0063	0.0035	2.2820	1.3933
600	0.0077	0.0030	1.7753	1.1470
700	0.0094	0.0036	1.3904	0.9622
800	0.0114	0.0038	1.2029	0.8629
900	0.0138	0.0057	1.0945	0.8028
1000	0.0159	0.0070	1.0018	0.7497
1100	0.0178	0.0063	0.9476	0.7193
1200	0.0200	0.0072	0.9256	0.7026
1300	0.0222	0.0075	0.8904	0.6811
1400	0.0240	0.0103	0.8828	0.6741

In addition to mean-normalized metrics, we employ range-based normalized metrics, R-NRMSE (Range-based Normalized RMSE) and R-NMAE (Range-based Normalized MAE), to provide scale-invariant error measures that are robust to outliers [57]. Specifically: $\text{R-NRMSE} = \frac{\sqrt{\frac{1}{n}\sum_{j=1}^{n}\left(p_j^* - \hat{p}_j\right)^2}}{\left(p_{\max} - p_{\min}\right)}$, $\text{R-NMAE} = \frac{\frac{1}{n}\sum_{j=1}^{n}\left|p_j^* - \hat{p}_j\right|}{\left(p_{\max} - p_{\min}\right)}$. As reported, while larger datasets incur greater computational overhead during prediction, the prediction error rates decline as more observations are processed. Both mean-normalized and range-normalized RMSE and MAE metrics have marginal improvements beyond a threshold, illustrating diminishing returns, which is shown in Figure 5 and Table II. The results empirically confirm the inherent trade-off between inference cost and prediction accuracy. Also, the observed inference time does not vary across different job size distributions: even if the job-processing times are highly skewed or heavy-tailed, the algorithmic path for GPR remains the same. The kernel matrix is formed and factorized based on distance computations between feature points, not the statistical distribution of the targets.

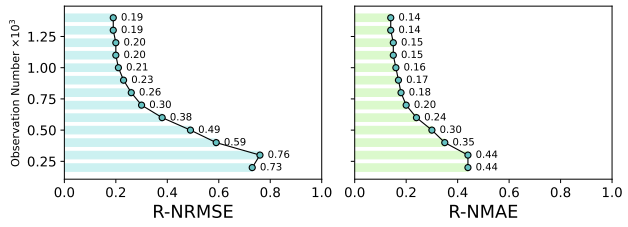

Fig. 5. Range-based errors across increasing observation numbers. The left panel plots range-based normalized RMSE, and the right panel plots range-based normalized MAE, each declining as more observations are used.

E. Empirical performance ratio

We evaluate the empirical competitive ratio (performance ratio) computed as the average ratio of total job completion time to the off-line optimum, confirming statistical stability. Different job distributions were analyzed under the scheduling baselines outlined below.

a) SPJF: SPJF exhibits a consistently declining competitive ratio as the training set grows, with minor fluctuations and occasional local peaks around smaller sample sizes. For normally distributed job sizes, the total completion time competitive ratio achieves a near-optimal point around 1200 observations, and the diminishing returns occur at an earlier time. After the minimum, inference overhead is dominating and the total completion time competitive ratio starts to increase. A clear U-shape trend also appears for lognormal and exponential job size distributions, respectively. For heavy-tailed data, the competitive ratio is higher across all observation numbers, and the empirical ratio decreases slowly, reflecting greater difficulty in predicting large outliers. However, exponential data performs slightly better than lognormal one. The overall U-shape curve indicates that an intermediate-scale training set balances prediction accuracy and inference cost, yielding near-optimal performance for SPJF.

b) PRR: The PRR competitive ratio initially surpasses that of SPJF when only minor training samples (e.g., less than 200) are used. As more observations become available, the ratio markedly declines, attaining a minimum around 1000 observations. Beyond that, the ratio climbs slightly, settling in the end. This pattern indicates that PRR and SPJF have a similar U-shape trend: enhanced prediction significantly improves PRR, yet eventually encounters diminishing returns as inference overhead intensifies. We explain that PRR operates as a preemptive scheduling policy that allocates CPU resources among jobs in quanta, giving extra priority to the job with the smallest predicted remaining time but ensuring no job is starved. SPJF, theoretically, achieves superior performance when predictions exhibit high fidelity. Referring to Table III, our experiment validates the theory: while small observation numbers still yield good predictions, SPJF outperforms PRR under conditions where robustness to prediction inaccuracies is not required.

