LEARNING TO OPTIMIZE NON-CLAIRVOYANT SCHEDULING VIA BEHAVIORAL CLONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale cluster systems must schedule jobs whose processing times are unavailable at decision time, a challenge known as *non-clairvoyant scheduling*. Learning-augmented scheduling has been applied to this problem, feeding machine-learned predictions of job sizes (processing time) into classical online algorithms. However, it collapses the scheduling decision into a single predicted scalar, and the precision of induced ordering only depends on the prediction error of the processing time. In contrast, behavioral cloning has been shown to be a simple yet effective learning-to-schedule method based on imitation learning. In this paper, we propose a behavioral-cloning framework for non-clairvoyant scheduling in both the single-machine and the parallel-machine settings. Our main contributions are as follows. (1) We directly learn the scheduling policy rather than feeding predicted job sizes into a classical online algorithm, so that the scheduling decision no longer depends on the single scalar of predicted processing time only. (2) We address the variable action space that both settings share an attention-based pointer mechanism, which assigns scores to an arbitrary number of pending jobs at every decision step. (3) We conduct experiments on two practical production workloads: ATLAS and AI Spot TRAces (ASTRA), which we construct from the publicly released trace of a production GPU cluster—spanning 463,279 jobs whose size distribution is far heavier-tailed than ATLAS. The learned policy outperforms the strongest learning-augmented baselines on total completion time, maximum stretch, and makespan.

1 INTRODUCTION

Millions of jobs have to be scheduled in modern cluster systems without knowing their processing times at submission, known as the *non-clairvoyant* scheduling problem (Motwani et al., 1994). It rules out optimal clairvoyant strategies such as Shortest Remaining Processing Time (SRPT) (Schrage, 1968), and leaves classical non-clairvoyant policies such as Round-Robin and FIFO with only weak worst-case guarantees (Motwani et al., 1994). To address this problem, machine learning methods have been applied to scheduling, an approach which is now known as *learning-augmented* scheduling (Purohit et al., 2018).

Learning-augmented scheduling supplies predicted job sizes (processing times) to classical online algorithms and has since been generalized to a broad range of scheduling objectives: queuing systems (Mitzenmacher, 2020), parallel-machine makespan (Zhao & Zomaya, 2026), and the fairness-oriented max-stretch objective (Zhao et al., 2025). The scheduling performance is expected to depend on the accuracy of those predictions, however, it depends not on the absolute accuracy of the prediction but on the relative ordering that prediction induces among jobs (Im et al., 2021). For example, even if the predictions of job sizes are with errors, the scheduling results will remain unchanged as long as the relative job sizes are still with the correct orders (Lindermayr & Megow, 2022). Metrics such as total completion time, makespan, and max-stretch are greatly affected in worst cases where the predicted job sizes are with small errors but the relative job sizes are incorrect. The total completion time is provably sensitive to small perturbations under the standard error measure; makespan admits only smoothness bounds that grow with the prediction error; max-stretch admits only weak robustness (Im et al., 2021; Zhao & Zomaya, 2026; Zhao et al., 2025). Meanwhile, *learning to schedule* has also been applied to a broad range of scheduling problems, in which a neural network maps the current system state directly to the next scheduling decision (Bengio et al.,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

2021; Mazyavkina et al., 2021; Mao et al., 2019). This approach has proven effective in job-shop, flow-shop, and Dynamic Acyclic graph (DAG) scheduling, where learned policies discover their own decision rules from data and routinely match or surpass hand-crafted heuristics whose design demands substantial domain expertise (Zhang et al., 2020; Corsini et al., 2024; Jeon et al., 2023). However, these applications typically know the processing time of each job before scheduling: the known processing times are included in coefficient matrices for job-shop and flow-shop (Pinedo, 2008; Mao et al., 2019; Li et al., 2024), and in node durations for DAGs, on instances with at most a few hundred jobs. In modern computing systems such as AI data centers (Weng et al., 2022), the processing times are usually unavailable and the learning-to-schedule framework cannot be directly applied (Gu et al., 2019), where far more jobs are pending in most time.

In this paper, we propose a behavioral-cloning framework for non-clairvoyant scheduling. Our main contributions can be summarized as follows. (1) We learn the policy directly from an expert policy, instead of routing predictions through a classical online algorithm. The supervision target becomes the scheduling decision at each state, with the predicted size as an auxiliary objective; the ordering is not determined by any single predicted value. (2) We handle the action space whose size varies with the pending set and reaches thousands of jobs, using an attention-based pointer mechanism that scores any number of pending jobs against a summary of the whole pending set. (3) We construct ASTRA, a job-level scheduling workload from a recent Alibaba production GPU-cluster trace. Its jobs are large-model tasks and its sizes are far more heavy-tailed than ATLAS's—the 99th percentile is $3{,}700\times$ the median, against $106\times$. The learned policy outperforms the strongest learning-augmented baselines on both workloads.

2 Related Work

Learning-augmented scheduling. Machine-learned predictions of job sizes are fed into online algorithms while preserving worst-case guarantees, following the consistency–robustness framework (Purohit et al., 2018; Wei & Zhang, 2020). First, for total completion time on a single machine, shortest-predicted-job-first (SPJF) and Preferential Round-Robin (PRR) were introduced as the first prediction-based rules (Purohit et al., 2018), and the underlying error measure was later refined to better match the objective (Im et al., 2021). Second, researchers also consider the fairness-oriented max-stretch objective (Zhao et al., 2025), which bounds the ratio of each job's response time to its job size. Third, the framework has been extended to the parallel-machine makespan (Zhao et al., 2022; Zhao & Zomaya, 2026), where a variant of the longest-processing-time-first (LPT) algorithm was shown to be consistent, smooth, and robust (Bampis et al., 2023). Beyond these objectives, the framework has also been studied under partial predictions, where only a subset of job sizes is predicted (Benomar & Perchet, 2024). However, these methods restrict the scheduling decision to the ordering induced by the predicted sizes. To address this, we learn the scheduling policy directly, deciding from the system state rather than from an ordering of jobs by predicted processing times.

Learning to schedule. A neural network serves as the scheduling policy, turning the current system state into the next scheduling decision. This approach has been pursued under several training paradigms (Bengio et al., 2021; Mazyavkina et al., 2021). The most widely used is reinforcement learning, where the policy is optimized against a scheduling objective through reward signals. Decima learns to schedule data-processing DAGs over clusters (Mao et al., 2019); L2D learns dispatching rules for job-shop problems (Zhang et al., 2020). However, such reinforcement learning based methods require a carefully designed reward and long training before the network converges. To relieve the dependence on reward designs, self-labeling variants let the model supervise itself, as in SLIM, which generates pseudo-labels for job-shop instances (Corsini et al., 2024), and multi-policy optimization with self-evolution (Choi et al., 2025). Alternatively, imitation learning replaces the model's own labels with those of an expert policy. The simplest instance is behavioral cloning, which fits the policy by supervised learning on state–action pairs collected from the expert, and has been applied to job-shop and flow-shop scheduling (Li et al., 2024; Lee & Kim, 2025). However, these methods assume that processing times are known to the scheduler (Pinedo, 2008) and are mostly evaluated on synthetic instances with at most hundreds of jobs (Zhang et al., 2020; Corsini et al., 2024; Li et al., 2024), whereas our non-clairvoyant scheduling must deal with thousands of jobs in most cases. To address these problems, we provide an end-to-end non-clairvoyant scheduling policy driven by an attention-based pointer mechanism, without the known processing time and

without complicated reward design. The proposed method is evaluated using two practical production datasets.

