Learning-Augmented Online Scheduling with Parsimonious Preemption

Mugen Blue
UC Santa Cruz
United States
msblue@ucsc.edu

Sungjin Im
UC Santa Cruz
United States
sim9@ucsc.edu

Alexander Lindermayr
TU Berlin
Germany
lindermayr@math.tu-berlin.de

Abstract

Learning-augmented algorithms have emerged as a powerful paradigm to surpass traditional worst-case lower bounds by integrating potentially noisy predictions. While this framework has seen success in online scheduling, existing work primarily optimizes job latency while relying on frequent, "blind" preemptions. This ignores the fundamental trade-off between algorithmic performance and preemption complexity. We provide the first systematic study of learning-augmented scheduling that curbs preemption while optimizing latency. We establish that the gap between theoretical latency bounds and preemption overhead can be bridged with solid analytical foundations. Our results include $O(1)$-competitive algorithms for single and unrelated parallel machines with only $O(1)$ preemptions per job under accurate predictions, with overhead scaling logarithmically with the prediction error. By providing the first bounded-preemption guarantees for unrelated and malleable machines, we extend the theoretical reach of the learning-augmented framework to more constrained and realistic settings. Finally, our algorithms are validated through experiments.

1 Introduction

The paradigm of *learning-augmented algorithms* has recently emerged as a powerful framework for overcoming the limitations of worst-case analysis [41]. By incorporating potentially noisy machine-learning predictions into algorithm design, this framework enables algorithms that are both *consistent*—near-optimal when predictions are accurate—and *robust*—with worst-case guarantees even under adversarial predictions. This approach is especially effective for online algorithms, where predictions can mitigate future uncertainty that is unavoidable in traditional worst-case models.

Online scheduling is a prominent application of this paradigm. Traditionally, algorithms often suffer from limited information about key parameters; a central example is the *non-clairvoyance*, where job sizes are unknown until completion [16, 44]. A growing literature shows that online algorithms can substantially benefit from job-size predictions when minimizing latency [3, 26, 40, 46, 55].

However, in practical systems such as large-scale data centers, *preemption* and *migration* can incur substantial overheads, including context-switching costs and cache invalidation [42, 53, 54]. Although preemption-aware scheduling is well studied in systems research, from mechanisms such as preemption thresholds [47] and deferred-preemption heuristics [13] to production-scale data-center resource management [10, 54], these approaches are largely qualitative and offer few provable guarantees on job latency, particularly under prediction error.

We argue that the absence of an analytical framework for this trade-off remains a significant gap in the learning-augmented literature. We address this gap by using predictions to design the first algorithms that minimize total completion time while using preemption parsimoniously. To our knowledge, this

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

is the first systematic study to analytically ground the trade-off between job latency and preemption complexity under job-size predictions. Our results move beyond the "blind" preemption required by classical non-clairvoyant models and provide the first bounded-preemption guarantees for unrelated and malleable machine settings.

1.1 Summary of our results

Problem setup: We assume all n jobs are available at time $t = 0$, because i) simultaneous arrival serves as a fundamental baseline in scheduling theory; and ii) while extensions to arbitrary arrival times are possible, they can obscure the core analytical ideas. We therefore provide a sketch of the extensions in the appendix to maintain a transparent presentation of our main results.

For each job j, we know its predicted requirement $\hat{p}_j$. In practice, these predictions $\hat{p}_j$ are typically derived by profiling the resource consumption of recurring jobs or by applying statistical models to historical execution traces from the cluster [18, 45]. A job j completes once it has been processed by its requirement p_j. However, the exact processing requirements remain unknown until completion; this setting is referred to as *non-clairvoyant scheduling* (when predictions are unavailable), in contrast to *clairvoyant scheduling*, where job sizes are known (upon arrival).

We focus on minimizing the total completion time objective, $\sum_j C_j$, where C_j is the completion time of job j. Since we evaluate the online algorithm relative to an optimum multiplicatively, our goal is equivalently stated as minimizing the average completion time. An algorithm is *c-competitive* if its total completion time is at most c times that of the offline optimum for all inputs.

We first present our findings under the assumption that all predictions are underestimates (i.e., $\hat{p}_j \leq p_j$ for all jobs j). Subsequently, we introduce a robust framework to handle overestimated job requirements. We define $D(p, \hat{p}) := \sum_j (\log_2(p_j/\hat{p}_j) + 1)$ as the ideal benchmark for the number of preemptions; we denote this simply as D when the context is clear. This benchmark is motivated by the fact that any non-clairvoyant algorithm performing $o(\log p_j)$ preemptions, even for a single job j, necessarily has a competitive ratio of $\Omega(n)$ (see Theorem A.1 in the appendix). Consequently, if $\hat{p}_j = 1$ for all j, any algorithm must perform $\Omega(D)$ preemptions to remain competitive.

Our results are summarized below. This is the *first* work to demonstrate how predictions can be utilized to minimize preemption. We provide constant-competitive algorithms while bounding the number of preemptions per job to within a constant factor of the benchmark D.

- **Single machine:** In the single-machine setting, which serves as a warm-up for our study, only one job can be processed at a time. For any $\varepsilon > 0$, we provide an algorithm that is $(2 + \varepsilon)$-competitive while incurring at most $O(\frac{1}{\varepsilon} D)$ preemptions (Theorem 2.1). This guarantee extends to identical parallel machines (Theorem C.1).
- **Unrelated machines:** This is our main result. In this heterogeneous parallel machine environment, each job j is processed at a rate of λ_{ij} on machine $i \in M$. A machine can process at most one job at a time. For this problem, we give a $14.48(1 + \varepsilon)$-competitive algorithm that performs at most $O(\frac{1}{\varepsilon^3} D)$ preemptions (Theorems 3.3 and 3.4).
- **Malleable job scheduling:** In this setting, jobs can be assigned to multiple machines (processors) simultaneously to expedite processing subject to eligibility constraints. Each job j is associated with a concave speed-up function that determines the processing rate based on the number of machines assigned. For any $\varepsilon > 0$, we provide an algorithm that is $27\frac{e}{e-1}(1 + \varepsilon)$-competitive, making at most $O(\frac{1}{\varepsilon^3} D)$ preemptions (Theorem 4.5); see Theorem F.1 for a slightly better bound on identical parallel machines.

We remark that our results extend readily to weighted jobs by considering the weighted objective $\sum_j w_j C_j$ and the weighted preemption benchmark $D(p, \hat{p}) := \sum_j w_j (\log_2(p_j/\hat{p}_j) + 1)$. For clarity of presentation, we focus on the unweighted case throughout the paper.

We also validate our algorithms through experiments, including heuristic adaptations of the theoretical versions developed in this paper. The results demonstrate that our algorithms effectively optimize total completion time while utilizing preemptions parsimoniously

Handling overestimation. Interestingly, overestimates are not symmetric to underestimates. To minimize total completion time, one must intuitively complete many jobs as quickly as possible [52].

Without knowledge about which jobs are short, we must hedge against short jobs being delayed by long ones via exponentially increasing our guesses; if a job does not complete after $(1+\delta)^k$ units of processing, we switch to other jobs, assuming its requirement is at least $(1+\delta)^{k+1}$, for some constant $\delta > 0$. Underestimations allow us to skip many initial guesses, but overestimations are less helpful: if all jobs are overestimated to the same requirement, the predictions provide no useful prioritization.

In fact, if we treat overestimated jobs identically to underestimated ones, the competitive ratio can degrade to $O(\max_j \hat{p}_j/p_j)$. A more robust approach is to *intentionally underestimate* job requirements by scaling down the initially provided estimates. This ensures that only a small number of jobs, g, remain overpredicted. We execute the algorithm for underestimated jobs until the number of unfinished jobs drops to g/ε. At this transition point, we reset the predictions for all remaining jobs to ensure they are henceforth treated as underestimates. This strategy maintains the competitive ratio while incurring only $O(\frac{g}{\varepsilon^2}\log_2 p_{\max})$ additional preemptions; see Theorem 2.2 for details.

As long as g is sublinear in n and job requirements are polynomial in n, a common and realistic scenario, the impact of overestimated jobs on the total number of preemptions is asymptotically negligible. Furthermore, a single global parameter g, which represents an upper bound on the number of overestimations, is arguably easier to predict accurately than jobs' individual sizes. This strategy successfully extends to the other settings considered in this work (Section D).

1.2 Our approaches and contributions

To understand the challenge of minimizing preemptions, note that we cannot use the popular Round-Robin (RR) algorithm. RR cycles through active jobs and processes each for one unit, resulting in prohibitively many preemptions. Despite this drawback, RR has been widely used in the learning-augmented literature because it is memoryless: its execution depends only on the set of currently active jobs [26, 37, 46]. Thus, if each instantaneous schedule approximates RR, we obtain a comparable performance guarantee; this is the basis of our approach.

To avoid excessive preemptions, we consider a less preemptive algorithm called Multi-Level Feedback (MLF) [44]. MLF maintains multiple FIFO queues to process jobs; for some constant $\delta > 0$, the k-th queue contains jobs that have been processed for at least $(1+\delta)^k$ and less than $(1+\delta)^{k+1}$ units of time. A non-clairvoyant algorithm, given no predictions, initially places each job in the queue corresponding to the smallest $k \geq 0$.

In our variant, which we call **Predicted Multi-Level Feedback (PMLF)**, a new job starts in the queue corresponding to its predicted requirement $\hat{p}_j$, rather than in the first queue. If all predictions are underestimates ($\hat{p}_j \leq p_j$), we show that PMLF achieves an $O(1)$-competitive ratio while incurring $O(D) = O(\sum_j(1+\log_{1+\delta}(p_j/\hat{p}_j)))$ preemptions. Hence, when predictions are accurate, we perform only $O(1)$ preemptions per job on average.

Our main result is the **Simulated Non-preemptive Adaptive Prediction (SNAP)** algorithm for unrelated machines. SNAP is designed to handle the complexities of unrelated machines, the most general parallel-machine model in classical scheduling theory [21, 34, 51]. Here, we cannot simply use a processing queue based on elapsed work, because the processing speeds λ_{ij} are unique to each job-machine pair: a given machine may be highly efficient for one job and extremely slow for another.

A prominent non-clairvoyant algorithm for unrelated machines is Proportional Fairness (PF), which identifies job processing rates that maximize their product, a task reducible to a convex program. This is not a computational bottleneck; rather, it enables efficient schedule computation, since the objective can be approximated to arbitrary precision in near-linear time [12]. However, a PF schedule is highly fractional (it alternates between multiple feasible schedules[1]) and is therefore highly preemptive. Furthermore, PF must recompute the rates whenever a job arrives or completes.

To address this, SNAP operates through a sequence of disjoint epochs. At the start of each epoch, the algorithm runs PF to compute the desired job processing rates. It then uses predicted job requirements to define the next checkpoint for each job: the smallest power of $1+\delta$ that is greater than both $\hat{p}_j$ and the current amount of processing the job has received. This strategy limits the total number of preemptions to a logarithmic factor of the multiplicative estimation error. We then choose an epoch duration so that, under PF, a β-fraction of jobs reach their next checkpoints. The preemptions made in the epoch are charged to the jobs that reached their checkpoints, and our algorithm remains

[1] A feasible schedule at each time can be thought of as a matching between jobs and machines.

$O(1)$-competitive because we follow PF for a $1-\beta$ fraction of jobs. Finally, we convert this virtual preemptive schedule into a non-preemptive schedule using an offline rounding procedure and run it.

For malleable jobs, however, we use a different approach. The reason is that the approximate super-additivity property is not known to hold for this problem, unlike in the unrelated-machines setting (see Lemma 3.10 for details). Therefore, we first analyze a more robust version of the PF algorithm. It is known that PF is $O(1)$-competitive for a general class of scheduling problems, called Polytope Scheduling (see Section 4 for the definition) [24, 28], which encompasses the malleable-jobs setting. To make PF less preemptive, we consider a version that does not change its rates until a constant fraction of jobs reach their next checkpoint. Here, unlike in SNAP, where we use the known performance guarantee of PF as a black box, we directly modify the analysis of PF from [24, 28]. This approach incurs at most $O(\frac{1}{\varepsilon^3}D)$ processing rate changes in total, and by simulating this algorithm as in SNAP with predictions, we obtain the desired result.

1.3 Further related work

We focus on the most closely related work here and defer a broader discussion to Appendix H.

Single and identical machines. In the clairvoyant setting, Shortest Job First (SJF), which processes jobs in non-decreasing order of processing time, is optimal for total completion time [52]. In the non-clairvoyant setting, RR is best-possible 2-competitive [44], but incurs excessively many preemptions. Multilevel (Adaptive) Feedback (MLF) reduces preemptions while remaining $O(1)$-competitive, and can be parameterized to be $(2+\varepsilon)$-competitive for any $\varepsilon > 0$ [44]. These guarantees for SJF and RR extend to parallel identical machines [7, 44].

Unrelated machines. In the clairvoyant online setting, utilizing immediate-dispatch and non-migratory algorithms while assuming each machine runs SPT, assigning an arriving job to the machine with the least marginal increase in the objective is $O(1)$-competitive [1, 20, 37]. For non-clairvoyant algorithms, SelfishMigrate and Proportional Fairness (PF) were analyzed with competitive ratios of 32 and 3.62, respectively [25, 28]. Clairvoyant algorithms were also explored in [9, 21, 39].

Online scheduling with predictions. Various objectives have been considered in the learning-augmented setting, see [38] for an overview. Examples include total completion time [15, 26, 31, 37, 46], total flow time [3, 40], makespan minimization [4, 32, 36], and energy consumption [2, 5].

2 Single machine

We start with our first result, which is for scheduling on a single machine. We first present our algorithm Predicted Multi-Level Feedback (PMLF) for a hyperparameter $\delta > 0$:

- Maintain a set of FIFO queues $\{Q_i\}_{i\geq 0}$. Initially, place a job j into Q_k where $\lfloor \log_{1+\delta} \hat{p}_j \rfloor$.
- At any time t, work on the front job of the non-empty queue of the lowest index.
- If $j \in Q_i$ was processed by in total by $(1+\delta)^{i+1}$, move j from Q_i to the end of Q_{i+1}.

PMLF becomes MLF when $\hat{p}_j = 1$ for each job j (under the standard assumption that $p_j \geq 1$). For now, we assume that $\hat{p}_j \leq p_j$ for each job j; we will soon discuss how to relax this assumption.