c) DPRR: We focus DPRR evaluation on single-machine scenarios to clearly demonstrate the benefits of dynamic λ adjustment without confounding effects from parallelization. The empirical competitive ratio of DPRR scheduling consistently improves as the number of observations increases. Unlike the clear U-shape for static PRR with fixed λ, DPRR exhibits a stabilized trend after moderate training without inference cost dominance. The empirical performance ratio is shown in Table III. This indicates that dynamically adjusting resource allocation between inference cost and execution time based on prediction fidelity effectively mitigates the large inference cost, particularly at larger observation scales. When early mispredictions, high η_j, are detected, DPRR reverts toward equal time-slicing. Then, predictions become more accurate, DPRR allocates less time for inference, and more CPU quantum to SPJF. So, it achieves a notably better competitive ratio at higher observation counts compared with PRR due to adaptive prioritization of predicted-shortest jobs. Also, DPRR

TABLE III
COMPARISON OF COMPETITIVE RATIOS OF SCHEDULING ALGORITHMS. COLUMNS REPRESENT OBSERVATION NUMBERS (n_i), WITH $n_i = 100(i+1)$.

Algorithm	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}	n_{11}	n_{12}	n_{13}
SPJF	1.4381	1.4354	1.4344	1.4369	1.4348	1.4285	1.4204	1.4188	1.4165	1.4152	**1.4149**	1.4151	1.4166
PRR	1.7308	1.7270	1.7239	1.7217	1.7216	1.7198	1.7177	1.7136	**1.7124**	1.7104	1.7111	1.7138	1.7186
DPRR	1.7321	1.7143	1.7050	1.6861	1.6801	1.6568	1.6482	**1.6369**	1.6418	1.6388	1.6393	1.6410	1.6444

can mitigate heavy tail's negative effect better than PRR, thus making their relative advantage more pronounced.

d) Multiple machines: In the multi-machine (m=5) scenario, SPJF achieves significantly improved competitive ratios compared to the single-machine scenario. This improvement is consistent across all observation sizes, highlighting that distributing workload over multiple machines effectively reduces the total completion time. Similarly, PRR demonstrates notable improvements when extended to multiple machines. Such improvements underline that PRR scheduling benefits even more significantly from multi-machine parallelism, likely due to effective balancing between predicted shortest jobs and equitable CPU sharing. The results could be explained by with multiple servers working in parallel, the scheduler can better distribute load and avoid extreme delays [58]. For example, in the classic non-clairvoyant scheduling model, a simple RR is 2-competitive on identical parallel machines, while for a single machine, no deterministic scheduler can achieve a constant competitive ratio in the worst case [59].

e) Non-zero release time: The expected inter-arrival time is $E[\Delta r] \approx 0.033$ seconds between each job. Consequently, both algorithms exhibit consistently lower (improved) competitive ratios when jobs arrive sequentially compared to the zero-release setting, as allowing the system to distribute the inference overhead throughout the scheduling horizon. The empirical results are shown in Table III and Figure 6, validated by [59]: they report that if jobs are spaced out in time, the online algorithm's competitive ratio is close to optimal. Both performance curves are U-shaped, being consistent with zero-release time scenarios. The out-performance is primarily because with staggered arrivals, the relative overhead cost of scheduling diminishes.

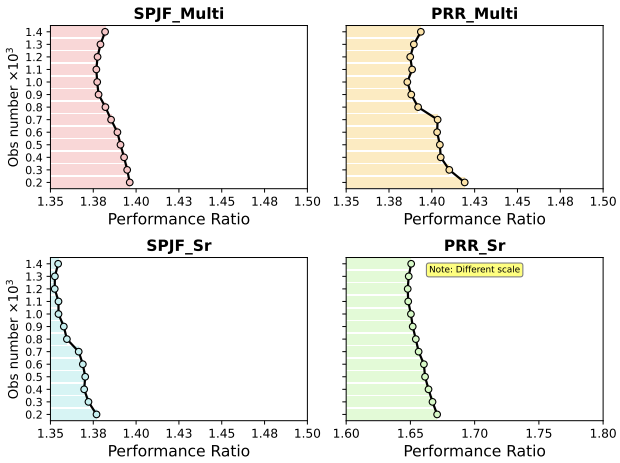

Fig. 6. Comparison of empirical competitive ratios across scheduling algorithms (SPJF and PRR) under Multi-Machine and Sequential Release, in terms of non-zero release time, scenarios at varying observation sizes.