3 Problem Formulation

We consider a set of n independent jobs $\{J_1, \ldots, J_n\}$ to be scheduled on a single machine or on multiple parallel machines. For each job J_j, the scheduler observes only its job feature vector $\mathbf{x}_j$, while the true processing time p_j^* remains unknown throughout execution. In the single-machine setting, jobs arrive online at their release times r_j and preemption is allowed. A job is pending from its release until completion. The scheduler may launch or resume a pending job at any time, and a preempted job returns to the pending set. The objective is to minimize both the total completion time $\sum_j C_j$ and the maximum stretch $\max_j S_j$, where C_j is the completion time of job J_j and $S_j = \frac{C_j - r_j}{p_j^*}$ is its response time relative to its size. In the parallel-machine setting, jobs are released simultaneously and are non-preemptive, and each job is assigned to one machine and runs to completion. The objective is to minimize the makespan $C_{\max} = \max_j C_j$, which equals the maximum machine load. To this end, we learn a policy π_θ that directly selects the next job to process at each decision step from the current state comprising the pending jobs and the runtime information of the system. The action space is the pending set, which varies at every decision step as jobs arrive and complete. Take the ATLAS dataset as an example. For single machine systems, as shown in Figure 1, the number of pending jobs fluctuates widely across decision steps, and the peak of production workloads can reach thousands of jobs. An illustrative example covering both single-machine and parallel-machine scenarios is given in Appendix B.

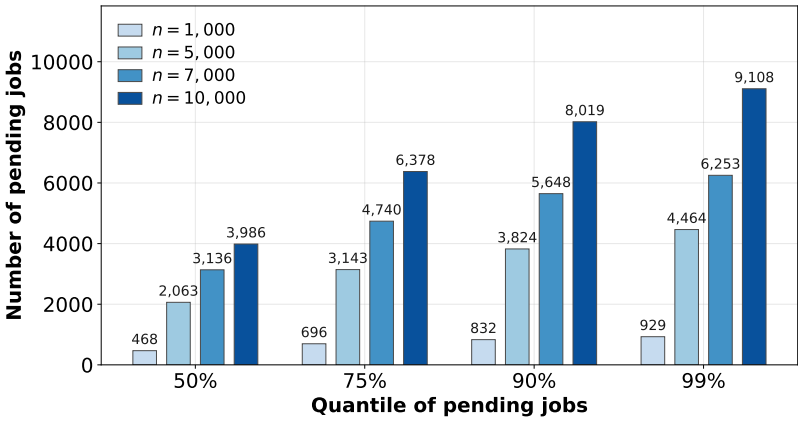


Figure 1: Number of pending jobs in the single-machine setting, for scheduling instances of $n =$ 1,000, 5,000, 7,000, and 10,000 jobs. Each bar reports this count at the 50th, 75th, 90th, and 99th percentiles across decision steps.

4 Methodology

In this section, we present the proposed framework for non-clairvoyant scheduling, illustrated in Figure 2. We first formulate the sequential decision process (§ 4.1), then describe the job encoder (§ 4.2) and the attention-based decoder (§ 4.3), and finally present the training scheme (§ 4.4).

4.1 Decision Process and State Transition

We formulate scheduling as a sequential decision process. At each decision step t, the policy observes the current state and selects the next job to be processed; the environment then advances to the next decision step. The state, action, and transition are defined below. Note that both the single-machine and parallel-machine settings share the same job selection framework as shown in Figure 2; however, for parallel-machine setting, there will be one additional step to locate the selected job into the machine with smallest predicted load.

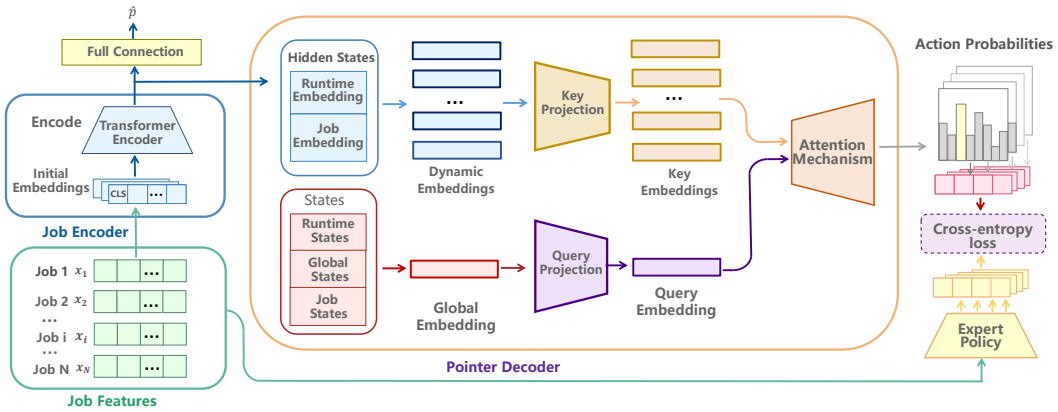


Figure 2: The proposed policy network π_θ maps the current state to a distribution over the pending jobs. The job encoder embeds each pending job from its submission-time features $\mathbf{x}_j$; the embeddings are augmented with runtime features, and the attention-based decoder scores the pending jobs to select the next action.

State. The state s_t consists of the pending jobs $\mathcal{P}_t$ together with their available information, which differs between the two settings. In the preemptive single-machine setting, for each pending job $j \in \mathcal{P}_t$ the state contains its job feature vector $\mathbf{x}_j$, its release time r_j, and the amount of service $v_j(t)$ it has received so far. In the parallel-machine setting, the state contains each pending job's feature vector $\mathbf{x}_j$ and the current predicted load of each machine, defined as the sum of the predicted processing times $\hat{p}_j$ of the jobs already assigned to it. The true processing times p_j^* are never part of the state.

Action. Any action a_t selects a job j_t from the pending set $\mathcal{P}_t$. On a single machine, j_t is the next job to be processed; on parallel machines, j_t is the next job to be assigned, to the machine with the smallest predicted load. The action space is therefore the set of pending jobs, whose size $|\mathcal{P}_t|$ varies over time. The policy assigns each pending job j a selection probability $\pi_\theta(j \mid s_t)$, and the action a_t is the highest-probability job.

Transition. In the single-machine setting, each decision step is triggered by a job arrival or completion: the selected job j_t runs until the next event, which is either its own completion or the arrival of a new job, whichever occurs first; accordingly its attained service grows, a completed job leaves the pending set, and a preempted job enters. In the parallel-machine setting, all jobs are released at time zero and are non-preemptive: each decision places the selected job j_t on the machine with the smallest predicted load; afterwards, that machine's load is increased by the job's predicted processing time $\hat{p}_{j_t}$. The detailed scheduling algorithms for single-machine and parallel-machine settings are shown in Appendix C.

4.2 Job Encoder

The job encoder maps each job j to an embedding $e_j \in \mathbb{R}^d$ from its job feature vector $\mathbf{x}_j$, obtained through feature engineering (Appendix D.2) and split into k^{num} numeric features $\mathbf{x}_j^{\text{num}}$ and k^{cat} categorical features $\mathbf{x}_j^{\text{cat}}$. A learnable embedding layer (Gorishniy et al., 2021) projects each feature into a d-dimensional space: each numeric feature $x_{j,i}^{\text{num}}$ is projected by a linear layer, yielding $z_{j,i}^{\text{num}} = W_i^{\text{num}} x_{j,i}^{\text{num}} + b_i^{\text{num}}$ with $W_i^{\text{num}}, b_i^{\text{num}} \in \mathbb{R}^d$, while each categorical feature $x_{j,c}^{\text{cat}}$ is mapped by a lookup $z_{j,c}^{\text{cat}} = E_c^{\text{cat}}[\, x_{j,c}^{\text{cat}} \,]$ with $E_c^{\text{cat}} \in \mathbb{R}^{V_c \times d}$, where V_c is the cardinality of feature c. A learnable [CLS] token z^{cls} is prepended to these embeddings, and the resulting sequence is passed through a stack of L transformer layers (Vaswani et al., 2017), each combining multi-head attention with a feed-forward network under residual connections and layer normalization; the [CLS] token of the final layer yields e_j, as illustrated in Figure 3. Besides the embedding, the encoder's output head also produces a predicted processing time $\hat{p}_j$, which is used to form the predicted machine loads in the parallel-machine setting.