Theorem 2.1. *For any constant $\delta > 0$, PMLF is $(2+2\delta)$-competitive for minimizing the total completion time on a single machine and performs at most $O(\frac{1}{\delta}\log_2(p_j/\hat{p}_j))$ preemptions for each job j if there are only underpredictions.*

Proof. We first bound the number of preemptions. Note that j is only preempted when it is moved between queues. Since job j starts in $Q_{k'}$ where $(1+\delta)^{k'} \leq \hat{p}_j < (1+\delta)^{k'+1}$ and completes in Q_k where $(1+\delta)^k \leq p_j < (1+\delta)^{k+1}$, it is moved at most $k - k' + 1 = O(\log_{1+\delta}(p_j/\hat{p}_j)) = O(\frac{1}{\delta}\log_2(p_j/\hat{p}_j))$ times across queues.

We next prove an upper bound on the competitive ratio, for which we use a classic delay decomposition [44]. Fix a job j with $(1+\delta)^k \leq p_j < (1+\delta)^{k+1}$ and let C_j denote its completion time. Note that at the completion time C_j, it belongs to queue Q_k. Let $d(i,j)$ denote the total delay that job i causes to job j, which is the total amount of work done on i before time C_j. Clearly, $d(i,j) \leq p_i$. Moreover, since j completes at time C_j while belonging to queue Q_k, the algorithm can only work on i before C_j whenever i belongs to a queue $Q_{k'}$ with $k' \leq k$. This implies $d(i,j) < (1+\delta)^{k+1}$.

Thus, $d(i,j) \le \min\{p_i, (1+\delta)^{k+1}\}$, which is at most $(1+\delta)\min\{p_i, p_j\}$ because $p_j \ge (1+\delta)^k$. Since our algorithm does not idle,

$$C_j \le \sum_{i=1}^{n} d(i,j) \le (1+\delta)\sum_{i=1}^{n} \min\{p_i, p_j\} \,.$$

Summing over all jobs j gives

$$\sum_{j=1}^{n} C_j \le (1+\delta)\sum_{j=1}^{n}\sum_{i=1}^{n} \min\{p_i, p_j\} \le (1+\delta)\cdot 2\mathsf{opt} \,,$$

where the second inequality follows the fact that scheduling jobs in non-decreasing order of their processing times is optimal [52], thus, $\mathsf{opt} = \sum_j p_j + \sum_{j=1}^{n}\sum_{i>j}^{n} \min\{p_i, p_j\}$. □

Handling overestimations. If we allow overestimations, let g be an upper bound on the number of overestimated jobs. By scaling down predicted sizes, we can make g small at the cost of more preemptions. We consider a variation of PMLF, which uses g and a parameter $\gamma > 0$: Let T be the time when the number of unfinished jobs is at most g/γ, and $q_j(T)$ denote the cumulative processed size of job j until time T. At time T, move each unfinished job j to the end of the queue Q_k where $(1+\delta)^k \le q_j(T) < (1+\delta)^{k+1}$. Then, we continue with PMLF. We prove in Appendix B:

Theorem 2.2. *If there are at most g overestimated jobs, then for every $\varepsilon > 0$, the adapted PMLF is $2(1+\varepsilon)$-competitive and incurs at most $O(\frac{1}{\varepsilon}\sum_j (1+\log_2 \frac{p_j}{\hat{p}_j}) + \frac{g}{\varepsilon^2}\log_2(\max_j p_j))$ preemptions.*

3 Unrelated machines

In this section, we describe SNAP for scheduling on unrelated machines. We are given a set M of unrelated machines. Each job j is available for scheduling from time 0, and we know its predicted size $\hat{p}_j$ and processing rates $\{\lambda_{ij}\}_{i\in M}$ across all machines. We assume that every job j has processing requirement equal to a power of $1+\delta$ for some $\delta > 0$; that is, $p_j = (1+\delta)^i$ for some integer $i \ge 0$. If p_j is not a power of $1+\delta$, we instead treat the job as active until it has been processed for $(1+\delta)^{\lceil \log_{1+\delta} p_j \rceil}$ units of time. This increases the competitive ratio by a factor of at most $1+\delta$. We assume that all job sizes are underestimated, i.e., $\hat{p}_j \le p_j$ for all jobs j. The case of overestimation is addressed in Appendix D, following the same approach as for the single-machine case.

Before describing SNAP, we introduce *checkpoints*. We say that a job j reaches a *checkpoint* at time t if its cumulative processed size $q_j(t)$ reaches a power of $1+\delta$. For convenience, when a job reaches a checkpoint, we say that it is *exhausted* (or that an *exhaustion* occurs). For job j, we begin tracking checkpoints at its predicted size $\hat{p}_j$: let $(1+\delta)^{\rho_j}$ be its first checkpoint, where $\rho_j := \lceil \log_{1+\delta} \hat{p}_j \rceil$. Thus, the checkpoints of j are $\{(1+\delta)^{\rho'}\}_{\rho' \ge \rho_j}$, and job j can reach a checkpoint at most $\lceil \log_{1+\delta}(p_j/\hat{p}_j) \rceil + 1$ times.

3.1 Algorithm: SNAP

Our algorithm, which we call **Simulated Non-preemptive Adaptive Prediction (SNAP)** algorithm, proceeds in epochs $1, 2, \ldots, K$. Epoch k starts at time e_k and ends at time e_{k+1}, at which point epoch $k+1$ begins immediately. The first epoch starts at time $e_1 = 0$. At the start of epoch k, let $J_k := J(e_k)$ denote the set of available jobs for processing. Here, $J(t) := \{j \mid r_j \le t < C_j\}$ is the set of jobs alive at time t. Let $n_k := |J_k|$. Let $\beta, \gamma \in (0,1)$ be some constants that we optimize later.

At the beginning of each epoch, we compute a complete schedule for the duration of the epoch and follow it. Specifically, we execute the following steps:

1. **Compute Proportional Fairness (PF) processing rates.** We compute the PF rates for the set J_k, denoted by $y^{(k)} := \mathrm{PF}(J_k)$. Formally, $y^{(k)}$ is the solution to the following convex

program with $J = J_k$:

$$\text{PF}(J): \quad \max \quad \sum_{j \in J} \log(y_j)$$
$$\text{s.t.} \quad \sum_{j \in J} x_{ij} \leq 1 \quad \forall i \in M, \quad \sum_{i \in M} x_{ij} \leq 1, \quad y_j = \sum_{i \in M} \lambda_{ij} x_{ij} \quad j \in J,$$
$$x_{ij} \geq 0 \quad \forall i \in M,\ j \in J.$$

This can be solved approximately in near linear time [12].

2. **Compute remaining requirement to the next checkpoint.** For each job $j \in J_k$, we define the remaining processing requirement to reach its next checkpoint as $u_{j,k} := (1+\delta)^{h+1} - q_j(e_k)$, where $h \in \mathbb{N}$ such that $(1+\delta)^h \leq \max\{\hat{p}_j, q_j(e_k)\} < (1+\delta)^{h+1}$.
3. **Determine the simulation duration.** Let l_k denote the length of the shortest time interval such that, if jobs are processed at rates $y^{(k)}$ during the time interval, at least $\lceil \beta n_k \rceil$ jobs from J_k reach their next checkpoint. Formally, let the jobs in J_k be sorted in non-decreasing order of $u_{j,k}/y_j^{(k)}$. If j^* is the $\lceil \beta n_k \rceil$-th job in this ordering, breaking ties arbitrarily, then the duration is defined as $l_k = u_{j^*,k}/y_{j^*}^{(k)}$. The targeted processing requirement for each job $j \in J_k$ in the current epoch is then $v_{j,k} = \min\{u_{j,k}, l_k y_j^{(k)}\}$.
4. **Convert to a non-preemptive schedule.** Since the fractional assignments x_{ij} from Step 1 constitute a valid preemptive schedule for the required processing requirement $v_{j,k}$, we can construct a non-preemptive (and non-migratory) schedule that begins at time e_k and completes by time $e_k + 4 \cdot l_k$ where each each job $j \in J_k$ is processed by $v_{j,k}$ units [11].

We note that the algorithm runs in polynomial time, as both the conversion method [11] and the computation of PF rates [24, 28] are polynomial-time solvable. From the description of SNAP, we can make the following two observations.

Observation 3.1. *In an epoch k, at least $\lceil \beta n_k \rceil$ jobs reach a checkpoint.*

Observation 3.2. *For each epoch k, we have that $e_{k+1} - e_k \leq 4 \cdot l_k$.*

We first argue about the number of preemptions and migrations of SNAP.

Theorem 3.3. *When all job sizes are underestimated, the total number of preemptions performed by SNAP is at most $O\big(\frac{1}{\beta\delta} \sum_j \big(\log_2 \frac{p_j}{\hat{p}_j} + 1\big)\big)$.*

Proof. In each epoch k, every job $j \in J_k$ is preempted (or migrated) at most once due to the non-preemptive nature of the schedule constructed in Step 4. Thus, the total number of preemptions is bounded by $\sum_k n_k$. By Observation 3.1, at least βn_k jobs reach a checkpoint during an epoch. Consequently, $\sum_k n_k \leq \frac{1}{\beta} \times$ (Total number of exhaustions). Each job j can reach a checkpoint at most $\lceil \log_{1+\delta}(p_j/\hat{p}_j) \rceil + 1$ times (under the assumption of underestimated predictions). Therefore,

$$\textstyle\sum_k n_k \leq \frac{1}{\beta} \sum_j \big(\lceil \log_{1+\delta} \frac{p_j}{\hat{p}_j} \rceil + 1\big) = O\big(\frac{1}{\beta\delta} \sum_j \big(1 + \log_2 \frac{p_j}{\hat{p}_j}\big)\big)\ . \qquad \square$$

We next aim to prove the following theorem on SNAP's competitive ratio.

Theorem 3.4. *For any $\varepsilon > 0$, the SNAP algorithm with $\eta = \frac{\varepsilon}{2(1+\varepsilon)}$ and $\beta = \frac{\varepsilon^2}{4(1+\varepsilon)(2+\varepsilon)}$. achieves a total completion time that is at most $14.48(1+\varepsilon)$ times the optimum.*

We split up the proof of this theorem into multiple lemmas. From now on fix the schedule of the SNAP algorithm. Let K be the number of epochs. Let $\mathsf{alg}(p)$ denote SNAP's total completion time objective for a processing time vector $p = \{p_j\}_j$.

Lemma 3.5. *It holds that $\mathsf{alg}(p) \leq 4 \sum_{k=1}^{K} n_k l_k$.*

Proof. We have

$$\mathsf{alg}(p) = \sum_{k=1}^{K} \sum_{t=e_k}^{e_{k+1}-1} |J(t)| \leq \sum_{k=1}^{K} \sum_{t=e_k}^{e_{k+1}-1} |J(e_k)| = \sum_{k=1}^{K} (e_{k+1} - e_k) n_k \leq 4 \sum_{k=1}^{K} l_k n_k\ ,$$

where the equalities use that each alive job contributes 1 per timestep and $n_k = |J(e_k)|$, and the inequalities use monotonicity of $|J(t)|$ and Observation 3.2. □

Let $\mathsf{opt}(z_j \mid j \in S)$ denote the minimum total completion time to complete a set of jobs S, where job $j \in S$ has a size z_j. For the sake of analysis, consider a set of jobs where we have one job for each $j \in J_k$, but with a different size, $l_k y_j^{(k)}$. Since $y_j^{(k)}$ is PF's processing rate on j when it runs on J_k, PF will finish them with these hypothetical sizes exactly at the same time. Using Theorem 4.1 of [28], which shows that PF achieves a competitive ratio of two when it completes all jobs with the same arrival time at the same time, we have the following claim.

Claim 3.6. *For each epoch k, we have $n_k l_k \le 2 \cdot \mathsf{opt}(l_k y_j^{(k)} \mid j \in J_k)$.*

Let X_k be the jobs exhausted during epoch k, strictly before the end of the epoch. Let $Y_k := J_k \setminus X_k$. Note that $|Y_k| \ge (1-\beta) n_k$ because of the definition of the epoch. Our next goal is to bound the RHS of the inequality in Claim 3.6 by a linear combination of analogous quantities restricted to X_k and Y_k, respectively. The proof is deferred to Appendix D.1.

Claim 3.7. *For each epoch k and any $\eta \in (0,1)$, we have,*

$$\mathsf{opt}(l_k y_j^{(k)} \mid j \in J_k) \le \frac{1}{\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k) + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)\ .$$

The following lemma upper bounds the cost incurred by our algorithm during epoch k by the cost that the optimal solution must pay for the processing performed on jobs that are not exhausted in the middle of the epoch. We defer to Appendix D.1 the proof of the following lemma, which follows by combining Claims 3.6 and 3.7 with $|X_k| \le \beta n_k$.

Lemma 3.8. *For each epoch k, we have $n_k l_k \le \frac{2\eta}{(\eta-2\beta)(1-\eta)} \cdot \mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)$.*

We then show that $\{l_k y^{(k)}\}_{j \in Y_k}$ is a decomposition of the full instance.

Lemma 3.9. *For every job j, it holds that $\sum_{k: j \in Y_k} l_k y_j^{(k)} \le p_j$.*

Proof. Fix a job j. Consider an epoch k with $j \in Y_k$. Since j is not exhausted until epoch k's end time, it is processed by $l_k y_j^{(k)}$ during the epoch. The sum of this quantity over all epochs k with $j \in Y_k$ cannot exceed j's size, p_j. □

We now employ the "decomposition" lemma, which follows from the super-additivity of the non-preemptive optimum for unrelated machine scheduling [28] and the 1.81-gap between the preemptive and non-preemptive optima for the total (weighted) completion time objective [50].

Lemma 3.10 (Lemma 4.5 in [28])**.** *Consider any set of jobs S where each job j has a size p_j. For each $j \in S$, let $p_{j,\kappa'} \ge 0$, $1 \le \kappa' \le \kappa$, be such that $\sum_{\kappa'=1}^{\kappa} p_{j,\kappa'} \le p_j$. Then, we have $\sum_{\kappa'=1}^{\kappa} \mathsf{opt}(p_{j,\kappa'} \mid j \in S) \le 1.81 \cdot \mathsf{opt}(p_j \mid j \in S)$.*

Lemmas 3.9 and 3.10 immediately yield the following.

Corollary 3.11. *It holds that $\sum_{k=1}^{K} \mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k) \le 1.81 \cdot \mathsf{opt}(p)$.*

We are now ready to prove Theorem 3.4. Using Lemma 3.5, Lemma 3.8, and Corollary 3.11, we obtain

$$\begin{aligned}\mathsf{alg}(p) \le 4\sum_{k=1}^{K} n_k l_k &\le 4\big(\tfrac{2\eta}{(\eta-2\beta)(1-\eta)}\big)\sum_{k=1}^{K}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)\\ &\le 14.48\big(\tfrac{\eta}{(\eta-2\beta)(1-\eta)}\big)\cdot \mathsf{opt}(p)\ .\end{aligned}$$

Hence, $\mathsf{alg} \le 14.48 \cdot (1+\varepsilon) \cdot \mathsf{opt}(p)$ using $\eta = \frac{\varepsilon}{2(1+\varepsilon)}$ and $\beta = \frac{\varepsilon^2}{4(1+\varepsilon)(2+\varepsilon)}$.