VI. CONCLUSIONS

Our study addresses the critical trade-off between inference costs and objective function optimization in learning-augmented scheduling algorithms. Specifically, we consider minimizing the total completion time and propose a simulation method that leverages GPR to provide predictions of job processing times, alongside explicit consideration of inference overhead. We implement state-of-the-art algorithms and introduce DPRR, which dynamically adjusts scheduling parameters. Performance ratios are obtained under various experimental settings, including varying data distributions, multiple-machine setups, and sequential job releases. The results for both SPJF and PRR reveal a clear trend: total completion time decreases with moderately accurate predictions when inference costs are low but eventually increases as inference costs dominate, despite more accurate predictions being produced. DPRR outperforms PRR in scenarios with higher inference costs by dynamically balancing inference and execution time based on prediction fidelity.

A. Limitations

Our study relies on simulated datasets intentionally to isolate the interplay between inference costs, prediction accuracy, and scheduling performance. Real-world scenarios involve additional complexities such as model selection and nonlinear inference-accuracy relationships. Our experiments employ relatively moderate dataset sizes, which may limit the generalization of findings to large-scale industrial or high-performance computing environments. We might need to conduct experiments with cooperated company for realistic data. We evaluate DPRR only in single-machine settings as it was specifically designed for observed prediction quality within a single processing queue, would require fundamental redesign for multi-machine coordination.

B. Future work

Future work should consider other time-sensitive metrics, such as total response time or makespan. Additionally, examining the validity of assumptions underlying dataset generation and GPR in practical settings merits. Future research should validate these findings on real-world datasets and employ predictive models with broader applicability.

ACKNOWLEDGMENT

Appreciate all reviews provided, and authors have carefully address the concerns. Professor Albert Zomaya and Dr Wei Li would like to acknowledge the support of the Australian Research Council Discovery Grant (DP200103494). Zhiyun Jiang would like to express deep gratitude to Si Wu for his invaluable support and encouragement throughout this research endeavor. She also acknowledges the continuous support from family and friends.

REFERENCES

[1] R. Kumar, M. Purohit, and Z. Svitkina, "Improving online algorithms via ML predictions," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 9684–9693.

[2] A. Wei and F. Zhang, "Optimal robustness-consistency trade-offs for learning-augmented online algorithms," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 8042–8053, 2020.

[3] M. Mitzenmacher, "Scheduling with predictions and the price of misprediction," 2019, *arXiv:1902.00732*.

[4] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, 2010, pp. 495–504.

[5] Y. Fan, P. Rich, W. E. Allcock, M. E. Papka, and Z. Lan, "Trade-off between prediction accuracy and under-estimation rate in job runtime estimates," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2017, pp. 530–540.

[6] R. Wolski, "Experiences with predicting resource performance on-line in computational grid settings," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 41–49, 2003.

[7] E. Gaussier, D. Glesser, V. Reis, and D. Trystram, "Improving backfilling by using machine learning to predict running times," in *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis (SC)*, 2015, pp. 1–10.

[8] Z. Liu, A. K. Nath, X. Ding, H. Fu, M. M. Khan, and W. Yu, "Multivariate modeling and two-level scheduling of analytic queries," *Parallel Comput.*, vol. 85, pp. 66–78, 2019.

[9] S.-l. Jiang, M. Liu, J.-h. Lin, and H.-x. Zhong, "A prediction-based online soft scheduling algorithm for real-world steelmaking-continuous casting production," *Knowledge-Based Syst.*, vol. 111, pp. 159–172, 2016.

[10] S. Im, J. Kulkarni, and K. Munagala, "Competitive algorithms from competitive equilibria: Non-clairvoyant scheduling under polyhedral constraints," *J. ACM*, vol. 65, no. 1, pp. 1–33, 2017.

[11] J. Cohen, C. Dürr, and T. Nguyen Kim, "Non-clairvoyant scheduling games," *Theory Comput. Syst.*, vol. 49, pp. 3–23, 2011.

[12] M. Harchol-Balter, "Task assignment with unknown duration," in *Proc. 20th IEEE Int. Conf. Distributed Computing Systems*, 2000, pp. 214–224.

[13] S. T. Maguluri and R. Srikant, "Scheduling jobs with unknown duration in clouds," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1938–1951, 2013.

[14] B. DasGupta and M. A. Palis, "Online real-time preemptive scheduling of jobs with deadlines," in *Proc. 3rd Int. Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, 2000, pp. 96–107.

[15] H. Hoogeveen, M. Skutella, and G. J. Woeginger, "Preemptive scheduling with rejection," *Math. Program.*, vol. 94, pp. 361–374, 2003.

[16] Y. Wang, J. Tan, W. Yu, L. Zhang, X. Meng, and X. Li, "Preemptive ReduceTask scheduling for fair and fast job completion," in *Proc. 10th Int. Conf. Autonomic Computing (ICAC)*, 2013, pp. 279–289.