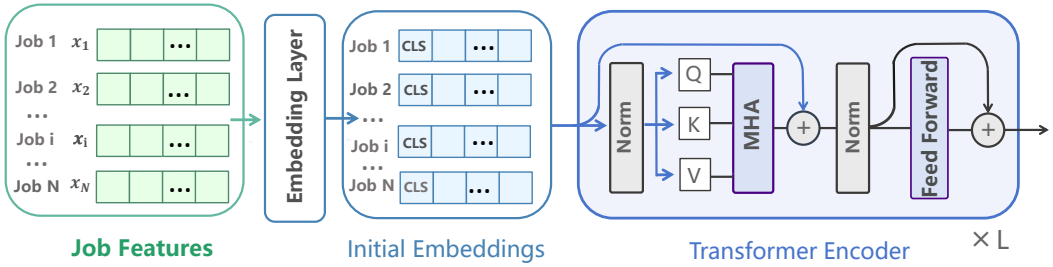


Figure 3: The job encoder maps each pending job to an embedding e_j. The submission-time features are tokenized and prepended with a learned [CLS] token, and the resulting sequence passes through L transformer layers to yield e_j.

4.3 Attention-based Decoder

The decoder selects the next job to process from the pending set $\mathcal{P}_t$ by a pointer mechanism (Vinyals et al., 2015; Kool et al., 2019). Following (Mao et al., 2019), the decoder augments each job embedding e_j from the encoder with the job's runtime features, forming a runtime-augmented embedding

$$u_j = [\, e_j \,;\, \tilde{r}_j \,], \tag{1}$$

where $\tilde{r}_j = f_r(rt_j)$ is an embedding of the job's runtime features rt_j, comprising its queuing time, the service it has attained so far, and an indicator of whether the job is currently running. The pending set is unordered, for which a permutation-invariant aggregation is the standard choice (Xie & Tong, 2025; Zaheer et al., 2017). The decoder pools the augmented embeddings by mean and maximum, and concatenates the result with an embedding of the global information to form the context,

$$c = \mathrm{Concat}\big(\bar{u},\, u_{\max},\, \tilde{g}\big), \tag{2}$$

where $\bar{u}$ and $u_{\max}$ are the element-wise mean and maximum over u_j, applied jointly to the job embeddings and the runtime features. The vector $\tilde{g} = f_g(g)$ is an embedding of the global information g, comprising the current time and the number of pending jobs. The score of job j is then obtained by an attention mechanism in which the context c serves as the query and u_j as the key, both projected by learnable matrices W_Q and W_K,

$$\sigma_j = \gamma \cdot \tanh\left(\frac{(W_Q\, c)^\top\, (W_K\, u_j)}{\sqrt{d_k}} \right), \tag{3}$$

where d_k is the projection dimension, σ_j denotes the score of job j, and γ is a scale parameter, set to 10, following (Bello et al., 2017). The scores are normalized by a softmax into the policy distribution,

$$\pi_\theta(j_t \mid s_t) = \frac{\exp(\sigma_{j_t})}{\sum_{k \in \mathcal{P}_t} \exp(\sigma_k)}, \qquad j_t \in \mathcal{P}_t, \tag{4}$$

where j_t is the selected job, and σ_{j_t} is its score. During training, this distribution is supervised on the expert's action; during inference, the decoder selects the highest-scoring job.

4.4 Training Scheme

We train the policy end-to-end by behavioral cloning of an expert policy. For each objective, the expert policy observes the true processing times with present state s_t, computes the optimal action a_t^* at every decision step, and running it on the training instances produces a dataset $\mathcal{D}$ of expert state-action pairs (s_t, a_t^*). We then minimize the cross-entropy loss over $\mathcal{D}$,

$$J_{\mathrm{BC}} = -\frac{1}{|\mathcal{D}|} \sum_{(s_t, a_t^*) \in \mathcal{D}} \log \pi_\theta(a_t^* \mid s_t), \tag{5}$$

Alongside the cloning objective, the encoder's output head is also trained to predict the processing time. Its regression loss is $J_{\mathrm{pred}} = \frac{1}{n} \sum_{j=1}^{n} \big(\log(\hat{p}_j + 1) - \log(p_j^* + 1)\big)^2$, where n is the number

of training jobs, p_j^* is the true processing time of job j, and the logarithm equalizes the weight of relative errors across the range of processing times. The two objectives are optimized jointly,

$$J = J_{\mathrm{BC}} + \lambda\, J_{\mathrm{pred}}, \tag{6}$$

where λ balances the two terms. In the single-machine setting, the decision of selecting the next process job can be directly achieved by π_θ, and the loss of the prediction of the processing time $\hat{p}_j$ is an auxiliary signal. In the parallel-machine setting, after obtaining the selected job, predicted processing time $\hat{p}_j$ estimates the machine loads that determine the assigned machine for that selected job.

5 Experiments

This section evaluates the learned policy on the three scheduling objectives, i.e., the total completion time, makespan, and max-stretch. It comprises the experimental setup and baselines, the main results on the two workloads, the ablation studies, and a visualization of the learned policy.

5.1 Experimental Setup

Dataset. We conduct experiments on two production workloads, ATLAS and ASTRA. ATLAS (Jiang et al., 2026) is a benchmark derived from the publicly available Alibaba PAI-2020 GPU-cluster trace (Weng et al., 2022). ASTRA is a workload we construct from Alibaba's production GPU cluster trace (Duan et al., 2026), extracting submission-time features for each of its $463{,}279$ jobs with the recorded execution duration as ground truth. Most of ASTRA's jobs are large-model tasks such as LLM training and online inference (Duan et al., 2026), and they run for a long time. ASTRA is far heavier-tailed than ATLAS: the two have comparable medians—$1{,}192$ s against 663 s (Jiang et al., 2026)—yet the mean job duration is $146{,}701$ s against $5{,}382$ s, and the 99th percentile is 52 days against 19.1 hours, a gap of $65\times$. Such large-model jobs demand strict service-level objectives and cannot be preempted without discarding their unsaved progress (Duan et al., 2026). Therefore, once the large-model jobs are placed, they run to completion—the non-preemptive regime our parallel-machine setting assumes. We therefore evaluate ASTRA in the parallel-machine setting only. For both workloads, we order jobs by submission time and partition them into time-disjoint training, validation, and test sets with proportions 70%, 15%, and 15%, so that no test job is submitted before any training job. A detailed description of the two workloads is deferred to Appendix D.1.

Implementation Details. The job encoder embeds the input features and passes the resulting sequence through a three-layer transformer. It is optimized jointly with the attention-based decoder using Adam. Each training instance spans $7{,}000$ jobs, and each iteration optimizes a batch of 128 states drawn from it, at an initial learning rate of 2×10^{-3}. The model is trained on two NVIDIA RTX PRO 6000 GPUs. Our code is available at `https://anonymous.4open.science/r/non-clairvoyant-with-behavior-5597`.