4 A general framework for PSP and malleable job scheduling

We next present a general framework for the Polytope Scheduling Problem (PSP) with parsimonious preemption, which encompasses various scheduling problems [24, 28]. There are n jobs $j = 1, \ldots, n$ with processing requirements p_j. A feasible schedule selects over time for every time t a processing rate $y_j(t)$ for each active job $j \in J(t)$ from a given downward-closed polytope $\mathcal{P} = \{y \in \mathbb{R}^n_{\geq 0} \mid By \leq \mathsf{1}\}$ for some matrix $B = [b_{dj}] \in \mathbb{R}^{D \times n}_{\geq 0}$. Job j completes when it has received p_j amount of processing, that is, its completion time C_j is the earliest point in time t' such that $\int_{r_j}^{t'} y_j(t)\,\mathrm{d}t \geq p_j$. The goal is to minimize the total completion time $\sum_j C_j$. We again make the assumption that all jobs are underpredicted; we can mitigate this assumption as in the previous sections.

The framework. Our framework is a generalization of SNAP in (cf. Section 3). The main difference lies in Step 4, where we assume to be given a procedure to convert a rate schedule (a vector belonging to $\mathcal{P}$) into a schedule with few preemptions.

Definition 4.1 (α-approximate $O(1)$-preemptive schedule)**.** For a given PSP schedule of length l with fixed allocation $\{y_j\}_{j \in J}$, an *α-approximate $O(1)$-preemptive schedule* satisfies processing requirements $\{ly_j\}_{j \in J}$, has a makespan of at most αl, and uses only $O(1)$ preemptions per job.

The definition of preemption depends on the specific PSP, as each polytope $\mathcal{P}$ determines a distinct scheduling problem. Consequently, Definition 4.1 allows us to reason within the abstract setting of the PSP, providing a general framework for our analysis. We again assume that all p_j's are powers of $1 + \delta$ for some constant $\delta > 0$, which increases the competitive ratio by at most a factor of $1 + \delta$.

Our framework generalizes the SNAP algorithm. Thus, we focus here on the key modifications and defer the full description to Appendix E. The primary distinction lies in Step 4, which is adapted as follows: we compute an α-approximate $O(1)$-preemptive schedule (as per Definition 4.1) for the set of jobs J_k. For each job $j \in J_k$, the processing targets are defined as $v_{j,k} = \min\{u_{j,k}, l_k y_j^{(k)}\}$. This schedule starts at time e_k and is guaranteed to finish no later than $e_{k+1} \leq e_k + \alpha l_k$.

Analysis. Bounding the number of preemptions is analogous to Theorem 3.3 (see Theorem E.1). For the analysis of the competitive ratio, for convenience we analyze our algorithm before the conversion to the α-approximate $O(1)$-preemptive schedule, which we will call the *virtual algorithm*; in this case, we effectively have $e_{k+1} = e_k + l_k$. This assumption does not alter the processing each job receives within an epoch. However, the conversion increases each job's completion time by a factor of at most α, resulting in a multiplicative loss in the competitive ratio by a factor of α.

The primary departure from the analysis of SNAP for unrelated machines is that PSP lacks the super-additivity guarantees for subinstances (cf. Lemma 3.10). Consequently, we must use different techniques to bound the objective. Our main technical novelty is the introduction of a *robust* variant of PSP, which we term β-robust PSP, and a corresponding analysis of PF within this framework.

Definition 4.2. In the β-robust PSP problem, we are given a PSP instance and a constant $\beta \in [0, 1)$. At any time t, an algorithm selects a processing allocation $y(t) \in \mathcal{P}$ for the set of active jobs $J(t)$. An adversary then chooses a subset $N(t) \subseteq J(t)$ of at most $\beta|J(t)|$ jobs whose processing rates are nullified. Formally, the resulting processing allocation is: $\hat{y}_j(t) := y_j(t) \cdot \mathsf{1}[j \notin N(t)]$.

We show in Appendix G the following theorem.

Theorem 4.3. *For any $\beta \in [0, 1)$, PF is $27/(1-\beta)^3$-competitive for β-robust PSP.*

The key observation is that the schedule of the virtual algorithm can be interpreted as a schedule of PF for β-robust PSP with a specific choice of the set of nullified jobs. Using this observation, we show in Appendix E that the virtual algorithm achieves a total completion time at most $27\frac{(1+\delta)}{(1-\beta)^3}$ times the optimum. Setting $\delta = (1+\varepsilon)(1-\beta)^3 - 1$ and $\beta = 1 - ((1+\delta)/(1+\varepsilon))^{1/3}$ yields:

Theorem 4.4. *Given a procedure for computing α-approximate $O(1)$-preemptive schedules, there exists a $27\alpha(1+\varepsilon)$-competitive algorithm for PSP that performs at most $O(\frac{1}{\varepsilon^3}\sum_j(1 + \log_2 p_j/\hat{p}_j))$ preemptions for any $\varepsilon > 0$.*

Finally, we can apply this theorem to malleable job scheduling on unrelated machines. Details are deferred to Appendix F.

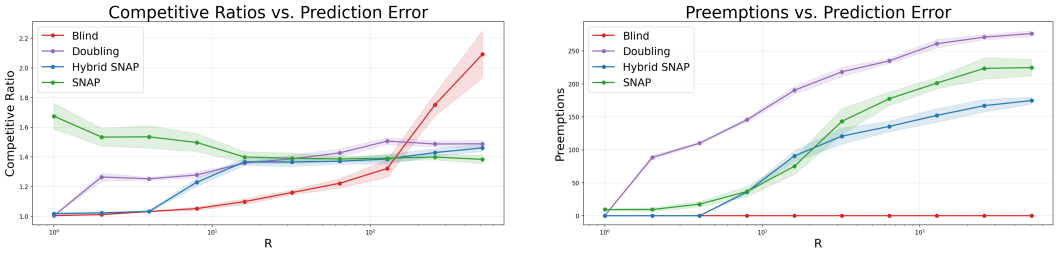


Figure 1: Left: Competitive ratio vs. prediction error parameter R. Performance of the Blind strategy degrades significantly as error increases, whereas SNAP remains stable. Right: Average preemptions per job vs. R. SNAP exhibits more parsimonious preemption behavior. Hybrid SNAP (with $c = 4$) improves upon SNAP for small values of R by leveraging the strengths of the Blind and Doubling.

Theorem 4.5. *For any $\varepsilon > 0$, there is a $27\frac{e}{e-1}(1+\varepsilon)$-competitive algorithm that incurs at most $O(\frac{1}{\varepsilon^3}\sum_j(1+\log_2 p_j/\hat{p}_j))$ preemptions for minimizing the total completion time of malleable jobs on unrelated machines with non-decreasing, concave, and non-decreasing work speed-up functions.*

5 Experiments

To the best of our knowledge, no existing algorithms limit preemption using predictions while simultaneously optimizing for the total completion time objective. To address this gap, we developed and implemented a suite of algorithms, including SNAP. With the exception of the Blind strategy, all of these algorithms incorporate novel algorithmic aspects. We use a default value of $\delta = 1$. For additional implementation details, see Appendix I.1.

SNAP: Our primary algorithm, modified to eliminate idle time by running PMLF on each machine until a β fraction of jobs are exhausted. Specifically, we follow Step 4 to assign jobs to machines in an epoch, which are then processed by PMLF on each machine. **Doubling:** A variant of the immediate-dispatch algorithm in [1] (where each machine runs SJF and new jobs are assigned to minimize the marginal increase of the objective). This variant doubles a job's estimated size whenever processing exceeds the current estimate and re-dispatches it to the machine that minimizes the objective increase. **Blind:** Based on the immediate-dispatch and non-migratory algorithm in [1], this strategy is executed by assuming $p_j = \hat{p}_j$ for all jobs. This algorithm makes no preemptions. **Hybrid SNAP:** A modified SNAP algorithm that begins with Doubling with large milestones $\{c(1+\delta)^i\hat{p}_j \mid i \geq 1\}$ for a large parameter $c \geq 10$ (to emulate the Blind strategy). Jobs that have reached several milestones are processed by SNAP.

Setup: We evaluated the algorithms on synthetic instances with $m = 10$ machines and $n = 100$ jobs. To model heterogeneity, 20% of jobs are designated as "special" ($p_j \sim \mathcal{U}[1, 200]$) and are restricted to 2 specific machines, while regular jobs ($p_j \sim \mathcal{U}[1, 10]$) can utilize any resource. Intuitively, special jobs represent tasks with large processing requirements that require specific hardware. (For other percentages of special jobs and machines we obtain similar results; see Appendix I).

The prediction error is controlled via a parameter $R \geq 1$; we set $\hat{p}_j = \lceil p_j/\xi_j \rceil$ where $\xi_j \sim \mathcal{U}[1, R]$. We evaluate the algorithms based on the total completion time objective normalized to the optimum (the competitive ratio) and the average number of preemptions per job. Comparisons are performed across exponentially increasing values of R. We note that large prediction errors are common in practice [33, 45]; in particular, even top-tier database systems routinely misestimate join sizes, which directly dictate execution runtimes, by factors of 1000 or more.

Results: Our experiments demonstrate that SNAP effectively optimizes total completion time while requiring fewer preemptions than the doubling algorithm, particularly when prediction errors are large. These findings are summarized in Figure 1. Our main observations are as follows:

- Our results show that we can effectively **leverage the highly preemptive PF algorithm to derive a preemption-parsimonious algorithm** like SNAP. SNAP incurs fewer preemptions than natural benchmarks such as Doubling.

- It is known that Blind is 4-competitive if all predictions are accurate [1], but its competitive ratio degrades linearly with error. Thus, Blind performs well for small R but becomes impractical as R increases.
- To achieve the "best of both worlds," we implemented Hybrid SNAP. This experimental approach yields promising results in terms of both objective value and minimizing preemptions. Hybrid SNAP performed comparably to Blind when R is small and similarly to SNAP when R is large.

Further experimental results, including a detailed analysis of the tradeoff between the competitive ratio and preemptions as a function of δ, are deferred to Appendix I.

6 Conclusions

In this work, we presented the first analytical study of the trade-off between job latency minimization and preemption overhead in online scheduling with predictions. By leveraging machine-learned predictions, our SNAP and PMLF algorithms avoid the excessive preemption required by classical non-clairvoyant algorithms. We proved that these algorithms are constant-competitive in various environments, including single, identical, and unrelated machines, with preemption costs that scale logarithmically with the prediction error. Our experiments show that SNAP remains stable in minimizing both preemption overhead and total completion time, and also demonstrate the favorable performance of the other heuristics developed in this study.

Future research could further explore hybridizing the Blind and SNAP frameworks. This approach seeks to leverage the efficiency of Blind under accurate predictions while maintaining the robustness of SNAP in the presence of large errors, effectively bridging the gap between optimistic performance and worst-case reliability.

Acknowledgments and Disclosure of Funding

Part of this work was done while AL was a research fellow in the Simons Institute program on "Algorithmic Foundations for Emerging Computing Technologies" in Fall 2025. AL was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – The Berlin Mathematics Research Center MATH$^+$ (EXC-2046/1, EXC-2046/2, project ID: 390685689). MB and SI were supported in part by NSF grants CCF-2423106, CCF-2537126, and CCF-2535599.

References

[1] S. Anand, N. Garg, and A. Kumar. Resource augmentation for weighted flow-time explained by dual fitting. In: *SODA*. SIAM, 2012, pp. 1228–1241.

[2] A. Antoniadis, P. Jabbarzade, and G. Shahkarami. A Novel Prediction Setup for Online Speed-Scaling. In: *SWAT*. Vol. 227. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 9:1–9:20.

[3] Y. Azar, S. Leonardi, and N. Touitou. Flow time scheduling with uncertain processing time. In: *STOC*. ACM, 2021, pp. 1070–1080.

[4] E. Balkanski, T. Ou, C. Stein, and H. Wei. Scheduling with Speed Predictions. In: *Theory Comput. Syst.* 69.1 (2025), p. 16.

[5] É. Bamas, A. Maggiori, L. Rohwedder, and O. Svensson. Learning Augmented Energy Minimization via Speed Scaling. In: *NeurIPS*. 2020.

[6] N. Bansal, A. Srinivasan, and O. Svensson. Lift-and-round to improve weighted completion time on unrelated machines. In: *STOC*. ACM, 2016, pp. 156–167.

[7] O. Beaumont, N. Bonichon, L. Eyraud-Dubois, and L. Marchal. Minimizing Weighted Mean Completion Time for Malleable Tasks Scheduling. In: *IPDPS*. IEEE Computer Society, 2012, pp. 273–284.

[8] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[9] S. Chakrabarti, C. A. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, and J. Wein. Improved Scheduling Algorithms for Minsum Criteria. In: *ICALP*. Vol. 1099. LNCS. Springer, 1996, pp. 646–657.

[10] W. Chen, J. Rao, and X. Zhou. Preemptive, Low Latency Datacenter Scheduling via Lightweight Virtualization. In: *USENIX ATC*. USENIX Association, 2017, pp. 251–263.

[11] J. R. Correa, M. Skutella, and J. Verschae. The Power of Preemption on Unrelated Machines and Applications to Scheduling Orders. In: *Math. Oper. Res.* 37.2 (2012), pp. 379–398.

[12] F. Criado, D. Martínez-Rubio, and S. Pokutta. Fast Algorithms for Packing Proportional Fairness and its Dual. In: *NeurIPS*. 2022.

[13] R. I. Davis and M. Bertogna. Optimal Fixed Priority Scheduling with Deferred Pre-emption. In: *RTSS*. IEEE Computer Society, 2012, pp. 39–50.

[14] X. Deng, N. Gu, T. Brecht, and K. Lu. Preemptive Scheduling of Parallel Jobs on Multiprocessors. In: *SIAM J. Comput.* 30.1 (2000), pp. 145–160.

[15] M. Dinitz, S. Im, T. Lavastida, B. Moseley, and S. Vassilvitskii. Algorithms with Prediction Portfolios. In: *NeurIPS*. 2022.

[16] J. Edmonds. Scheduling in the Dark. In: *Theor. Comput. Sci.* 235.1 (2000), pp. 109–141.

[17] E. Eisenberg and D. Gale. Consensus of Subjective Probabilities: The Pari-Mutuel Method. In: *Ann. Math. Statist.* 30.1 (1959), pp. 165–168.