[17] L. Epstein and A. Levin, "Robust algorithms for preemptive scheduling," *Algorithmica*, vol. 69, pp. 26–57, 2014.

[18] R. Graham, E. Lawler, J. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Discrete Optimization II*, P. Hammer, E. Johnson, and B. Korte, Eds. Elsevier, 1979, pp. 287–326.

[19] O. Holthaus, "Scheduling in job shops with machine breakdowns: an experimental study," *Comput. Ind. Eng.*, vol. 36, no. 1, pp. 137–162, 1999.

[20] R. L. Cunha, E. R. Rodrigues, L. P. Tizzei, and M. A. Netto, "Job placement advisor based on turnaround predictions for HPC hybrid clouds," *Future Gener. Comput. Syst.*, vol. 67, pp. 35–46, 2017.

[21] H. Li, D. Groep, and L. Wolters, "Efficient response time predictions by exploiting application and resource state similarities," in *Proc. 6th IEEE/ACM Int. Workshop on Grid Computing*, 2005, pp. 8–15.

[22] M. Khan, Y. Jin, M. Li, Y. Xiang, and C. Jiang, "Hadoop performance modeling for job estimation and resource provisioning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 441–454, 2015.

[23] L. Zhou, X. Zhang, W. Yang, Y. Han, F. Wang, Y. Wu, and J. Yu, "Prep: Predicting job runtime with job running path on supercomputers," in *Proc. 50th Int. Conf. Parallel Processing (ICPP)*, 2021, pp. 1–10.

[24] S. Mustafa, I. Elghandour, and M. A. Ismail, "A machine learning approach for predicting execution time of Spark jobs," *Alexandria Eng. J.*, vol. 57, no. 4, pp. 3767–3778, 2018.

[25] T. Li, J. Tang, and J. Xu, "Performance modeling and predictive scheduling for distributed stream data processing," *IEEE Trans. Big Data*, vol. 2, no. 4, pp. 353–364, 2016.

[26] P. A. Dinda, "A prediction-based real-time scheduling advisor," in *Proc. 16th Int. Parallel and Distributed Processing Symposium (IPDPS)*, 2002, p. 8.

[27] A. M. Girgis, J. Park, M. Bennis, and M. Debbah, "Predictive control and communication co-design via two-way Gaussian process regression and AoI-aware scheduling," *IEEE Trans. Commun.*, vol. 69, no. 10, pp. 7077–7093, 2021.

[28] A. Yadav, R. Bareth, M. Kochar, M. Pazoki, and R. A. E. Sehiemy, "Gaussian process regression-based load forecasting model," *IET Gener. Transm. Distrib.*, vol. 18, no. 5, pp. 899–910, 2024.

[29] M. Deisenroth and J. W. Ng, "Distributed Gaussian processes," in *Proc. 32nd Int. Conf. Machine Learning (ICML)*, 2015, pp. 1481–1490.

[30] H. Liu, Y.-S. Ong, X. Shen, and J. Cai, "When Gaussian process meets big data: A review of scalable GPs," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 11, pp. 4405–4423, 2020.

[31] D. Flam-Shepherd, J. Requeima, and D. Duvenaud, "Mapping Gaussian process priors to Bayesian neural networks," in *Proc. NIPS Bayesian Deep Learning Workshop*, vol. 3, pp. 1–7, 2017.

[32] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Oper. Res.*, vol. 16, no. 3, pp. 687–690, 1968.

[33] S. Im, R. Kumar, M. M. Montazer Qaem, and M. Purohit, "Non-clairvoyant scheduling with predictions," *ACM Trans. Parallel Comput.*, vol. 10, no. 4, pp. 1–26, 2023.

[34] Z. Benomar and V. Perchet, "Non-clairvoyant scheduling with partial predictions," *arXiv preprint arXiv:2405.01013*, 2024.

[35] R. Motwani, S. Phillips, and E. Torng, "Nonclairvoyant scheduling," *Theor. Comput. Sci.*, vol. 130, no. 1, pp. 17–47, 1994.

[36] L. Becchetti and S. Leonardi, "Non-clairvoyant scheduling to minimize the average flow time on single and parallel machines," in *Proc. 33rd ACM Symp. Theory of Computing (STOC)*, 2001, pp. 94–103.

[37] A. Uzunoglu, C. Gahm, S. Wahl, and A. Tuma, "Learning-augmented heuristics for scheduling parallel serial-batch processing machines," *Comput. Oper. Res.*, vol. 151, Art. 106122, 2023.

[38] S. Xu, P. Liu, R. Wang, and S. S. Panwar, "Realtime scheduling and power allocation using deep neural networks," in *Proc. 2019 IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2019, pp. 1–5.