Classical Baselines. For non-clairvoyant baselines, we use first-in-first-out (FIFO), random assignment, round-robin (RR), which shares the machine equally among active jobs (Motwani et al., 1994), and least-attained-service (LAS), which prioritizes the job that has received the least service so far (Nuyens & Wierman, 2008). For clairvoyant baselines, which observe the true processing times and serve as performance theoretical bounds, we use methods such as shortest remaining processing time (SRPT), earliest-deadline-first (EDF) (Harchol-Balter, 2013), shortest-job-first (SJF), longest-processing-time (LPT), and shortest-processing-time (SPT). For maximum stretch, the optimal policy is EDF with deadlines $r_j + S^* p_j^*$, where S^* is the optimal stretch and is found by bisection on the stretch bound with EDF-feasibility. For total completion time, SRPT is optimal (Schrage, 1968). For non-preemptive makespan, no classical policy is optimal, the problem being NP-hard; LPT is near-optimal on these instances and serves as the clairvoyant reference. To stay consistent with the ATLAS baselines (Jiang et al., 2026), we adopt the same classical baselines.

Learning-augmented Baselines. These baselines feed predicted job sizes $\hat{p}_j$ into the classical clairvoyant online algorithms, using the predictors of ATLAS (Jiang et al., 2026). To stay consistent with existing methods, we adopt the same learning-augmented baselines. For total completion

time, we evaluate shortest-predicted-job-first (SPJF) and preferential round-robin (PRR), which reserves a processor share for the job with the smallest prediction and distributes the remaining rate equally (Purohit et al., 2018). For makespan, we substitute the true sizes with predictions to obtain longest-predicted-processing-time (LPPT) and shortest-predicted-processing-time (SPPT). For maximum stretch, we evaluate SPRPT, which runs SRPT on predicted remaining work, and EDF-P, an earliest-deadline-first policy that schedules by predicted deadlines. It is worth noting that ATLAS (Jiang et al., 2026) assumes that every job has a known upper bound on its processing time, so the predictions of jobs whose sizes are overestimated are limited to this bound. However, we believe that in practice the processing time bound could be challenging to obtain, and may even be unavailable in many scenarios. To make the method more practical, we do not assume any time bound.

Evaluation Metrics. Each policy is scored by its empirical competitive ratio, defined as the cost it attains relative to an optimal solution or a tight bound. For maximum stretch $1|r_j, \mathrm{pmtn}| \max_j S_j$, we obtain the optimal stretch by bisection with EDF feasibility (Harchol-Balter, 2013) and normalize by the realized stretch $S_{\mathrm{emp}} = \max_j (C_j - r_j)/p_j$, reporting $\rho_{S,\max} = \max_j S_j / S_{\mathrm{emp}}$ together with the 99th percentile $\rho_{S,99}$. For total completion time $1|r_j, \mathrm{pmtn}| \sum_j C_j$, we normalize by SRPT: $\rho_{\mathrm{TC}} = \sum_j C_j / \sum_j C_j^{\mathrm{SRPT}}$. For makespan $(P||C_{\max})$, we normalize by the preemptive bound $\mathrm{OPT}_{\mathrm{pre}} = \max\{\sum_j p_j^*/m,\ \max_j p_j^*\}$ (McNaughton, 1959): $\rho_{\mathrm{MS}} = C_{\max}/\mathrm{OPT}_{\mathrm{pre}}$.

5.2 Main Results

We evaluate the maximum stretch and total completion time (single-machine metrics) on ATLAS, and evaluate the makespan (parallel-machine metric) on both datasets, reporting the results in Table 1.

ATLAS. Table 1 compares our policy against the baselines on ATLAS (Jiang et al., 2026); our policy clones EDF, SRPT, and LPT for maximum stretch, total completion time, and makespan, respectively. For total completion time, the learned policy achieves $\rho_{\mathrm{TC}} = 1.094$, a 9.4% degradation from the clairvoyant optimum, against 10.6% for the strongest baseline Meta-SPJF (1.106), indicating that directly learning the decision supersedes the ranking based on predicted job processing times, even when the predicted times are rather accurate. For makespan, the learned policy attains $\rho_{\mathrm{MS}} = 1.434$, against 57.4% for the strongest baseline Iso-LPPT (1.574), as this objective depends primarily on identifying the largest jobs. For maximum stretch, the learned policy attains $\rho_{S,\max} = 42.333$, a 95.8% lower ratio than the best learning-augmented baseline HRAS-SPRPT (1013.22) and a $4.8\times$ improvement over the non-learning LAS/FB (205.28). Also, its advantage holds across the distribution rather than only in the tail, where the 99th percentile falls 56.4%, from 12.57 to 5.485. Since maximum stretch is governed by the worst single job, the prediction-based baselines are vulnerable to a single severely overestimated prediction once this bound is absent; our policy does not schedule by predicted deadlines and is considerably less affected.

ASTRA. We report the same baselines as in ATLAS, with the predictors retrained on the ASTRA training split. On this dataset, we evaluate makespan only, and the learned policy clones SPT. In Table 1, it attains $\rho_{\mathrm{MS}} = 1.670$ (67.0% above the preemptive optimum), against 87.4% for the strongest baseline TwoSt-SPPT (1.874). Also, the shortest-predicted rule dominates, with every SPPT variant outperforming its LPPT counterpart.

5.3 Ablation Studies

Choice of expert. Table 2 compares different imitated expert policies on both datasets. In the single-machine setting, we evaluate maximum stretch and total completion time on ATLAS. For maximum stretch, cloning EDF attains $\rho_{S,\max} = 42.333$ and $\rho_{S,99} = 5.485$, against 50.539 and 7.146 for SRPT. For total completion time, cloning SRPT attains $\rho_{\mathrm{TC}} = 1.094$, against 1.141 for EDF. In the parallel-machine setting, we evaluate makespan on both ATLAS and ASTRA. Cloning LPT is better on ATLAS (1.434 against 1.585 for SPT), whereas on ASTRA the order reverses: SPT attains 1.670 against 1.715 for LPT. For non-clairvoyant scheduling, behavioral cloning leaves an inherent gap, since it imitates a clairvoyant expert which has access to the true processing time. The theoretical optimum thus does not imply the optimal imitation.

Table 1: Scheduling performance on ATLAS across three objectives, together with makespan on ASTRA; lower is better. Max-stretch on $n = 5{,}000$ jobs; total completion time on $n = 10{,}000$ jobs; makespan ratios averaged over $m \in \{5, 10, 20, 50, 100\}$ and reported as ATLAS / ASTRA. Best learned result per column in bold.