[18] A. B. Faisal, N. Martin, H. M. Bashir, S. Lamelas, and F. R. Dogar. When will my ML Job finish? Toward providing Completion Time Estimates through Predictability-Centric Scheduling. In: *OSDI*. 2024, pp. 487–505.

[19] D. Fotakis, J. Matuschke, and O. Papadigenopoulos. Malleable scheduling beyond identical machines. In: *J. Sched.* 26.5 (2023), pp. 425–442.

[20] V. Gupta, B. Moseley, M. Uetz, and Q. Xie. Greed Works - Online Algorithms for Unrelated Machine Stochastic Scheduling. In: *Math. Oper. Res.* 45.2 (2020), pp. 497–516.

[21] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein. Scheduling to Minimize Average Completion Time: Off-Line and On-Line Approximation Algorithms. In: *Math. Oper. Res.* 22.3 (1997), pp. 513–544.

[22] D. G. Harris. Dependent rounding with strong negative-correlation, and scheduling on unrelated machines to minimize completion time. In: *SODA*. SIAM, 2024, pp. 2275–2304.

[23] H. Hoogeveen, P. Schuurman, and G. J. Woeginger. Non-approximability Results for Scheduling Problems with Minsum Criteria. In: *IPCO*. Vol. 1412. Springer, 1998, pp. 353–366.

[24] S. Im, J. Kulkarni, and K. Munagala. Competitive Algorithms from Competitive Equilibria: Non-Clairvoyant Scheduling under Polyhedral Constraints. In: *J. ACM* 65.1 (2018), 3:1–3:33.

[25] S. Im, J. Kulkarni, K. Munagala, and K. Pruhs. SelfishMigrate: A Scalable Algorithm for Non-clairvoyantly Scheduling Heterogeneous Processors. In: *FOCS*. IEEE Computer Society, 2014, pp. 531–540.

[26] S. Im, R. Kumar, M. M. Qaem, and M. Purohit. Non-clairvoyant Scheduling with Predictions. In: *ACM Trans. Parallel Comput.* 10.4 (2023), 19:1–19:26.

[27] S. Im and S. Li. Improved Approximations for Unrelated Machine Scheduling. In: *SODA*. SIAM, 2023, pp. 2917–2946.

[28] S. Jäger, A. Lindermayr, and N. Megow. The Power of Proportional Fairness for Nonclairvoyant Polytope Scheduling. In: *SIAM J. Comput.* 55.2 (2026), pp. 247–279.

[29] S. Jäger, G. Sagnol, D. Schmidt genannt Waldschmidt, and P. Warode. Competitive Kill-and-Restart and Preemptive Strategies for Non-clairvoyant Scheduling. In: *IPCO*. Vol. 13904. LNCS. Springer, 2023, pp. 246–260.

[30] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. In: *J. ACM* 47.4 (2000), pp. 617–643.

[31] A. A. Lassota, A. Lindermayr, N. Megow, and J. Schlöter. Minimalistic Predictions to Schedule Jobs with Online Precedence Constraints. In: *ICML*. PMLR, 2023, pp. 18563–18583.

[32] S. Lattanzi, T. Lavastida, B. Moseley, and S. Vassilvitskii. Online Scheduling via Learned Weights. In: *SODA*. SIAM, 2020, pp. 1859–1877.

[33] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? In: *Proceedings of the VLDB Endowment* 9.3 (2015), pp. 204–215.

[34] J. K. Lenstra, D. B. Shmoys, and É. Tardos. Approximation algorithms for scheduling unrelated parallel machines. In: *Mathematical programming* 46.1 (1990), pp. 259–271.

[35] S. Li. Approximating unrelated machine weighted completion time using iterative rounding and computer assisted proofs. In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM. 2025, pp. 553–571.

[36] S. Li and J. Xian. Online Unrelated Machine Load Balancing with Predictions Revisited. In: *ICML*. Proceedings of Machine Learning Research. PMLR, 2021, pp. 6523–6532.

[37] A. Lindermayr and N. Megow. Permutation Predictions for Non-Clairvoyant Scheduling. In: *ACM Trans. Parallel Comput.* 12.2 (2025), 4:1–4:26.

[38] A. Lindermayr and N. Megow. *Repository of papers on algorithms with predictions*. `http://algorithms-with-predictions.github.io/`. 2025.

[39] A. Lindermayr, N. Megow, and M. Rapp. Speed-Oblivious Online Scheduling: Knowing (Precise) Speeds is not Necessary. In: *ICML*. Vol. 202. PMLR. 2023, pp. 21312–21334.

[40] M. Mitzenmacher. Scheduling with Predictions and the Price of Misprediction. In: *ITCS*. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 14:1–14:18.

[41] M. Mitzenmacher and S. Vassilvitskii. Algorithms with Predictions. In: *Commun. ACM* 65.7 (2022), pp. 64–73.

[42] J. C. Mogul and A. Borg. The Effect of Context Switches on Cache Performance. In: *ASPLOS*. ACM Press, 1991, pp. 75–84.

[43] B. Moseley and S. Vardi. The efficiency-fairness balance of Round Robin scheduling. In: *Oper. Res. Lett.* 50.1 (2022), pp. 20–27.

[44] R. Motwani, S. J. Phillips, and E. Torng. Non-Clairvoyant Scheduling. In: *Theor. Comput. Sci.* 130.1 (1994), pp. 17–47.

[45] J. W. Park, A. Tumanov, A. Jiang, M. A. Kozuch, and G. R. Ganger. 3sigma: distribution-based cluster scheduling for runtime uncertainty. In: *EuroSys*. 2018, pp. 1–17.

[46] M. Purohit, Z. Svitkina, and R. Kumar. Improving Online Algorithms via ML Predictions. In: *NeurIPS*. 2018, pp. 9684–9693.

[47] M. Saksena and Y. Wang. Scalable Real-Time System Design using Preemption Thresholds. In: *RTSS*. IEEE Computer Society, 2000, pp. 25–34.

[48] A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. In: *SIAM J. Discrete Math.* 15.4 (2002), pp. 450–469.

[49] J. Sethuraman and M. S. Squillante. Optimal Scheduling of Multiclass Parallel Machines. In: *SODA*. ACM/SIAM, 1999, pp. 963–964.

[50] R. Sitters. Approximability of average completion time scheduling on unrelated machines. In: *Math. Program.* 161.1-2 (2017), pp. 135–158.

[51] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. In: *Journal of the ACM (JACM)* 48.2 (2001), pp. 206–242.

[52] W. E. Smith. Various optimizers for single-stage production. In: *Nav. Res. Logist. Q.* 3.1-2 (1956), pp. 59–66.

[53] D. Tsafrir. The context-switch overhead inflicted by hardware interrupts (and the enigma of do-nothing loops). In: *Experimental Computer Science*. ACM, 2007, p. 4.

[54] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at Google with Borg. In: *EuroSys*. ACM, 2015, 18:1–18:17.

[55] T. Zhao, W. Li, and A. Y. Zomaya. Real-Time Scheduling with Predictions. In: *RTSS*. IEEE, 2022, pp. 331–343.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

Justification: The abstract and Section 1.1 state the main contributions, assumptions, and scope, including constant-competitive guarantees for single, unrelated, and malleable machine settings and experiments in Section 5. These claims are supported by the corresponding theoretical results in Sections 2–4 and the empirical evaluation in Section 5.

Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: The paper discusses key scope limitations and assumptions in Section 1.1, including simultaneous arrivals, the main underprediction setting, and how overpredictions are handled. In the appendix we sketch arrivals over time and overestimation of job sizes, and Section 6 notes future directions.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper states assumptions with each theoretical setting, such as underpredicted sizes, simultaneous arrivals, and model-specific processing constraints. Proofs or proof sketches are provided in the main text and full proofs are deferred to the appendix for the main theorems and lemmas.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The experiments use synthetic instances whose distributions, prediction-error construction, metrics, and baselines are described in Section 5, with additional implementation details in the appendix. There we also state that the full source code and plotting scripts are provided in the supplemental material.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: The experimental data are synthetically generated as described in Section 5, and in the appendix we state that the full source code is provided in the supplemental material. The algorithmic and plotting details needed to reproduce the reported figures are described in Section 5 and in the appendix

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section 5 specifies the experimental instance sizes, machine/job distributions, prediction-error model, default δ, evaluated metrics, and compared algorithms. Appendix I gives further details for each implemented algorithm.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We give confidence bars for our main experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The generation of synthetic instances and the simulation of our algorithms does not require special hardware and can be executed on an average computer.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The work is theoretical and uses synthetic experiments, with no human-subject data, private data, or high-risk model release. To the best of our knowledge, the research conforms to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: Our results do not have immediate or almost immediate consequences on society.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: The paper does not release data or models with high risk for misuse. The released material is experimental code and instances for synthetic scheduling simulations.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [N/A]

Justification: The paper does not use existing datasets, models, or external code assets as experimental inputs; the experiments use synthetic data generated by the authors. Prior algorithms and theoretical results are credited through citations in the relevant sections.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The new asset is the experimental source code and plotting scripts, which are documented in the supplemental material.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, data collection from workers, or research with human subjects. Therefore participant instructions, screenshots, and compensation details are not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: The paper does not involve crowdsourcing or human-subject research. Therefore IRB or equivalent approval and participant-risk disclosures are not applicable.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [No]

Justification: We do not use LLMs as a non-standard component of our core methods.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

A Lower bounds

A.1 General lower bound on the number of preemptions

For a fixed algorithm, let $\mathsf{PM}(x)$ denote the maximum number of preemptions that the algorithm makes on a job of size x. The following theorem was originally stated and proven by Motwani et al. [44]. We give a more detailed proof for completeness and clarity.

Theorem A.1 ([44])**.** *Consider a non-clairvoyant algorithm that preempts a job of size x at most $\mathsf{PM}(x)$ times. If $\mathsf{PM}(x) = o(\log x)$, then for every $n \geq 2$ there exists an instance with n jobs on which the algorithm has a competitive ratio of at least $n/2$.*

Proof. Let $B := 2n^2$ and $m = B^{2n}$. Since $\log_m x = \log x / \log m$, there exists a sufficiently large x_0 such that for every $x \geq x_0$ we have $\mathsf{PM}(x) \leq \log_m x$, and additionally we can assume that $x_0 = B^{K+1}$ for some integer $K \geq 2$.

Consider an instance on n jobs, each of size x_0, and the algorithm's schedule at time $t = x_0$. Note that before this time, the algorithm cannot complete any job. By our assumption, the total number of preemptions until time t can be at most

$$n \cdot \log_m x_0 = n \cdot \log_{B^{2n}} B^{K+1} = n \cdot \frac{K+1}{2n} = \frac{K+1}{2} < K\,,$$

because $K \geq 2$.

Now, partition the time interval $[1, B^K] \subseteq [0, x_0]$ into K intervals $I_i = [B^{i-1}, B^i)$ for $i = 1, \ldots, K$. Since there are K intervals before time t but less than K preemptions, by the pigeonhole principle, there must be at least one interval I_i that contains no preemption. Fix such an interval I_i and let j be the job that is processed during I_i. Note that during I_i, job j receives a total processing of at least $B^i - B^{i-1} = (B-1) \cdot B^{i-1} = (2n^2 - 1) \cdot B^{i-1}$. Let $e_{j'}(B^i)$ denote the amount of processing that

job j' received until time B^i. Note that $e_{j'}(B^i) \geq (2n^2-1) \cdot B^{i-1}$, and for every job $j' \neq j$ we have $e_{j'}(B^i) \leq B^{i-1}$.

We next define a new instance with the same set of jobs, but with processing times

- $p_j = e_j(B^i)$ and
- $p_{j'} = e_{j'}(B^i) + \varepsilon$ for all $j' \neq j$, where $\varepsilon = \frac{B^{i-1}}{2n^2}$.

By construction and the fact that the algorithm is deterministic, observe that no job completes before time B^i. Moreover, job j completes exactly at time B^i.

We compute a lower bound on the competitive ratio of the algorithm on this instance. Note that $C_j \geq C_{j'}$ for all $j' \neq j$. Hence,

$$\mathsf{alg} \geq nC_j \geq n \cdot p_j \ .$$

We next give an upper bound on the optimal objective value. Note that for every job $j' \neq j$, we have

$$n^2 p_{j'} \leq n^2 \left(B^{i-1} + \frac{B^{i-1}}{2n^2} \right) = n^2 B^{i-1} + \frac{B^{i-1}}{2} \leq (2n^2-1) \cdot B^{i-1} \leq p_j \ .$$

Let $p = p_j/n^2$. Scheduling job j last achieves a total completion time of at most

$$\frac{(n-1)n}{2} p + (n-1)p + p_j \leq n^2 p + p_j \leq 2p_j \ .$$

Thus,

$$\frac{\mathsf{alg}}{\mathsf{opt}} \geq \frac{n \cdot p_j}{2p_j} = \frac{n}{2} \ ,$$

which completes the proof. □

Theorem A.2. *If for each job j it holds that $\hat{p}_j \leq p_j \leq \lambda_j \hat{p}_j$ and there exists a job j such that the algorithm preempts j at most $o(\log \lambda_j)$, then the algorithm has a competitive ratio of at least $\Omega(n)$ for sufficiently large λ and n.*

Proof. Follows directly from the proof of Theorem A.1 by setting $\hat{p}_j = 1$ for all jobs j. This makes the jobs indistinguishable and the proof strategy applicable. Moreover, $\lambda_j = p_j/\hat{p}_j = p_j$ for all jobs j, hence the guarantees translate. □

A.2 Trade-off lower bound

Theorem A.3. *If a deterministic learning-augmented algorithm uses at most s preemptions in total for a correctly predicted instance with n jobs, then there exists an instance where every job is overestimated and the algorithm has a competitive ratio of at least $\Omega((n-s)/s)$.*

Proof. Let $\hat{p}_j = p_j = P$ for all jobs $j \in J$ be the correctly predicted instance with n jobs. Consider the schedule of the fixed deterministic algorithm on this instance. Until time P, the algorithm cannot have completed any job. Assume that until time P, the algorithm does at most s preemptions, hence the algorithm can touch at most $s+1$ jobs until time P. Let Z denote the set of jobs that were not touched by the algorithm until time P, and note that $|Z| \geq n - (s+1)$. For each job $j \in J \setminus Z$, denote by q_j the amount of work done by the algorithm on j until time P.