[39] T. Zhao, W. Li, and A. Y. Zomaya, "Real-time scheduling with predictions," in *Proc. 2022 IEEE Real-Time Systems Symp. (RTSS)*, 2022, pp. 331–343.

[40] A. Coles, E. Karpas, A. Lavrinenko, W. Ruml, S. E. Shimony, and S. Shperberg, "Planning and acting while the clock ticks," in *Proc. Int. Conf. Automated Planning and Scheduling*, vol. 34, 2024, pp. 95–103.

[41] M. Cashmore, A. Coles, B. Cserna, E. Karpas, D. Magazzeni, and W. Ruml, "Temporal planning while the clock ticks," in *Proc. Int. Conf. Automated Planning and Scheduling*, vol. 28, 2018, pp. 39–46.

[42] C. N. Potts and M. Y. Kovalyov, "Scheduling with batching: a review," *Eur. J. Oper. Res.*, vol. 120, no. 2, pp. 228–249, 2000.

[43] C. Wang, C. Liu, Z.-h. Zhang, and L. Zheng, "Minimizing the total completion time for parallel machine scheduling with job splitting and learning," *Comput. Ind. Eng.*, vol. 97, pp. 170–182, 2016.

[44] S. Im, R. Kumar, M. M. Montazer Qaem, and M. Purohit, "Non-clairvoyant scheduling with predictions," in *Proc. 33rd ACM Symp. Parallelism in Algorithms and Architectures (SPAA '21)*, 2021, pp. 285–294.

[45] E. Bampis, K. Dogeas, A. Kononov, G. Lucarelli, and F. Pascual, "Scheduling with untrusted predictions," in *Proc. 31st Int. Joint Conf. Artificial Intelligence (IJCAI)*, 2022, pp. 4581–4587.
[46] W. Scott, P. Frazier, and W. Powell, "The correlated knowledge gradient for simulation optimization of continuous parameters using Gaussian process regression," *SIAM J. Optim.*, vol. 21, no. 3, pp. 996–1026, 2011.
[47] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.
[48] O. Holthaus, "Scheduling in job shops with machine breakdowns: an experimental study," *Comput. Ind. Eng.*, vol. 36, no. 1, pp. 137–162, 1999.
[49] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 1st ed., 2013.
[50] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*. Springer, 2014.
[51] S. Gualandi and G. Toscani, "Call center service times are lognormal: a Fokker–Planck description," *Math. Models Methods Appl. Sci.*, vol. 28, no. 8, pp. 1513–1527, 2018.
[52] J. Shortle, J. Thompson, D. Gross, and C. Harris, *Fundamentals of Queueing Theory*, 5th ed. Wiley, 2018.
[53] H. Liu, J. Cai, Y.-S. Ong, and Y. Wang, "Understanding and comparing scalable Gaussian process regression for big data," *Knowledge-Based Syst.*, vol. 164, pp. 324–335, 2019.
[54] A. Stuke, P. Rinke, and M. Todorović, "Efficient hyperparameter tuning for kernel ridge regression with Bayesian optimization," *Mach. Learn.: Sci. Technol.*, vol. 2, no. 3, Art. 035022, 2021.
[55] J. Gardner, G. Pleiss, R. Wu, K. Weinberger, and A. Wilson, "Product kernel interpolation for scalable Gaussian processes," in *Proc. Int. Conf. Artificial Intelligence and Statistics (AISTATS)*, 2018, pp. 1407–1416.
[56] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, no. 4, pp. 679–688, 2006.
[57] I. T. Jolliffe and D. B. Stephenson, "Forecast verification: a practitioner's guide in atmospheric science," 2nd ed. John Wiley & Sons, 2012.
[58] B. Moseley and S. Vardi, "The efficiency-fairness balance of round robin scheduling," *Operations Research Letters*, vol. 50, no. 1, pp. 20–27, 2022.
[59] E. Bampis, K. Dogeas, A. V. Kononov, G. Lucarelli, and F. Pascual, "Scheduling with untrusted predictions," in *Proc. International Joint Conference on Artificial Intelligence (IJCAI)*, Jul. 2022, pp. 4581–4587.
[60] J. L. Folks and R. S. Chhikara, "The inverse Gaussian distribution and its statistical application—a review," *J. Roy. Stat. Soc. B Stat. Methodol.*, vol. 40, no. 3, pp. 263–275, 1978.
[61] J. P. Lehoczky, "Real-time queueing network theory," in *Proc. Real-Time Syst. Symp.*, 1997, pp. 58–67.