(A) Max-Stretch			**(B) Total compl. time**		**(C) Makespan**	
Algorithm	$\rho_{S,\max}$	$\rho_{S,99}$	**Algorithm**	ρ_{TC}	**Algorithm**	ρ_{MS}
OPT (EDF at S^*)	**1.000**	**1.000**	**SRPT**	**1.000**	**LPT**	**1.000 / 1.000**
SRPT	1.581	1.304	SJF	1.001	SPT	1.539 / 1.324
LAS/FB	205.28	134.18	RR	1.975	Random	1.898 / 2.278
			FIFO	5.372		
CQR-SPRPT	2302.15	17.26	CQR-SPJF	1.135	CQR-LPPT	1.721 / 2.160
CQR-EDF	2528.50	232.60	CQR-PRR	1.328	CQR-SPPT	1.701 / 2.098
HRAS-SPRPT	1013.22	273.34	HRAS-SPJF	1.686	HRAS-LPPT	1.937 / 2.234
HRAS-EDF	3202.49	979.31	HRAS-PRR	1.887	HRAS-SPPT	1.753 / 2.097
Iso-SPRPT	2328.36	99.18	Iso-SPJF	1.161	Iso-LPPT	1.574 / 2.513
Iso-EDF	2623.61	281.84	Iso-PRR	1.354	Iso-SPPT	1.717 / 2.175
Meta-SPRPT	2690.41	12.57	Meta-SPJF	1.106	Meta-LPPT	1.689 / 2.171
Meta-EDF	2755.16	109.07	Meta-PRR	1.292	Meta-SPPT	1.731 / 2.159
TwoSt-SPRPT	2046.05	19.58	TwoSt-SPJF	1.124	TwoSt-LPPT	1.643 / 2.142
TwoSt-EDF	2243.80	263.61	TwoSt-PRR	1.310	TwoSt-SPPT	1.722 / 1.874
Rec-SPRPT	2866.71	18.83	Rec-SPJF	1.172	Rec-LPPT	1.842 / 2.318
Rec-EDF	3053.57	288.62	Rec-PRR	1.369	Rec-SPPT	1.943 / 1.890
Ours	**42.333**	**5.485**	**Ours**	**1.094**	**Ours**	**1.434 / 1.670**

Table 2: Ablation of different imitated expert policies on the ATLAS and ASTRA workloads, with makespan as ATLAS / ASTRA

(A) Max-Stretch			**(B) Total compl. time**		**(C) Makespan**	
Expert	$\rho_{S,\max}$	$\rho_{S,99}$	**Expert**	ρ_{TC}	**Expert**	ρ_{MS}
EDF	**42.333**	**5.485**	EDF	1.141	LPT	**1.434** / 1.715
SRPT	50.539	7.146	SRPT	**1.094**	SPT	1.585 / **1.670**

Attention-based decoder. Table 3 reports the results of removing the learned decoder or the attention mechanism. **w/o. decoder** removes the decoder and follows the learning-augmented rules: SPRPT on the single machine, and LPPT on ATLAS and SPPT on ASTRA for makespan. **w/o. attn.** replaces only the attention mechanism with a shared MLP that scores each runtime-augmented embedding together with the mean and maximum over all pending embeddings. Removing the learned decoder degrades every objective: $\rho_{S,\max}$ from 42.333 to 1460.6, $\rho_{S,99}$ from 5.485 to 1267.885, ρ_{TC} from 1.094 to 1.93, and ρ_{MS} from $1.434/1.670$ to $1.8293/2.261$. This shows that learning to schedule end-to-end outperforms routing the same predictions through a classical algorithm. Removing the attention mechanism is milder—$\rho_{S,\max} = 101.838$, $\rho_{S,99} = 6.185$, $\rho_{\mathrm{TC}} = 1.321$, and $\rho_{\mathrm{MS}} = 1.547/1.860$—but still worse on every column. Because a job's priority depends on the rest of the pending set, the query-key attention projects the global context into a query and scores each

job by matching this query against the job's key, thereby better capturing each job's priority relative to the whole set.

Table 3: Ablation of the attention-based decoder on the ATLAS and ASTRA workloads, with makespan as ATLAS / ASTRA.

(A) Max-Stretch			**(B) Total compl. time**		**(C) Makespan**	
Algorithm	$\rho_{S,\max}$	$\rho_{S,99}$	**Algorithm**	ρ_{TC}	**Algorithm**	ρ_{MS}
w/o. decoder	1460.6	1267.885	w/o. decoder	1.93	w/o. decoder	1.8293 / 2.261
w/o. attn.	101.838	6.185	w/o. attn.	1.321	w/o. attn.	1.547 / 1.860
Ours	**42.333**	**5.485**	**Ours**	**1.094**	**Ours**	**1.434 / 1.670**

5.4 VISUALIZATION

As shown in Figure 4, we visualize a parallel-machine case from ASTRA, where our policy clones SPT as its expert and the two strongest learning-augmented baselines on this workload instead follow SPPT. Every method, including our model, mispredicts the largest job J_3 as the smallest, so SPPT selects J_3 first, whereas our policy selects J_1 first. The misprediction then compounds: once the other machines are occupied, M_0 becomes the smallest in predicted load and is selected again. SPPT's fixed order then assigns a large job there, aggravating the bottleneck. The learned policy does not select jobs by predicted size and places the smaller J_1 there instead, alleviating the compounding and lowering the makespan.

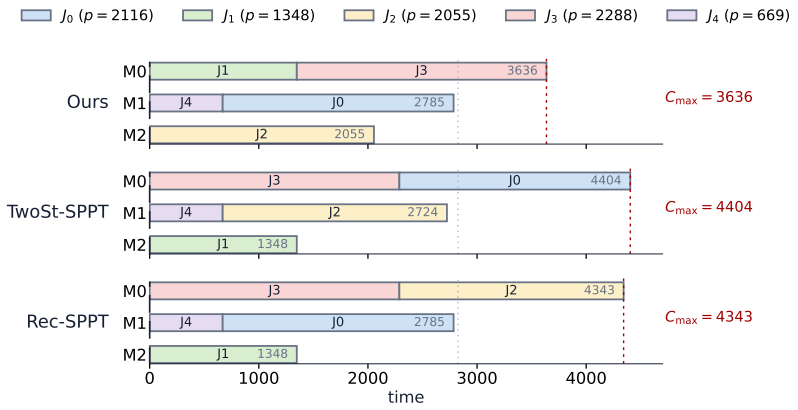


Figure 4: Gantt chart of the learned policy and two learning-augmented baselines on a parallel-machine case from ASTRA. The three rows M_0, M_1 and M_2 are the machines to which the five jobs, of sizes 2116, 1348, 2055, 2288 and 669, are assigned.

6 CONCLUSION

This paper proposes a behavioral-cloning framework for non-clairvoyant scheduling, in which a policy is trained directly from an expert policy rather than by routing predicted job sizes through a classical online algorithm. The policy is supervised by the scheduling decision at each state, with the predicted job size as an auxiliary objective, so that the ordering of jobs is no longer governed by a single predicted scalar. An attention-based pointer mechanism assigns scores to an arbitrary number of pending jobs, handling the variable action space that could reach thousands of jobs in production workloads. We further construct ASTRA, a workload from an open-source production GPU-cluster trace whose size distribution is far heavier-tailed than ATLAS. In the single-machine setting, the proposed method attains the best ratios for maximum stretch and total completion time on ATLAS. The advantage extends to the parallel-machine setting, where it also holds for makespan on both ATLAS and ASTRA.

AI USE STATEMENT

We used GPT-5.6 for polishing the writing and improving the readability of the manuscript, for searching and summarizing related work during the literature review, and for implementing and debugging parts of the experimental code. GPT-5.6 was not involved in research ideation, the design of the scheduling methodology, experiment design, data analysis, or the interpretation of results. All experiments and analyses were performed and verified. We take full responsibility for all content of this work.

REFERENCES

Evripidis Bampis, Alexander Kononov, Giorgio Lucarelli, and Fanny Pascual. Non-clairvoyant makespan minimization scheduling with predictions. In *International Symposium on Algorithms and Computation (ISAAC)*, pp. 9:1–9:15, 2023.

Nikhil Bansal, Kedar Dhamdhere, Jochen Könemann, and Amitabh Sinha. Non-clairvoyant scheduling for minimizing mean slowdown. In *Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 260–270, 2003.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. In *International Conference on Learning Representations (ICLR), Workshop Track*, 2017.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2): 405–421, 2021.

Ziyad Benomar and Vianney Perchet. Non-clairvoyant scheduling with partial predictions. In *International Conference on Machine Learning (ICML)*, pp. 3506–3538, 2024.