We now construct another instance as follows. We set $\hat{p}_j = P$ for all $j \in J$ and

$$p_j = \begin{cases} 1 & \text{if } j \in Z \\ q_j + \varepsilon & \text{if } j \in J \setminus Z \end{cases}$$

Since the algorithm is deterministic, it behaves the same on this new instance as on the original instance, because with the same schedule as before it again will not complete any job until time P. Thus, $\mathsf{alg} \geq P \cdot |Z|$ for this instance. However, an optimal solution can schedule all short jobs in Z first, and then schedule the long jobs in $J \setminus Z$ in any order. This gives $\mathsf{opt} \leq \frac{|Z|(|Z|+1)}{2} + (n - |Z|)(|Z| + P + n\varepsilon)$. By choosing $P \geq n^3$, we conclude that

$$\frac{\mathsf{alg}}{\mathsf{opt}} \geq \frac{P \cdot |Z|}{\frac{|Z|(|Z|+1)}{2} + (n-|Z|)(|Z|+P+n\varepsilon)} \geq \frac{P \cdot (n-(s+1))}{n^2 + (s+1)(n+P+n\varepsilon)} \geq \Omega\left(\frac{n-s}{s}\right) \ .$$

This completes the proof of the theorem. □

B Single machine

B.1 Job arrival over time

We next show how PMLF and its analysis in the proof of Theorem 2.1 can be generalized to the setting with online arrival of jobs. In this setting, a job j arrives at its *release time* $r_j \geq 0$, and an online algorithm does not know about j's existence before that time. PMLF can be extended in a straightforward way to this dynamic scenario:

- Maintain a set of FIFO queues $\{Q_0, Q_1, \ldots\}$.
- Whenever a job j with prediction $\hat{p}_j$ arrives, we push it into queue Q_k where $(1+\delta)^k \leq \hat{p}_j \leq (1+\delta)^{k+1}$.
- At any time t, work on the front job of the non-empty queue of the lowest index.
- If a job $j \in Q_i$ has received a total processing equal to $(1+\delta)^{i+1}$, move it from Q_i to the end of Q_{i+1}.

We prove the following theorem.

Theorem B.1. *For any constant $\delta > 0$, PMLF is $(3+2\delta)$-competitive for minimizing the total completion time on a single machine with online job arrival and performs at most $O(\frac{1}{\delta}\log_2(p_j/\hat{p}_j))$ preemptions for each job j if there are only underpredictions.*

Proof. The proof for the bound on the number of preemption is the same as for Theorem 2.1.

We next prove an upper bound on the competitive ratio, for which we use a classic delay decomposition [44]. Fix a job j with $(1+\delta)^k \leq p_j < (1+\delta)^{k+1}$ and let C_j denote its completion time. Note that at the completion time C_j, it belongs to queue Q_k. Let $d(i,j)$ denote the total delay that job i causes to job j after j has been released, which is the total amount of work done on i between times r_j and C_j. Clearly, $d(i,j) \leq p_i$. Moreover, since j completes at time C_j while belonging to queue Q_k, the algorithm can only work on i during $[r_j, C_j]$ whenever i belongs to a queue $Q_{k'}$ with $k' \leq k$. This implies $d(i,j) < (1+\delta)^{k+1}$. Thus, $d(i,j) \leq \min\{p_i, (1+\delta)^{k+1}\}$, which is at most $(1+\delta)\min\{p_i, p_j\}$ because $p_j \geq (1+\delta)^k$. Since our algorithm does not idle unnecessarily, we have

$$C_j \leq r_j + \sum_{i=1}^{n} d(i,j) \leq r_j + (1+\delta)\sum_{i=1}^{n} \min\{p_i, p_j\} \,.$$

Summing over all jobs j gives

$$\sum_{j=1}^{n} C_j \leq \sum_{j=1}^{n} r_j + (1+\delta)\sum_{j=1}^{n}\sum_{i=1}^{n} \min\{p_i, p_j\} \leq \mathsf{opt} + (1+\delta)\cdot 2\mathsf{opt} \,,$$

where the second inequality uses

- that scheduling jobs in non-decreasing order of their processing times is optimal [52], thus, $\mathsf{opt} = \sum_j p_j + \sum_{j=1}^{n}\sum_{i>j}^{n} \min\{p_i, p_j\}$, and
- $\sum_{j=1}^{n} r_j \leq \mathsf{opt}$ is a trivial lower bound.

This completes the proof of the theorem. □

B.2 Weakly enforcing underestimations

In this section, we prove Theorem 2.2, which we first restate for convenience.

Theorem 2.2. *If there are at most g overestimated jobs, then for every $\varepsilon > 0$, the adapted PMLF is $2(1+\varepsilon)$-competitive and incurs at most $O(\frac{1}{\varepsilon}\sum_j(1+\log_2\frac{p_j}{\hat{p}_j}) + \frac{g}{\varepsilon^2}\log_2(\max_j p_j))$ preemptions.*

Proof. Let J^u and J^o be the set of overestimated and underestimated jobs. Note that $|J^o| \leq g$. Let $n^u(t)$ be the number of underestimated unfinished jobs at time t, and $n(t)$ be the total number of

unfinished jobs at time t. Thus, $n(t) \geq g/\gamma \geq (n(t) - n^u(t))/\gamma$ for every time $t \leq T$. Now we can compute

$$\begin{aligned}\mathsf{alg} = \int_0^\infty n(t)\,dt &= \int_0^T n^u(t)\,dt + \int_0^T (n(t) - n^u(t))\,dt + \int_T^\infty n(t)\,dt \\ &\leq \int_0^T n^u(t)\,dt + \int_0^T \gamma n(t)\,dt + \int_T^\infty n(t)\,dt \\ &\leq \int_0^T n^u(t)\,dt + \gamma\mathsf{alg} + \int_T^\infty n(t)\,dt\,.\end{aligned}$$

Hence,

$$\begin{aligned}(1-\gamma)\mathsf{alg} &\leq \int_0^T n^u(t)\,dt + \int_T^\infty n(t)\,dt \\ &\leq \sum_{j\in J^u} C_j + \sum_{j\in J^o}(C_j - T)\,, \end{aligned} \tag{1}$$

where the last inequality uses the fact that each job $j \in J^u$ is counted in both $n^u(t)$ and $n(t)$, and each job $j \in J^o$ is only counted in $n(t)$ and not in $n^u(t)$. We can now consider both sums individually.

For every underestimated job $j \in J^u$, we can use the same proof as in the proof of Theorem 2.1. Before j's completion, it can only be delayed by jobs i that are in queues of index at most $\lfloor \log_{1+\delta} p_j \rfloor$, hence i delays j by at most $\min\{p_i, (1+\delta)p_j\}$ many times. Thus,

$$C_j \leq \sum_{i\in J} \min\{p_i, (1+\delta)p_j\}\,.$$

For every overestimated job $j \in J^o$, we can also use the same argumentation but only for delays that happen after time T. This gives

$$C_j - T \leq \sum_{i\in J} \min\{p_i, (1+\delta)p_j\}\,.$$

Combining both bounds implies that (1) is at most

$$(1+\delta)\sum_{j\in J}\sum_{i\in J} \min\{p_i, p_j\} \leq (1+\delta)2\cdot\mathsf{opt}\,.$$

Thus, $\mathsf{alg} \leq \frac{1+\delta}{1-\gamma}\cdot 2\cdot\mathsf{opt} = 2(1+\varepsilon)\cdot\mathsf{opt}$ by choosing $\delta = \frac{\varepsilon}{2}$ and $\gamma = \frac{\varepsilon}{2(1+\varepsilon)}$.

We next bound the number of preemptions. For underpredicted jobs the same bound as in Theorem 2.1 applies for all preemptions before time T. At time T, each unfinished job is reinserted at most once, and after time T, each job is preempted at most $O(\log_{1+\delta} p_j)$. Hence, the total number of preemptions is at most

$$O\left(\frac{1}{\delta}\sum_{j\in J}\log\frac{p_j}{\hat{p}_j} + \frac{g}{\gamma\delta}\log p_{\max}\right),$$

which implies the stated bound for our choice of δ and γ. □

C Scheduling on identical parallel machines

In this section, we show that our single machine algorithm PMLF can be extended to scheduling on m parallel identical machines. We first restate the generalization of PMLF for identical parallel machines. Let $\delta > 0$ be a constant.

- Maintain a set of queues $\{Q_0, Q_1, \ldots\}$. Initially, place a job j with $(1+\delta)^k \leq \hat{p}_j < (1+\delta)^{k+1}$ into Q_k.
- At any time t, for each available job j, let (k, a) be the tuple where k is the index of j's current queue, and a is its position in that queue, where the head of the queue has position 1. Let $\preceq$ denote the total over all available jobs that corresponds to the lexicographic order of those tuples. Schedule the first at most m jobs of $\preceq$.

- If a job $j \in Q_i$ has received a total processing equal to $(1+\delta)^{i+1}$ at time t, remove it from Q_i and place it at the end of Q_{i+1}.

We show the following theorem, which matches the guarantees of our single machine result given in Theorem 2.1.

Theorem C.1. *For the setting of only underestimated predictions, PMLF is $(2+2\delta)$-competitive for minimizing the total completion time on parallel identical machines and performs at most $O(\frac{1}{\delta}\sum_j \log(p_j/\hat{p}_j))$ preemptions.*

We split the proof of this theorem into two lemmas. We first bound the number of preemptions, which is equivalent to the argument in the proof of Theorem 2.1.

Lemma C.2. *For the setting of only underestimated predictions, PMLF uses at most $O(\frac{1}{\delta}\sum_j \log(p_j/\hat{p}_j))$ preemptions on m parallel identical machines.*

Proof. We first argue about the number of preemptions, which is equivalent to the argument in the proof of Theorem 2.1. Note that j is only preempted when it is moved between queues. Since job j starts in $Q_{k'}$ where $(1+\delta)^{k'} \le \hat{p}_j < (1+\delta)^{k'+1}$ and completes in Q_k where $(1+\delta)^k \le p_j < (1+\delta)^{k+1}$, it is moved at most $k - k' + 1 = O(\log_{1+\delta}(p_j/\hat{p}_j))$ often. □

For proving the bound on the competitive ratio, we require the following mixed lower bound.

Lemma C.3 (Lemma 9 in [7])**.** *For the problem of minimizing the total completion time of n jobs on m identical parallel machines, given a partition $p_j = p_j^{(1)} + p_j^{(2)}$ for each job j, it holds that*

$$\mathsf{opt} \ge \frac{1}{2m}\sum_{j=1}^{n}\sum_{i=1}^{n}\min\{p_i^{(1)}, p_j^{(1)}\} + \sum_{j=1}^{n} p_j^{(2)} .$$

We now prove that the algorithm is $(2+2\delta)$-competitive.

Lemma C.4. *For the setting of only underestimated predictions, PMLF is $(2+2\delta)$-competitive for minimizing the total completion time on parallel identical machines.*

Proof. Note that since there are no release dates, the number of available jobs cannot increase over time. Let τ be the first time in the schedule when there are less than m jobs available. Note that after time τ, there is no time where at least m jobs are available, and before time τ, there is no time when a machine is idling.

Fix a job j. Let $p_j^{(1)}$ and $p_j^{(2)}$ be the amounts of processing that j receives until time τ and after time τ, respectively. ($p_j^{(2)} = 0$ if j completes before time τ.) Note that $p_j = p_j^{(1)} + p_j^{(2)}$.

Let k_j be the highest queue index that j reaches in the schedule before time τ. Note that for a total time of $\min\{C_j, \tau\}$, the algorithm is busy processing jobs in $Q_0, \dots, Q_{k_j}$ on exactly m machines. Since each job i can be processed by at most $\min\{p_i^{(1)}, (1+\delta)^{k_j+1}\}$ amount of time in these queues before time τ, we have

$$m\cdot\min\{C_j,\tau\} \le \sum_{i=1}^{n}\min\{p_i^{(1)}, (1+\delta)^{k_j+1}\} \le \sum_{i=1}^{n}\min\{p_i^{(1)}, (1+\delta)p_j^{(1)}\} \le (1+\delta)\sum_{i=1}^{n}\min\{p_i^{(1)}, p_j^{(1)}\} .$$

Since $C_j \le \min\{C_j,\tau\} + p_j^{(2)}$, we have

$$\begin{aligned}\mathsf{alg} \le \sum_j C_j &\le \frac{1+\delta}{m}\sum_{j=1}^{n}\sum_{i=1}^{n}\min\{p_i^{(1)}, p_j^{(1)}\} + \sum_{j=1}^{n} p_j^{(2)} \\ &\le 2(1+\delta)\left(\frac{1}{2m}\sum_{j=1}^{n}\sum_{i=1}^{n}\min\{p_i^{(1)}, p_j^{(1)}\} + \sum_{j=1}^{n} p_j^{(2)}\right) \le 2(1+\delta)\mathsf{opt} ,\end{aligned}$$

where the final inequality is due to Lemma C.3. □

D Scheduling on unrelated machines

D.1 Deferred proofs from Section 3

Claim 3.7. *For each epoch k and any $\eta \in (0,1)$, we have,*

$$\mathsf{opt}(l_k y_j^{(k)} \mid j \in J_k) \leq \frac{1}{\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k) + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)\,.$$

Proof. Consider schedules S_J^*, S_X^*, and S_Y^* that correspond to $\mathsf{opt}(l_k y_j^{(k)} \mid j \in J_k)$, $\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k)$, and $\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)$, respectively. Further, let $C_{j,X}$ and $C_{j,Y}$ be job j's completion times in S_X^* and S_Y^*, respectively.

To establish the inequality, it suffices to construct a schedule where each job $j \in J_k$ with size $l_k y_j^{(k)}$ completes by time $C_{j,X}/\eta$ if $j \in X_k$ and by time $C_{j,Y}/(1-\eta)$ if $j \in Y_k$. Towards this end, we "slow down" S_X^* by a factor of $\eta \in (0,1)$ and S_Y^* by a factor of $1-\eta$. Formally, let $z_{j,t,X}$ and $z_{j,t,Y}$ be job j's processing rates at time t in S_X^* and S_Y^*, respectively. Then, j's processing rate at time t in the new schedule is defined as:

$$\eta z_{j,\eta t,X} + (1-\eta) z_{j,(1-\eta)t,Y}.$$

It is straightforward to check that this is a valid schedule since the set of feasible processing rates forms a polytope (and is therefore convex), and $z_{j,\eta t,X}$ and $z_{j,(1-\eta)t,Y}$ are both feasible. Further, we have $\int_{t=0}^{C_{j,X}/\eta} \eta z_{j,\eta t,X}\,dt = \int_{\tau=0}^{C_{j,X}} z_{j,\tau,X}\,d\tau$. This means that by time $C_{j,X}/\eta$, j has been processed in the new schedule as much as it has been by time $C_{j,X}$ in schedule S_X^*. Therefore, the new schedule completes job $j \in X_k$ by time $C_{j,X}/\eta$ as desired. We can similarly show that $j \in Y_k$ completes by time $C_{j,Y}/(1-\eta)$. This completes the proof. □

Lemma 3.8. *For each epoch k, we have $n_k l_k \leq \frac{2\eta}{(\eta-2\beta)(1-\eta)} \cdot \mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)$.*

Proof. Since PF can complete each job of size $l_k y_j^{(k)}$ in $X_k \subseteq J_k$ within l_k time steps, we have

$$\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k) \leq l_k |X_k|\,. \tag{2}$$

Combining Claims 3.6 and 3.7 with $|X_k| \leq \beta n_k$, we obtain

$$\begin{aligned}
&n_k l_k \\
&\leq 2\left(\frac{1}{\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k) + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)\right) \\
&\leq 2\left(\frac{1}{\eta} l_k |X_k| + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)\right) \\
&= 2\left(\frac{\beta}{\eta} l_k n_k + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k)\right)\,.
\end{aligned}$$

By rearraging terms, we obtain the desired inequality. □

D.2 Job arrival over time: sketch

We now discuss how to extend our algorithm and analysis for unrelated machines to accommodate jobs arriving over time. We note that we do not optimize the parameters introduced here, specifically ν. For convenience we will set $\nu = 2\beta$. Furthermore, while we detail the necessary algorithmic changes and address the subtle points of the analysis, we provide only a compact proof sketch. As discussed in the introduction, this follows our primary focus, which remains on settings where jobs do not arrive over time.