Inguk Choi, Woo-Jin Shin, Sang-Hyun Cho, and Hyun-Jung Kim. Towards generalizable multi-policy optimization with self-evolution for job scheduling. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 24569–24616, 2025.

Andrea Corsini, Angelo Porrello, Simone Calderara, and Mauro Dell'Amico. Self-labeling the job shop scheduling problem. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 105528–105551, 2024.

Jiaang Duan, Shenglin Xu, Shiyou Qian, Dingyu Yang, Kangjin Wang, Chenzhi Liao, Yinghao Yu, Qin Hua, Hanwen Hu, Qi Wang, Wenchao Wu, Dongqing Bao, Tianyu Lu, Jian Cao, Guangtao Xue, Guodong Yang, Liping Zhang, and Gang Chen. GFS: A preemption-aware scheduling framework for GPU clusters with predictive spot instance management. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pp. 117–131. ACM, 2026.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 18932–18943, 2021.

Juncheng Gu, Mosharaf Chowdhury, Kang G. Shin, Yibo Zhu, Myeongjae Jeon, Junjie Qian, Hongqiang Liu, and Chuanxiong Guo. Tiresias: A GPU cluster manager for distributed deep learning. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pp. 485–500, 2019.

Mor Harchol-Balter. *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge University Press, 2013.

Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 285–294, 2021.

Wonseok Jeon, Mukul Gagrani, Burak Bartan, Weiliang Will Zeng, Harris Teague, Piero Zappi, and Christopher Lott. Neural DAG scheduling via one-shot priority sampling. In *International Conference on Learning Representations (ICLR)*, 2023.

Wenhua Jiang and Cun-Hui Zhang. General maximum likelihood empirical Bayes estimation of normal means. *The Annals of Statistics*, 37(4):1647–1684, 2009.

Zhiyun Jiang, Tianming Zhao, Chunqiu xia, and Albert Zomaya. Atlas: Alibaba dataset and benchmark for learning-augmented scheduling. In *International Conference on Learning Representations (ICLR)*, pp. 137404–137421, 2026.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations (ICLR)*, 2019.

Je-Hun Lee and Hyun-Jung Kim. Graph-based imitation learning for real-time job shop dispatcher. *IEEE Transactions on Automation Science and Engineering*, 22:8593–8606, 2025.

Longkang Li, Siyuan Liang, Zihao Zhu, Chris Ding, Hongyuan Zha, and Baoyuan Wu. Learning to optimize permutation flow shop scheduling via graph-based imitation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, pp. 20185–20193, 2024.

Alexander Lindermayr and Nicole Megow. Permutation predictions for non-clairvoyant scheduling. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 357–368, 2022.

Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication (SIGCOMM)*, pp. 270–288, 2019.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

Robert McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1): 1–12, 1959.

Michael Mitzenmacher. Scheduling with predictions and the price of misprediction. In *11th Innovations in Theoretical Computer Science Conference (ITCS)*, pp. 14:1–14:18, 2020.

Rajeev Motwani, Steven Phillips, and Eric Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130(1):17–47, 1994.

Misja Nuyens and Adam Wierman. The foreground-background queue: A survey. *Performance Evaluation*, 65(3-4):286–307, 2008.

Michael L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, third edition, 2008.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 9684–9693, 2018.

Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC)*, pp. 7:1–7:13, 2012.

Linus Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

Muhammad Tirmazi, Adam Barker, Nan Deng, Md E. Haque, Zhijing Gene Qin, Steven Hand, Mor Harchol-Balter, and John Wilkes. Borg: The next generation. In *Proceedings of the Fifteenth European Conference on Computer Systems (EuroSys)*, pp. 30:1–30:14, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

Alexander Wei and Fred Zhang. Optimal robustness-consistency trade-offs for learning-augmented online algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 8042–8053, 2020.

Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu 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)*, pp. 945–960. USENIX Association, 2022.

Jiahao Xie and Guangmo Tong. Advances in set function learning: A survey of techniques and applications. *ACM Computing Surveys*, 57(7):1–37, 2025.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander Smola. Deep sets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Xu Chi. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1621–1632, 2020.

Tianming Zhao and Albert Zomaya. On smoothness bounds for non-clairvoyant scheduling with predictions. In *International Conference on Learning Representations (ICLR)*, pp. 140116–140147, 2026.

Tianming Zhao, Wei Li, and Albert Y. Zomaya. Uniform machine scheduling with predictions. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, pp. 413–422, 2022.

Tianming Zhao, Chunqiu xia, Xiaomin Chang, Chunhao Li, Wei Li, and Albert Zomaya. Competitive fair scheduling with predictions. In *International Conference on Learning Representations (ICLR)*, pp. 66548–66585, 2025.

A NOTATIONS

A.1 BASIC ENTITIES AND TIME VARIABLES

Symbol	Name	Meaning
J_j	Job	An independent job to be scheduled
n	Number of jobs	Total number of jobs
r_j	Release time	Time at which job J_j arrives
p_j^*	True processing time	Ground-truth size of job J_j
$\mathbf{x}_j$	Job features	Submission-time features of job J_j
C_j	Completion time	Time at which job J_j finishes
S_j	Stretch	Response time relative to size, $S_j = \frac{C_j - r_j}{p_j^*}$
m	Machine count	Number of identical parallel machines
$C_{\max}$	Makespan	Maximum completion time, $\max_j C_j$

A.2 JOB ATTRIBUTES

Symbol	Name	Meaning
n_j	Instance count	Number of instances requested by job J_j
q_j	Resource request	Per-instance CPU/GPU request of job J_j
R_j	Total request	Job-level request, $R_j = n_j \cdot q_j$
$\hat{p}_j$	Predicted processing time	Predicted size of job J_j

A.3 DECISION PROCESS

Symbol	Name	Meaning
$\mathcal{P}_t$	Pending set	Jobs available for scheduling at decision step t
$v_j(t)$	Attained service	Service received by job J_j up to time t
$\mathbf{L}$	Predicted machine loads	Predicted load of each machine, the sum of $\hat{p}_j$ over the jobs assigned to it
s_t	State	Pending jobs and their observed information at step t
a_t	Action	Action taken at decision step t, selecting a job from $\mathcal{P}_t$
j_t	Selected job	Job chosen by action a_t
π_θ	Scheduling policy	Learned policy mapping state to decision

A.4 LEARNING

Symbol	Name	Meaning
e_j	Job embedding	Encoded representation of job J_j
$\mathcal{D}$	Expert dataset	State-action pairs from the expert policy
a_t^*	Expert action	Optimal action of the expert policy at step t
θ	Policy parameters	Trainable parameters of π_θ

B ILLUSTRATIVE EXAMPLE

We illustrate the two settings of Section 3 with small examples, to make concrete what each objective optimizes and what decision a learned policy must imitate.

Single machine. On one machine with preemption, Figure 5 traces the schedules of two clairvoyant policies, SRPT and SJF, together with that of the non-clairvoyant FIFO, from which $\sum_j C_j$ and $\max_j S_j$ are computed. SRPT is the only policy that preempts: when J_3 arrives at $t = 6$, the remaining work of J_1 exceeds J_3's size, so SRPT interrupts J_1 and finishes the shorter job first, whereas SJF and FIFO, being non-preemptive once started, allow J_1 to run to completion. SRPT attains the smallest total completion time and the smallest maximum stretch, while FIFO delays the

shortest job J_3 the most and is penalized disproportionately, since its stretch divides the same delay by the smallest size.

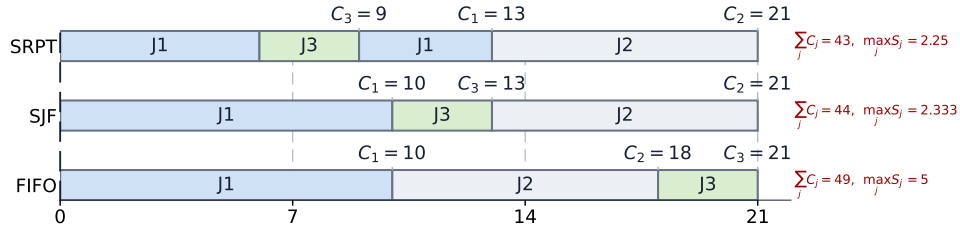


Figure 5: A toy example comparing clairvoyant and non-clairvoyant scheduling on a single machine with preemption, with three jobs of sizes 10, 8, and 3 arriving at 0, 3, and 6. SRPT interrupts the running job when a shorter one arrives and attains a total completion time of 43, remaining ahead of SJF's 44 and FIFO's 49; the same ordering holds for the maximum stretch, where SRPT achieves 2.25 against SJF's 2.33 and FIFO's 5.

Parallel machines. On three identical machines without preemption, Figure 6 compares two assignments of the same five jobs, with sizes $(J_1, \ldots, J_5) = (13, 8, 9, 6, 6)$. Because the jobs are released simultaneously and run to completion, the makespan $C_{\max}$ equals the maximum machine load, so the assignment alone determines the objective. The balanced assignment distributes the jobs so that the three machines finish at nearly the same time, whereas the imbalanced one groups J_1 (13) and J_2 (8) onto a single machine, whose load 21 becomes the bottleneck while a third machine holds only J_4 (6).

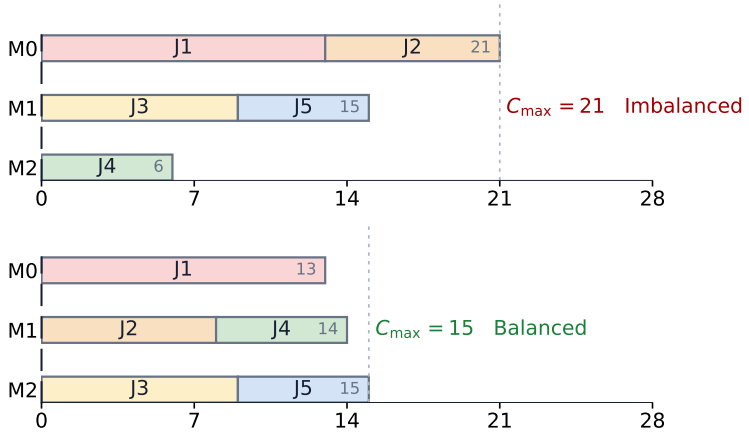


Figure 6: A toy example comparing two assignments of the same five jobs to three identical machines without preemption, with job sizes 13, 8, 9, 6, and 6. Since the makespan equals the maximum machine load, a balanced assignment yields a makespan of 15, whereas an imbalanced one leaves the largest job on a single machine and yields 21.

C Decision Process

We formalize the decision process of Section 4.1 as two self-contained algorithms, one per setting, as outlined in Algorithm 1 and Algorithm 2. The two algorithms share a common structure: at each decision step the policy observes the state of the pending jobs and selects a job to process. They differ in the transition. In the single-machine setting the chosen job runs directly. In the parallel-machine setting it must additionally be assigned to a machine according to the predicted loads.

Algorithm 1 realizes the preemptive single-machine loop. At each decision step the policy selects the next job to process, and the environment then advances to the next arrival or completion event.

Two boundary cases are resolved as follows: if no further job is released, the selected job runs to completion, whereas if a job completes at exactly the instant of the next release $\min_{j:r_j>t} r_j$, the completion takes precedence. Accordingly, a selected job that completes no later than the next release is moved to the completion set, while one that is preempted by an arrival has its attained service increased and remains pending.

Algorithm 1 Single-machine decision process.

Input: jobs $\{(\mathbf{x}_j, r_j)\}_{j=1}^n$, policy π_θ, true sizes $\{p_j^*\}$ (environment only)
Output: completion times $\{C_j\}$ and stretches $\{S_j\}$
Initialize clock $t \leftarrow \min_j r_j$, pending set $\mathcal{P} \leftarrow \{j : r_j = \min_j r_j\}$, completion set $\mathcal{C} \leftarrow \emptyset$
while $|\mathcal{C}| < n$ **do**
  if $\mathcal{P} = \emptyset$ **then**
    Advance to the next release $t \leftarrow \min_{j:r_j>t} r_j$, $\mathcal{P} \leftarrow \{j : r_j = t\}$
  end if
  Form state $s_t \leftarrow \{(\mathbf{x}_j, r_j, v_j(t))\}_{j\in\mathcal{P}}$
  Sample action $a_t \sim \pi_\theta(\cdot \mid s_t)$ over $\mathcal{P}$, selecting job j_t
  if $t + p_{j_t}^* - v_{j_t}(t) \;\le\; \min_{j:r_j>t} r_j$ **then**
    Advance to the completion of j_t, $t \leftarrow t + p_{j_t}^* - v_{j_t}(t)$
    Set $C_{j_t} \leftarrow t$, $S_{j_t} \leftarrow (t - r_{j_t})/p_{j_t}^*$, move j_t to $\mathcal{C}$, $\mathcal{P} \leftarrow \mathcal{P} \setminus \{j_t\}$
    Add jobs released at t: $\mathcal{P} \leftarrow \mathcal{P} \cup \{j : r_j = t\}$
  else
    Advance to the next release $t' \leftarrow \min_{j:r_j>t} r_j$
    Update $v_{j_t} \leftarrow v_{j_t} + t' - t$, $t \leftarrow t'$
    Add released jobs $\mathcal{P} \leftarrow \mathcal{P} \cup \{j : r_j = t'\}$
  end if
end while

Algorithm 2 realizes the non-preemptive parallel-machine loop, in which all jobs are released at time zero and run to completion once assigned. At each decision step the policy selects a job from the pending set, and the environment places it on the machine with the smallest predicted load, increasing that load by the job's predicted processing time $\hat{p}_{j_t}$. The loop terminates once every job has been assigned, and the makespan equals the maximum machine load, computed from the true processing times.

Algorithm 2 Parallel-machine decision process.

Input: jobs $\{\mathbf{x}_j\}_{j=1}^n$, predictions $\{\hat{p}_j\}$, policy π_θ, true sizes $\{p_j^*\}$ (environment only)
Output: makespan $C_{\max} = \max_i \sum_{j\in\mathcal{M}_i} p_j^*$
Initialize $\mathcal{P} \leftarrow \{1, \ldots, n\}$, predicted loads $\mathbf{L} \leftarrow \mathbf{0}$, assignment sets $\mathcal{M}_i \leftarrow \emptyset$ for each machine $i = 1, \ldots, m$
while $\mathcal{P} \neq \emptyset$ **do**
  Form state $s_t \leftarrow (\{\mathbf{x}_j\}_{j\in\mathcal{P}}, \mathbf{L})$
  Sample action $a_t \sim \pi_\theta(\cdot \mid s_t)$ over $\mathcal{P}$, selecting job j_t
  Assign j_t to the least-loaded machine $i^* \leftarrow \operatorname{argmin}_i L_i$, $\mathcal{M}_{i^*} \leftarrow \mathcal{M}_{i^*} \cup \{j_t\}$
  Update predicted load $L_{i^*} \leftarrow L_{i^*} + \hat{p}_{j_t}$
  Remove j_t, $\mathcal{P} \leftarrow \mathcal{P} \setminus \{j_t\}$
end while