Algorithmically, since our approach simulates the PF schedule within each epoch using a non-preemptive schedule, we should refine epochs whenever a significant number of new jobs arrives. Specifically, we initiate a sub-epoch if the number of new arrivals reaches a ν-fraction of the currently

active jobs: if $J(e_k)$ denotes the set of jobs present at the beginning of epoch k, the first sub-epoch (if it exists) begins at time $e_{k,1}$ when $\nu|J(e_k)|$ new jobs have arrived while the non-preemptive PF-simulated schedule (Step 4) is being executed. In general, if $J(e_{k,h})$ are the jobs alive at the start of the hth sub-epoch, then the $(h+1)$th sub-epoch begins at time $e_{k,h+1}$ when $\nu|J(e_{k,h})|$ new jobs have arrived while the non-preemptive PF-simulated schedule (Step 4) of the hth sub-epoch is being executed.

We note that from an algorithmic standpoint, epochs and sub-epochs are identical; this distinction is maintained primarily for the purpose of analysis.

Bounding preemptions. We first discuss how we can upper bound the number of preemptions. If an epoch k has no sub-epochs, then the analysis proceeds as before as the newly arriving jobs are not preempted. So, suppose it has sub-epochs, $1, 2, \ldots, h, \ldots, H$. The additional preemptions initiated at the beginning of each sub-epoch, that is at time $e_{k,h}$, can be charged to the new jobs arriving during $(e_{k,h-1}, e_{k,h}]$. Here, for convenience, $e_{k,0} := e_k$. Note that because each arrival is responsible for at most $1/\nu$ jobs in this charging scheme and is charged only once during its lifetime, the scheme is valid.

Bounding total completion time. We now focus on bounding the total completion time objective. To this end, consider a fixed epoch k. First suppose it contains no sub-epochs. Recall that the cost of each epoch is bounded by $4n_k l_k$, where $n_k := |J(e_k)|$ and $4l_k$ represents an upper bound on the epoch's length. Since the additional arrivals (up to νn_k jobs) cannot be processed until time e_k, the optimal solution must also wait, just as our algorithm does, thereby incurring a comparable cost. During the epoch, the additional cost incurred is at most $4\nu n_k l_k$. Consequently, this increases the objective by a factor of at most $1 + \nu$.

We now consider the more interesting case where the fixed epoch k contains sub-epochs. For easy notation, let $t_h := e_{k,h}$ and $t_0 := e_k$. Similarly, $J_0 := J(e_k)$, and $J_h := J(e_{k,h})$. Also, for simplicity, we call the time interval $[t_0, t_1)$ as the 0th epoch. Let t_{H+1} denote the last epoch's end time.

Since we set $\nu = 2\beta$, less than $\nu/2$ fraction of jobs reach their checkpoints, meaning that at most $\nu/2$ fraction of jobs can be completed; yet ν fraction of jobs newly arrive. Thus, it follows that $|J_{h+1}| \geq (1 + \nu/2)|J_h|$ for all $h = 0, 1, \ldots, H-1$. We charge the cost of jobs that were alive at any point during $[e_0, e_{H-2})$, denoted as the set V_1, up until time t_{H-1} to the arrival times of the jobs arriving in the penultimate epoch, $H-1$, which we denote as V_2. To see this, observe that $|V_2|$ is at least $\nu|J_{H-1}|$ and they all arrive no earlier than t_{H-1}. Furthermore, the cost of V_1 until t_{H-1} is at most $t_{H-1}(1+\nu)|J_{H-1}| \leq t_{H-1}(1+\nu)|V_2|(\frac{1}{\nu})$. So, we will have a competitive ratio of $\frac{1}{\nu}(1+\nu)$ in this charging.

Before bounding the waiting time for jobs in the final two sub-epochs, we discuss a subtlety in the analysis. Suppose the majority of jobs in V_2 arrive shortly after t_{H-1}. In this case, these arrivals might not have a sufficiently large "arrival time window" to offset the cost incurred by jobs in J_{H-1} up to t_H. (The cost of J_H through t_{H+1} can be easily handled, similar to the case with no sub-epochs or only the 0th epoch). However, in sub-epoch $H-1$, we simulated the PF schedule based on the jobs in J_{H-1} while ignoring V_2. If we had to preempt sub-epoch $H-1$ after simulating very few jobs up to what PF would have done, our non-preemptive simulation of PF in that interval may not be faithful to the actual PF schedule, meaning that the simulation time was effectively "wasted."

The remaining goal is to upper bound the waiting cost W of J_{H-1} accumulating during the interval $[t_{H-1}, t_H)$ (including V_2, the waiting cost will be a factor of $1+\nu$ larger). To upper bound W, we consider several cases; in some cases, we will charge W to specific subsequent epochs.

1. The last sub-epoch H of epoch k completes at least $(\nu/4)|J_H|$ jobs. We charge W to the total completion time of these jobs, which increases the competitive ratio by a factor of $\Theta(1/\nu)$. This charging is done "locally" within the same epoch, ensuring no overcharging.
2. Otherwise, let $k' > k$ be the latest epoch such that at least $(\nu/4)|J_H|$ jobs are completed during epochs $k+1, k+2, \ldots, k'$.
   (a) If any of the epochs $k+1, k+2, \ldots, k'-1$ has a sub-epoch—say k'' is the earliest epoch having a sub-epoch—then there must exist at least $\nu(1-\nu/2)|J_H|$ jobs that arrive during epoch k''. Each of these jobs has an arrival time greater than the waiting

time of each job accounting for W during sub-epoch $H-1$ of epoch k. Thus, we can charge W to the total arrival time of these jobs, which increases the competitive ratio by at most $O(1/\nu)$ additively. Note that this cross-epoch charging occurs to them only once, ensuring no overcharging.

(b) Otherwise, we charge W to the total completion time of jobs completed during epochs $k+1,\dots,k'$. Note that the waiting cost W occurs during epoch k (which has a sub-epoch) and it is charged to some of consecutive subsequent epochs that have no sub-epochs. Therefore, overcharging does not occur.

This completes the sketch of the analysis.

D.3 When predictions include overestimation

In this section, we show how to extend our results to cases where predictions include overestimation, i.e., $\hat{p}_j > p_j$ for some jobs j. We consider two strategies to handle this.

D.3.1 Strictly enforcing underestimation

The first approach is to ensure that no overestimation is effectively perceived by the algorithm. Specifically, if a job completes after receiving less than $\hat{p}_j$ units of processing, we treat the job as remaining active until it has received exactly $\hat{p}_j$ units. This is equivalent to processing jobs where each job j has an effective size of $\max\{p_j, \hat{p}_j\}$ and a predicted size of $\hat{p}_j$. Under this strategy, the bound in Theorem 3.3 becomes:

$$O\left(\frac{1}{\beta\delta}\sum_j \log\left(\frac{\max\{p_j,\hat{p}_j\}}{\hat{p}_j}+1\right)\right) = O\left(\frac{1}{\beta\delta}\sum_j \log\left(\frac{p_j}{\hat{p}_j}+1\right)\right).$$

Thus, the asymptotic bound for preemptions remains unchanged. For the total completion time objective, this modification increases each job's effective size by a factor of at most $u := \max_j \hat{p}_j/p_j$. Since increasing every job size by a factor of u increases the optimal objective by at most the same factor, the competitive ratio becomes $14.48(1+\varepsilon)u$.

D.3.2 Weakly enforcing underestimation

Since overprediction has a significantly negative effect on the competitive ratio, we can intentionally underpredict job sizes at the cost of additional preemptions. If the number of overpredicted jobs is small and we have an upper bound g on this quantity with high probability (w.h.p.), we can handle these cases more effectively. Intuitively, predicting a single global parameter—the total number of overpredicted jobs—is easier than identifying specifically which jobs are overpredicted.

Suppose we know that there are at most g *jobs with overpredicted sizes*. We run the SNAP algorithm as before until the number of alive jobs drops to g/ε or less. More specifically, at time e_k, if $n_k \le g/\varepsilon$, we reset the predicted size of every alive job to the smallest power of $1+\delta$ that is no smaller than its current cumulative processed size. This effectively means that from this point forward, we ignore the original predictions to ensure we only deal with *underpredictions*. Let this transition occur at the beginning of epoch $K'+1$. Another small caveat in running SNAP until epoch K' is if a job turns out to be overestimated. So, a job may complete before it receives $v_{j,k}$ units of processing in epoch k but this can only reduce the current epoch's length, which is only good for the algorithm.

Competitive ratio analysis. The analysis then proceeds similarly. The primary difference is that until epoch K', we focus only on the underestimated jobs. That is, n_k^u denotes the number of *underestimated* jobs alive at e_k. Lemma 3.5 is modified as follows:

Lemma D.1 (Modification of Lemma 3.5)**.** $\mathsf{alg} \le \sum_{k=1}^{K'}(n_k^u+g)4l_k + \sum_{k=K'+1}^{K} n_k^u 4l_k \le 4(1+\varepsilon)\sum_{k=1}^{K} n_k^u l_k$.

Here, note that since epoch $K'+1$, all jobs are effectively underestimated (in the analysis of the completion time objective only), thus, $n_k = n_k^u$ for all $k \ge K'+1$.

The subsequent claims, lemmas, and corollaries need only very minor changes. Let J_k^u be the subset of jobs in J_k that are underestimated. We analogously restrict X_k and Y_k to X_k^u and Y_k^u, respectively.

Claim D.2 (Modification of Claim 3.6)**.** *For each epoch k, we have $n_k^u l_k \le 2 \cdot \mathsf{opt}(l_k y_j^{(k)} \mid j \in J_k^u)$.*

Claim D.3 (Modification of Claim 3.7)**.** *For each epoch k and any $\eta \in (0,1)$, we have,*

$$\mathsf{opt}(l_k y_j^{(k)} \mid j \in J_k^u) \le \frac{1}{\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k^u) + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k^u)\,. \tag{3}$$

Lemma D.4 (Modification of Lemma 3.8)**.** *For each epoch k, we have $n_k^u l_k \le \left(\frac{2\eta}{(\eta - 2\beta(1+2\varepsilon))(1-\eta)}\right)\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k^u)$.*

Proof. We first note that $n_k \le n_k^u/(1-\varepsilon)$: After the transition, all jobs are effectively considered, and therefore, this holds true obviously. Before the transition, we have $n_k \ge g/\varepsilon$. Since $n_k \le n_k^u + g \le n_k^u + \varepsilon n_k$, we have the claimed inequality.

Combining Claims D.2 and D.3 with the fact that $|X_k| \le \beta n_k \le \beta n_k^u/(1-\varepsilon) \le \beta(1+2\varepsilon)n_k^u$, we obtain

$$\begin{aligned} n_k^u l_k &\le 2\left(\frac{1}{\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in X_k^u) + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k^u)\right) \\ &\le 2\left(\frac{1}{\eta} l_k |X_k^u| + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k^u)\right) \\ &= 2\left(\frac{\beta(1+2\varepsilon)}{\eta} l_k n_k^u + \frac{1}{1-\eta}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k^u)\right)\,. \end{aligned}$$

By rearranging terms, we obtain the desired inequality. □

Lemma D.5 (Modification of Lemma 3.9)**.** *For every job j, it holds that $\sum_{k: j \in Y_k^u} l_k y_j^{(k)} \le p_j$.*

Corollary D.6 (Modification of Corollary 3.11)**.** $\sum_{k=1}^{K} \mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k) \le 1.81 \cdot \mathsf{opt}(p)$.

Finally, we upper bound the total completion time objective of this two-stage modification of SNAP. Specifically, we have,

$$\begin{aligned} \mathsf{alg}(p) &\le 4(1+\varepsilon)\sum_{k=1}^{K} n_k l_k \\ &\le 4(1+\varepsilon)\left(\frac{2\eta}{(\eta - 2\beta(1+2\varepsilon))(1-\eta)}\right)\sum_{k=1}^{K}\mathsf{opt}(l_k y_j^{(k)} \mid j \in Y_k) \\ &\le 14.48(1+\varepsilon)\left(\frac{\eta}{(\eta - 2\beta(1+2\varepsilon))(1-\eta)}\right)\mathsf{opt}(p) \\ &\le 14.48(1+\varepsilon)^2\mathsf{opt}(p)\,, \end{aligned}$$

where the first inequality is by Lemma D.1, the second inequality is by Lemma D.4, and the third inequality is by Corollary D.6 and by choosing $\eta = \frac{\varepsilon}{2(1+\varepsilon)}$ and $\beta = \frac{\varepsilon^2}{4(1+\varepsilon)(2+\varepsilon)(1+2\varepsilon)}$ for any sufficiently small $\varepsilon > 0$. By scaling ε appropriately, we essentially maintain the *same* competitive ratio as that stated in Theorem 3.4.