D DATASET FEATURE EXTRACTION

D.1 DATASET DESCRIPTION AND WORKLOAD CHARACTERIZATION

Data pre-processing. ASTRA is derived from a production GPU cluster trace recorded under a spot-instance mechanism, released with the GFS scheduling framework (Duan et al., 2026). The data is organized as a job-level table recording, for each job, its submission time, submitter, GPU model, per-instance CPU and GPU requests q_j, instance count n_j, and job type, with the job-level

request defined as $R_j = n_j \cdot q_j$. The job duration serves as the ground-truth processing time p_j^*. The released trace records timestamps in seconds, and none of its jobs has a missing field or a non-positive duration. However, 3,588 of the 466,867 jobs carry a submission timestamp of zero, whereas any other timestamp is shared by at most 107 jobs. These jobs also have anomalous recorded durations, with a median of 45 days against 20 minutes for the remaining jobs. We exclude these jobs, and the resulting workload contains 463,279 jobs spanning roughly 184 days, submitted by 84 tenants and requesting six GPU models.

Heavy-tailed distribution. As shown in Figure 7, per-job GPU requests, CPU requests, and processing times span several orders of magnitude. At the job level, GPU requests range from fractional shares to 1,456 cards, and CPU requests from 0 to 33,488 cores. Processing times are the most extreme: the median job runs for 20 minutes, the mean for 1.70 days, and the longest for 184 days. Measured relative to each workload's own median, ASTRA's tail is far sharper than that of ATLAS: the 99th percentile exceeds the median by 3,700× in ASTRA against 106× in ATLAS. This concentration of work in the tail motivates the log transformation of the prediction target used throughout the experiments. The tail is a property of the workload: most of ASTRA's jobs are large-model tasks such as LLM training and online inference, which cannot be preempted and now dominate GPU clusters (Duan et al., 2026). Such a tail concentrates the work in a few extreme jobs (Tirmazi et al., 2020), which is precisely what makes non-clairvoyant scheduling hard (Bansal et al., 2003).

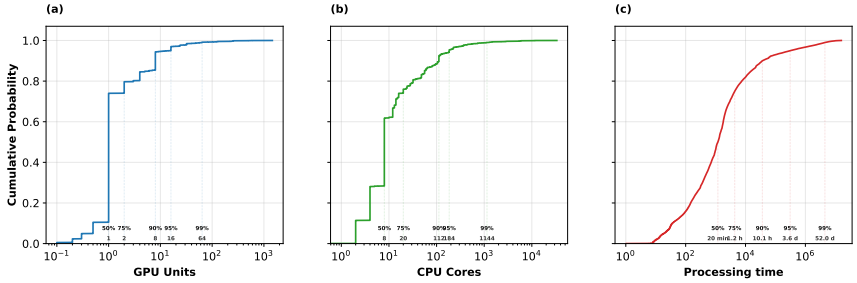


Figure 7: Resource request and processing-time distributions in the workload. Panels (a)–(c) plot the cumulative distribution function (CDF) of per-job requests and processing time on a log-scaled horizontal axis with a common vertical axis of cumulative probability: (a) total requested GPUs, (b) total requested CPU cores, and (c) processing time. Quantiles 50–99% are annotated on each curve.

Submission periodicity. Prior work reports that cluster workloads exhibit strong diurnal and weekly regularity, with submissions concentrating on weekdays and late-night hours while the weekends remain markedly less active (Tirmazi et al., 2020; Reiss et al., 2012). For ASTRA, we group submissions by hour of week and average the counts over the complete weeks of the trace, $\bar{S}(h) = \frac{1}{|\mathcal{W}|} \sum_{w \in \mathcal{W}} S_w(h)$, where $S_w(h)$ is the number of submissions in week w at hour h of the week, and $\mathcal{W}$ is the set of complete weeks. Averaging over the whole trace rather than a single week dampens one-off bursts and makes the resulting profile more stable, as shown in Figure 8. This periodicity motivates the submission-timing features we extract for each job.

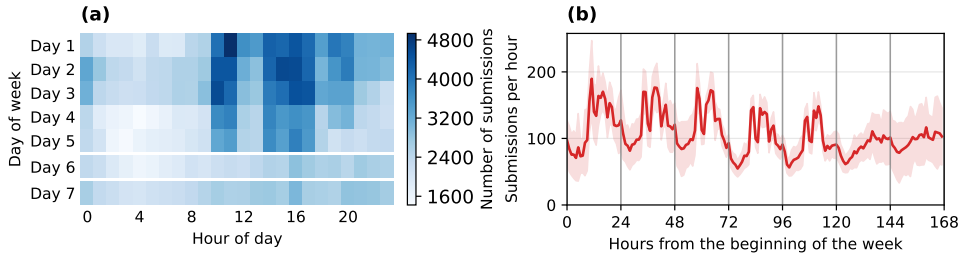


Figure 8: Temporal patterns of submissions. (a) Heatmap of the total submission count by relative day of the week (rows) and hour of the day (columns). (b) Weekly-average submissions per hour of the week; shaded bands show the 95% confidence interval across complete weeks.

The requested resources follow the same periodicity but are even more concentrated than the submissions, as shown in Figure 9, indicating that resource demand is considerably more bursty than the arrival process itself.

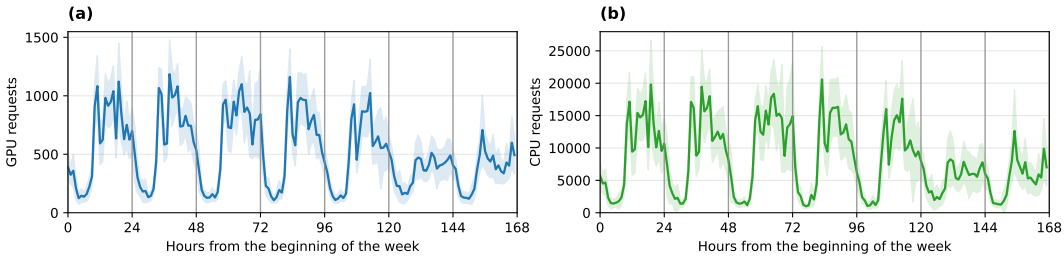


Figure 9: Temporal patterns of the requested resources. Weekly averages of the per-hour (a) GPU requests and (b) CPU requests, computed as per-instance requests multiplied by the number of instances and aggregated per job; shaded bands show 95% confidence intervals across complete weeks.

D.2 Feature Engineering

Each job is described by features available at submission time, and all statistics, bins, and vocabularies are fit on the training split only. (1) Requested resources: the declared CPU and GPU demand, the instance count, the per-instance demand, and a CPU:GPU ratio. (2) Temporal periodicity: sinusoidal encodings of the submission hour and weekday (Kazemi et al., 2019), a weekend indicator, and the logarithm of the number of jobs arriving within the same second. (3) Historical runtime statistics: We group jobs into signatures by submitter, job type, and the decile of their CPU, GPU, and instance requests. For each signature we record how many past jobs it contains and the mean, median, and quartiles of their durations, shrinking the mean toward the global average with an empirical-Bayes estimator (Jiang & Zhang, 2009). For each submitter we also track a running mean and count of earlier durations, an exponentially weighted moving average, and the gap since their previous submission. All these statistics are computed from earlier jobs only. (4) Identity attributes: the submitter, job type, and GPU model, encoded as indices from the training vocabulary, with unseen values sharing one out-of-vocabulary index.