Bounding number of preemptions. We now turn our attention to upper-bounding the number of preemptions. By following exactly the same steps as the proof of Theorem 3.3, we can bound the number of preemptions until epoch K' by:

$$\sum_{k \le K'} n_k \le O\left(\frac{1}{\beta\delta}\sum_j \log\left(\frac{p_j}{\hat{p}_j} + 1\right)\right).$$

Next, consider the jobs alive at the transition time $e_{K'+1}$. By the definition of the transition, we know $|J_{K'+1}| \le g \cdot (1 + 1/\varepsilon)$. Since we reset the predicted sizes of all jobs in $J_{K'+1}$ such that they are

all henceforth underestimated, we can apply Theorem 3.3. Letting $p_{\max} := \max_j p_j$, the number of preemptions for these jobs is at most:

$$O\left(\frac{1}{\beta\delta}\sum_{j\in J_{K'+1}}(1+\log p_j)\right) = O\left(\frac{g}{\beta\delta\varepsilon}\log p_{\max}\right) = O\left(\frac{g}{\varepsilon^4}\log p_{\max}\right).$$

To summarize, we have shown the following theorem:

Theorem D.7. *For any sufficiently small $\varepsilon > 0$, assuming there are at most g overpredicted jobs, there exists an algorithm that is $14.48(1+\varepsilon)$-competitive for minimizing total completion time while performing at most $O(\frac{1}{\varepsilon^3}(\sum_j \log(\frac{p_j}{\hat{p}_j}+1) + \frac{g}{\varepsilon}\log p_{\max}))$ preemptions and migrations.*

E A general framework for PSP

In this section, we present the full description of our framework for PSP, and the proof of Theorem 4.4.

Our framework works in epochs $1, 2, \ldots, K$. Epoch k starts at time e_k and ends at time e_{k+1}; Epoch 1 starts at time $e_1 = 0$. Let J_k denote the set of available jobs at the start of epoch k and let $|J_k| = n_k$. At time e_k, we plan a complete schedule for epoch k as described in the following:

1. **Compute PF Rates.** We compute the PF rates $y^{(k)}$ for the jobs J_k. Formally, $y^{(k)}$ is the solution to
$$\begin{aligned}(\mathsf{CP}(J)) \quad \max \quad & \sum_{j\in J} w_j \cdot \log(y_j) \\ \text{s.t.} \quad & \sum_{j\in J} b_{dj}\cdot y_j \quad \leq \quad 1 \qquad \forall d \in [D] \\ & y_j \quad \geq \quad 0 \qquad \forall j \in J\end{aligned}$$
2. **Compute Next Checkpoints.** For each job $j \in J_k$, we define the remaining processing requirement to reach its next checkpoint as $u_{j,k} = (1+\delta)^{h+1} - q_j(e_k)$, where $h \in \mathbb{N}$ s.t. $(1+\delta)^h \leq \max\{q_j(e_k), \hat{p}_j\} < (1+\delta)^{h+1}$.
3. **Determine the Simulation Duration.** Let l_k denote the length of the shortest time interval such that, if jobs are processed at rates $y^{(k)}$ during the time interval, at least $\lceil \beta n_k \rceil$ jobs from J_k reach their next checkpoint. The targeted processing requirement for each job $j \in J_k$ is then $v_{j,k} = \min\{u_{j,k}, l_k y_j^{(k)}\}$.
4. **Convert to a $O(1)$-Preemptive Schedule.** We now compute an α-approximate $O(1)$-preemptive schedule (cf. Definition 4.1) for jobs J_k with processing requirements $v_{j,k}$ for each $j \in J_k$. If we start this schedule at time e_k, it will complete by time at most $e_{k+1} \leq e_k + \alpha l_k$. Then, Epoch k ends and we start Epoch $k+1$ if not all jobs have been completed yet.

Theorem E.1. *The total number of preemptions performed by our framework is at most $O(\frac{1}{\beta\delta}\sum_j(1+\log_2 p_j/\hat{p}_j))$.*

Proof. In each epoch k, the α-approximate $O(1)$-preemptive schedule uses by definition at most $O(1)$ preemptions per job, so $O(n_k)$ preemptions in total. By our choice of l_k, at least βn_k jobs each reach a checkpoint during epoch k. Since there can be at most $\sum_j(1+\lceil\log_{1+\delta} p_j/\hat{p}_j\rceil)$ checkpoints in total, there can be at most $\beta\cdot\sum_j(1+\lceil\log_{1+\delta} p_j/\hat{p}_j\rceil)$ many epochs. Thus, the total number of preemptions used by our algorithm is at most $O(\frac{1}{\beta}\sum_j \log_{1+\delta} p_j/\hat{p}_j) = O(\frac{1}{\beta\delta}\sum_j(1+\log_2 p_j/\hat{p}_j))$. □

Lemma E.2. *For any $\beta \in (0,1)$, the virtual algorithm achieves a total weighted completion time at most $\frac{27(1+\delta)}{(1-\beta)^3}$ times the optimum for PSP.*

Proof. Consider any epoch k in the schedule S of the virtual algorithm for this epoch. For simplicity, we assume without loss of generality that epoch k starts at time 0 and ends at time l_k. Let X_k denote the set of jobs that reach their next checkpoint during epoch k. For each X_k, let $l_{j,k}$ denote the time at which job j reaches its next checkpoint, that is, a processing requirement of $v_{j,k}$.

We next construct an instance I'_k for β-robust PSP. PF's initial rates y for I'_k are equal to $y^{(k)}$, as both are based on the set of jobs J_k and the rates are unique. For any time $t \in [0, l_k]$, we choose the nullified set of jobs to be $N(t) = \{j \in X_k \mid t \leq l_k - l_{j,k}\}$. By the definition of $N(t)$, for each job $j \in J_k \setminus X_k$, we have that $j \notin N(t)$ for all $t \in [0, l_k]$. For each job $j \in X_k$, we have that $j \in N(t)$ for all $t \in [0, l_{j,k}]$. This implies that in the schedule S' of PF for I'_k, every job completes exactly at time l_k. Hence $|N(t)| \leq \beta |J_k| = \beta |J(t)|$ for all $t \in [0, l_k]$, so I'_k is a β-robust PSP instance. In conclusion, the total weighted completion time of PF for I'_k, denoted by $\mathrm{PF}(I'_k)$, is at most $n_k l_k$.

Similarly to Lemma 3.5, we have

$$\mathsf{alg}^{virt} \leq \sum_{k \geq 1} n_k l_k = \sum_{k \geq 1} \mathrm{PF}(I'_k) \,.$$

Now we can concatenate all instances I'_k to form a new instance I'. It is easy to check that this is again a β-robust PSP instance where each job j has a total processing requirement equal to p_j. Moreover, the total weighted completion time of PF for I' is equal to $\sum_{k \geq 0} \mathrm{PF}(I'_k)$. Therefore,

$$\mathsf{alg}^{virt} \leq \sum_{k \geq 1} n_k l_k = \sum_{k \geq 1} \mathrm{PF}(I'_k) = \mathrm{PF}(I') \,.$$

Finally, by Theorem 4.3, we have $\mathrm{PF}(I') \leq 27/(1-\beta)^3 \cdot \mathsf{opt}$. □

Since the total completion time of the framework is at most α times the total completion time of the virtual algorithm, Theorem E.1 and Lemma E.2 imply Theorem 4.4.

F Application to malleable scheduling

We apply our framework to malleable job scheduling on identical and unrelated machines. We are given a set of machines M and for every job $j \in J$ a speed-up function $f_j \colon \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$. In a job-to-machine assignment, let $x_{ij}(t)$ be the fraction of machine i given to job j at time t. The processing rate of job j at time t is given by $f_j(\sum_i s_{ij} x_{ij}(t))$. We assume that each f_j is non-decreasing and concave. This is a PSP with the rate polytope $\mathcal{P} = \{f(x) \mid \sum_{i \in M} x_{ij} \leq 1 \, \forall j \in J, \sum_{j \in J} x_{ij} \leq 1 \forall i \in M\}$; see [24] for details.

We present algorithms for computing α-approximate $O(1)$-preemptive schedules (cf. Definition 4.1), which together with Theorem 4.4 imply competitive algorithms with few preemptions.

We start with the case of m identical parallel machines. Here, we additionally assume that each function f_j is piecewise linear with breakpoints at integers points.

Theorem F.1. *For any $\varepsilon > 0$, there is a $54(1+\varepsilon)$-competitive algorithm that incurs at most $O(\frac{1}{\varepsilon^2} \sum_j (1 + \log_2 p_j / \hat{p}_j))$ preemptions for minimizing the total completion time for malleable job scheduling on identical parallel machines if each function f_j is piecewise linear with breakpoints at integer points.*

Proof. We provide an algorithm for computing 2-approximate 0-preemptive schedules for the problem. The theorem then follows from Theorem 4.4.

Consider fixed rates y that correspond to a fractional schedule. where each job j is assigned to x_j machines and a length L. Let $J_1 = \{j \mid x_j \geq 1\}$ and $J_2 = \{j \mid x_j < 1\}$. We create a non-preemptive integral schedule as follows: For each job $j \in J_1$, we assign job j to $\lfloor x_j \rfloor$ machines M_j and those machines only work on job j throughout the whole schedule. Since f_j is concave and piecewise linear with breakpoints at integers, and since $x_j \geq 1$, we have $f_j(x_j) \leq 2 f_j(\lfloor x_j \rfloor)$. Thus, the completion time of j in the integral schedule is at most $2L$. Note that jobs in J_1 use at most $m_1 = \lfloor \sum_{j \in J_1} x_j \rfloor$ many machines. Next, consider all jobs $j \in J_2$. We greedily schedule those jobs on the remaining $m - m_1 = \lceil \sum_{j \in J_2} x_j \rceil$ machines via list scheduling, where each job gets one machine until it finishes. By the well-known list scheduling properties, this ensures that all jobs in J_2 complete within at most $2L$ time. Since no job is preempted, this completes the proof of the theorem. □

We next present our results for unrelated machines. For this problem we additionally assume that $f_j(s) \cdot s$ is non-increasing as a function of s. This is a common *non-decreasing work* assumption, see e.g., [19].

Theorem 4.5. *For any $\varepsilon > 0$, there is a $27\frac{e}{e-1}(1+\varepsilon)$-competitive algorithm that incurs at most $O(\frac{1}{\varepsilon^3}\sum_j(1+\log_2 p_j/\hat{p}_j))$ preemptions for minimizing the total completion time of malleable jobs on unrelated machines with non-decreasing, concave, and non-decreasing work speed-up functions.*

Proof. We provide an algorithm for computing $e/(e-1)$-approximate 0-preemptive schedules for the problem. The theorem then follows from Theorem 4.4.

Consider fixed rates y that correspond to a fractional schedule, where each job j is fractionally assigned to machine i with x_{ij}. Assume w.l.o.g. by scaling that the length of the fractional schedule is 1. Define

- $\sigma_j(x) := \sum_i \lambda_{ij} x_{ij}$ for every job j,
- $\gamma_j := \min\{q \in \mathbb{R} \mid y_j \leq f_j(q)\}$ for every job j, and
- $r_{ij} := \max\{\lambda_{ij}, \gamma_j\}$ for every machine i and job j.

Consider the new allocation

$$\hat{x}_{ij} = \frac{\lambda_{ij} \cdot x_{ij}}{\sigma_j(x)}$$

for all machines i and jobs j. We will show that the allocation $\hat{x}$ satisfies the following two constraints:

$$\sum_i \hat{x}_{ij} = 1 \quad \forall j$$
$$\sum_j \frac{y_j}{f_j(r_{ij})} \cdot \frac{r_{ij}}{\lambda_{ij}} \cdot \hat{x}_{ij} \leq 1 \quad \forall i$$

Fotakis, Matuschke, and Papadigenopoulos [19] showed that if these two constraints hold, then there exists a non-preemptive schedule with a makespan of at most $e/(e-1)$, which implies our theorem.

The first constraint holds immediately from the definition of $\hat{x}$. To verify the second constraint for any machine i, consider a job j with $x_{ij} > 0$. Note that $\sigma_j(x) \geq x_{ij} s_{ij}$ and $\sigma_j(x) \geq \gamma_j$, because $f_j(\sigma_j(x)) = y_j$. Thus, we have $\sigma_j(x) \geq x_{ij} r_{ij}$. Then, it holds that

$$\begin{aligned}
\frac{y_j}{f_j(r_{ij})} \cdot \frac{r_{ij}}{s_{ij}} \cdot \hat{x}_{ij} = \frac{y_j}{f_j(r_{ij})} \cdot \frac{r_{ij}}{s_{ij}} \cdot \frac{s_{ij} \cdot x_{ij}}{\sigma_j(x)} &= \frac{y_j}{f_j(r_{ij})} \cdot \frac{r_{ij} \cdot x_{ij}}{\sigma_j(x)} \\
&\leq \frac{y_j}{f_j(\sigma_j(x)/x_{ij})} \qquad (4) \\
&\leq x_{ij} \frac{y_j}{f_j(\sigma_j(x))} = x_{ij}\,, \qquad (5)
\end{aligned}$$

where Inequality (4) follows from $\sigma_j(x) \geq x_{ij} r_{ij}$ and Fact 5 in [19], and Inequality (5) follows from the concavity of f_j. Summing over all jobs j gives the second constraint, because $\sum_j x_{ij} \leq 1$. □

G Proportional Fairness for β-robust PSP

In this section, we analyze PF for β-robust PSP and show Theorem 4.3, which we first restate. Our analysis holds even for the setting where jobs arrive online over time at their release dates r_j.

Theorem 4.3. *For any $\beta \in [0,1)$, PF is $27/(1-\beta)^3$-competitive for β-robust PSP.*

We first restate the PF algorithm. At every time t, it computes the processing allocation $y(t)$ that is optimal for the following convex program $\mathsf{CP}(J(t))$ for the active jobs $J(t)$ at time t.

$$\begin{aligned}
(\mathsf{CP}(J)) \quad \max \quad & \sum_{j \in J} w_j \cdot \log(y_j) \\
\text{s.t.} \quad & \sum_{j \in J} b_{dj} \cdot y_j \leq 1 \qquad \forall d \in [D] \\
& y_j \geq 0 \qquad \forall j \in J
\end{aligned}$$

In the β-robust PSP setting, the adversary can then select a set of jobs $N(t) \subseteq J(t)$ such that $|N(t)| \leq \beta \cdot |J(t)|$, for some fixed $\beta \in (0,1)$. The final processing allocation of the algorithm is then

$$\hat{y}_j(t) := \begin{cases} 0 & \text{if } j \in N(t) \\ y_j(t) & \text{else.} \end{cases}$$

Note that if $y(t) \in \mathcal{P}$ then also $\hat{y}(t) \in \mathcal{P}$ since we assume that $\mathcal{P}$ is downward closed.

Our analysis is based on the analysis of PF for PSP [24, 28]. Our main contribution here is to carefully integrate that changed rates of nullified jobs into the traditional analysis. A critical issue is that the new rate vector $\hat{y}$ is not necessarily an optimal solution to $\mathsf{CP}(J)$ and thus does not satisfy crucial optimality conditions on which the proof in [24, 28] relies on.

G.1 KKT conditions

To characterize the optimal solution of $(\mathsf{CP}(J))$, we consider its KKT conditions for the Lagrange multipliers η_d for each $d \in [D]$ [8].

$$\frac{1}{y_j} - \sum_{d=1}^{D} b_{dj}\eta_d = 0 \quad \forall j \in J \tag{6}$$

$$\eta_d \cdot \left(1 - \sum_{j \in J} b_{dj} y_j\right) = 0 \quad \forall d \in [D] \tag{7}$$

$$\eta_d \geq 0 \quad \forall d \in [D] \tag{8}$$

A direct consequence of the KKT conditions is the following lemma, which is a classic connection between the number of active jobs and their dual multipliers [17, 24]. We refer to [24, 28] for a proof.

Lemma G.1 ([24]). *Let J be a set of jobs, and let η be optimal Lagrange multipliers for $(\mathsf{CP}(J))$. Then $\sum_{d=1}^{D} \eta_d = |J|$.*

G.2 Linear programming relaxation

We compare the performance of PF to an optimal schedule that is allowed to run at a slower speed. More precisely, we compare PF to an optimal solution that runs at speed $1/\kappa$ for some parameter $\kappa \geq 1$. The following linear program is a relaxation of this optimal solution, where the variables y_{jt} indicate the total amount of processing that job j receives in the interval $[t, t+1]$.

$$\begin{aligned} (\mathsf{LP}(\kappa)) \quad \min \quad & \sum_{j \in J} \sum_{t \geq r_j} \frac{y_{jt}}{p_j}\left(t + \frac{1}{2}\right) & & \\ \text{s.t.} \quad & \sum_{t \geq r_j} y_{jt} \geq p_j & & \forall j \in J \\ & \sum_{j \in J} b_{dj} \cdot y_{jt} \leq \frac{1}{\kappa} & & \forall d \in [D],\ \forall t \geq 0 \\ & y_{jt} \geq 0 & & \forall j \in J,\ \forall t \geq r_j \end{aligned}$$

Its dual can be written as follows.

$$\begin{aligned} (\mathsf{DP}(\kappa)) \quad \max \quad & \sum_{j \in J} a_j - \sum_{d=1}^{D} \sum_{t \geq 0} b_{dt} & & \\ \text{s.t.} \quad & \frac{a_j}{p_j} - \frac{1}{p_j}\left(t + \frac{1}{2}\right) \leq \kappa \sum_{d=1}^{D} b_{dj} b_{dt} & & \forall j \in J,\ \forall t \geq r_j \\ & a_j, b_{dt} \geq 0 & & \forall j \in J,\ \forall t \geq 0,\ \forall d \in [D] \end{aligned}$$

G.3 Dual fitting

Fix an instance and consider the schedule produced by PF for that instance. Let C_j denote the completion time of job j in that schedule, and let $\mathsf{alg} := \sum_{j=1}^{n} C_j$ denote the objective function value of PF. For every time t, let $U(t)$ denote the set of unfinished (and potentially unreleased) jobs at time t. Let λ be a constant with $\beta < \lambda < 1$ that we fix later. In the following, we assume w.l.o.g. by scaling that all weights are integers.

For every time t, let $\zeta(t)$ be the λ-quantile of the values $\hat{y}_j(t)/p_j$, $j \in U(t)$. More formally, if Z_t denotes the sorted (ascending) list of length $|U(t)|$ composed of $\hat{y}_j(t)/p_j$ for every $j \in U(t)$, then $\zeta(t)$ is the $\lceil \lambda |U(t)| \rceil$th value in Z_t. Further, for $t \geq 0$ let $\eta_d(t)$ be the optimal Lagrange multiplier corresponding to the constraint $d \in [D]$ of $(\mathsf{CP}(J(t)))$. We consider the following assignment of dual variables for $(\mathsf{DP}(\kappa))$:

- $a_j := \sum_{t=0}^{C_j} a_{jt}$ for every job $j \in J$, where $a_{jt} := \mathsf{1}\left[\hat{y}_j(t)/p_j \leq \zeta(t)\right] \cdot \mathsf{1}\left[j \in U(t) \setminus N(t)\right]$ for $t \geq 0$ and $j \in U(t)$,
- $b_{dt} := \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \cdot \eta_d(t')$ for every $d \in [D]$ and time $t \geq 0$.

We first show in the following lemma that the objective value of $(\mathsf{DP}(\kappa))$ for this assignment upper bounds a constant fraction of the algorithm's objective value.

Lemma G.2. *It holds that* $\left(\lambda - \beta - \frac{1}{(1-\lambda)\kappa}\right)\mathsf{alg} \leq \sum_{j \in J} a_j - \sum_{d=1}^{D} \sum_{t \geq 0} b_{dt}$.

The lemma follows from the following two statements.

Lemma G.3. *It holds that* $\sum_{j \in J} a_j \geq (\lambda - \beta) \cdot \mathsf{alg}$.

Proof. Consider a time t. Observe that $\sum_{j \in U(t)} a_{jt}$ contains the total number of jobs j that satisfy $\hat{y}_j(t)/p_j \leq \zeta(t)$ and $j \in U(t) \setminus N(t)$. By the definition of $\zeta(t)$, we conclude that $\hat{y}_j(t)/p_j \leq \zeta(t)$ satisfy at least $\lambda \cdot |U(t)|$ jobs. By our assumption on $N(t)$, we conclude that $j \in U(t) \setminus N(t)$ satisfy at least $(1-\beta)|U(t)|$ jobs. Thus, at time t, we have a total contribution of at least $(1 - (1-\lambda) - \beta) \cdot |U(t)| = (\lambda - \beta) \cdot |U(t)|$ to $\sum_{j \in J} a_j$. The statement then follows because $\mathsf{alg} = \sum_{t \geq 0} |U(t)|$. □

The following lemma and its proof appear also in [28]. We include it here for completeness.

Lemma G.4 ([28])**.** *At any time t, it holds that* $\sum_{d=1}^{D} b_{dt} \leq \frac{1}{(1-\lambda)\kappa} |U(t)|$.

Proof. Fix a time t. Observe that for every $t' \geq t$, the definition of $\zeta(t')$ implies that $|U(t')| \cdot \mathsf{1}\left[\hat{y}_j(t')/p_j \geq \zeta(t')\right] \geq (1-\lambda) \cdot |U(t')|$. Thus,

$$\zeta(t') \cdot (1-\lambda) \cdot |U(t')| \leq |U(t')| \cdot \zeta(t') \cdot \mathsf{1}\left[\frac{\hat{y}_j(t')}{p_j} \geq \zeta(t')\right] \leq |U(t')| \cdot \frac{\hat{y}_j(t')}{p_j} \,. \tag{9}$$

The definition of b_{it} and Lemma G.1 imply

$$\begin{aligned} \sum_{d=1}^{D} b_{dt} = \sum_{d=1}^{D} \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \cdot \eta_d(t') = \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \sum_{d=1}^{D} \eta_d(t') &= \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \cdot |J(t')| \\ &\leq \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \cdot |U(t')| \,, \end{aligned}$$

where the final inequality holds because $J(t') \subseteq U(t')$ for all times t'. Using (9), we conclude for the right-hand side of the above inequality that

$$\begin{aligned} \frac{1}{\kappa} \sum_{t' \geq t} \zeta(t') \cdot |U(t')| &\leq \frac{1}{(1-\lambda)\kappa} \sum_{t' \geq t} \sum_{j \in U(t')} \frac{\hat{y}_j(t')}{p_j} \\ &\leq \frac{1}{(1-\lambda)\kappa} \sum_{j \in U(t)} \sum_{t' \geq t} \frac{\hat{y}_j(t')}{p_j} \leq \frac{1}{(1-\lambda)\kappa} \sum_{j \in U(t)} 1 \,. \end{aligned}$$

The second inequality follows from $U(t') \subseteq U(t)$ for $t' \geq t$. The third inequality holds because $\sum_{t' \geq t} \hat{y}_j(t') \leq p_j$ for every job j. This concludes the proof of the lemma. □

Next, we show dual feasibility.

Lemma G.5. *The dual solution* $(a_j)_j$ *and* $(b_{it})_{i,t}$ *is feasible for* $(\mathsf{DP}(\kappa))$.

Proof. First, note that the dual solution is non-negative. To verify the dual constraint, fix a job j and a time $t \geq r_j$. The left-hand side of the dual constraint is then given by

$$\begin{aligned}
\frac{a_j}{p_j} - \frac{1}{p_j}\Big(t + \frac{1}{2}\Big) &\leq \sum_{t'=0}^{C_j} \frac{1}{p_j} \cdot \mathsf{1}\Big[\frac{\hat{y}_j(t')}{p_j} \leq \zeta(t')\Big] \cdot \mathsf{1}\Big[j \in U(t') \setminus N(t')\Big] - \frac{1}{p_j} t \\
&\leq \sum_{t'=t}^{C_j} \frac{1}{p_j} \cdot \mathsf{1}\Big[\frac{\hat{y}_j(t')}{p_j} \leq \zeta(t')\Big] \cdot \mathsf{1}\Big[j \in U(t') \setminus N(t')\Big] \\
&= \sum_{t'=t}^{C_j} \frac{w_j}{y_j(t')} \cdot \frac{\hat{y}_j(t')}{p_j} \cdot \mathsf{1}\Big[\frac{\hat{y}_j(t')}{p_j} \leq \zeta(t')\Big] \cdot \mathsf{1}\Big[j \in U(t') \setminus N(t')\Big] .
\end{aligned}$$

The last equality is possible because for every $r_j \leq t' \leq C_j$, PF assigns job $j \in U(t)$ a positive rate $y_j(t') > 0$ and if $j \in U(t') \setminus N(t')$, then $y_j(t') = \hat{y}_j(t')$. By the KKT condition (6), we can substitute $\frac{1}{y_j(t')} = \sum_{d=1}^{D} b_{dj} \cdot \eta_d(t')$ above. Since for every $d \in [D]$ we have that

$$\begin{aligned}
\sum_{t'=t}^{C_j} b_{dj} \cdot \eta_d(t') \cdot \frac{\hat{y}_j(t')}{p_j} \cdot \mathsf{1}\Big[\frac{\hat{y}_j(t')}{p_j} \leq \zeta(t')\Big] &\leq \sum_{t'=t}^{C_j} b_{dj} \cdot \eta_d(t') \cdot \zeta(t') \\
&\leq b_{dj} \sum_{t' \geq t} \eta_d(t') \cdot \zeta(t') = \kappa b_{dj} b_{dt} ,
\end{aligned}$$

we obtain the right-hand side of the dual constraint by summing over all $d \in [D]$. □

We can finally prove Theorem 4.3.

Proof of Theorem 4.3. Weak duality, Lemma G.2, and Lemma G.5 imply

$$\kappa \cdot \mathsf{opt} \geq \sum_{j \in J} a_j - \sum_{d=1}^{D} \sum_{t \geq 0} b_{dt} \geq \Big(\lambda - \beta - \frac{1}{(1-\lambda)\kappa}\Big) \cdot \mathsf{alg}$$

for all parameters that satisfy $\lambda - \beta - \frac{1}{(1-\lambda)\kappa} > 0$.

We now set $\lambda := \frac{\beta+2}{3}$ and $\kappa := \frac{9}{(1-\beta)^2}$, which satisfy the above condition. This gives

$$\lambda - \beta - \frac{1}{(1-\lambda)\kappa} = \frac{2(1-\beta)}{3} - \frac{1-\beta}{3} = \frac{1-\beta}{3} ,$$

and thus, we conclude that $\mathsf{alg} \leq \frac{27}{(1-\beta)^3} \cdot \mathsf{opt}$. □

H Other related work

Since various objectives are considered in the literature, we mostly focus on the total (weighted) completion time objective. Unless otherwise specified, we assume that preemption is permitted in the results discussed below. For brevity, "completion time" will hereafter refer to the objective of minimizing the sum of completion times.

Single Machine: In the clairvoyant scenario, it is folklore that Shortest Remaining Time First (SRPT) is optimal. Note that SRPT becomes Shortest Job First (SJF), which processes jobs in non-decreasing order of their processing time. For non-clairvoyant scheduling, Round-robin (RR) is known to be 2-competitive for completion time, which is the best competitive ratio achievable in this setting

[44]. Shortest Elapsed Time First (SETF) processes all jobs that have received the least amount of processing equally; note that SETF coincides with RR if all jobs arrive simultaneously. Recently, it was shown that SETF is 2-competitive for (weighted) completion time [29].

While being $O(1)$-competitive, both SETF and RR incur excessively many preemptions. Multilevel (Adaptive) Feedback (MLF) seeks to minimize the number of preemptions while retaining $O(1)$-competitiveness for completion time. MLF can be parameterized to achieve a $(2+\epsilon)$-competitive ratio for any $\epsilon > 0$ [44].

Identical Machines: The performance guarantees of SRPT and RR for a single machine extend to parallel identical machines [7, 44]; see also [14, 43] for closely related work. Under the resource augmentation framework, SETF is shown to be $O(1)$-competitive for identical machines [30].

Completion Time vs. Flow Time: The optimal schedule for total completion time is identical to that of the total flow time objective, where a job's flow time is defined as $F_j = C_j - r_j$. However, from an approximation and competitive analysis standpoint, the flow time objective is significantly more challenging. Consequently, online algorithms are often provided with extra speed and compared against the offline optimum. It is well-established that an algorithm that is $O(1)$-competitive with $O(1)$-speed for flow time is also $O(1)$-competitive for completion time. For example, while [30] primarily studies flow time minimization, their results readily translate to the completion time objective.

Unrelated Machines: In the offline setting, minimizing total completion time on unrelated machines is solvable in polynomial time if all jobs arrive at $t = 0$. If jobs have release times, the problem becomes NP-hard and, in fact, APX-hard [23]. Constant-factor approximations are well-established [6, 22, 27, 48, 49, 51], and the current state-of-the-art approximation ratio is $1.36 + \varepsilon$ due to [35]. In the clairvoyant online setting, utilizing immediate-dispatch and non-migratory algorithms while assuming each machine runs SPT or SRPT, assigning an arriving job to the machine with the least marginal increase in the objective is $O(1)$-competitive [1, 20]. For non-clairvoyant algorithms, SelfishMigrate and Proportional Fairness (PF) were analyzed with competitive ratios of 32 and 4.62, respectively [25, 28]. Various clairvoyant algorithms were also explored in [9, 21, 39].

Polytope Scheduling: Polytope scheduling is a general framework that captures various scheduling objectives and encompasses all the aforementioned problems. For this general scheduling problem, [24] showed that Proportional Fairness (PF) is 128-competitive for total weighted completion time, a result that was later improved to 27 by [28]. Furthermore, [28] demonstrated that PF achieves surprisingly strong competitive ratios for various special cases, including the unrelated machines setting.

Online Scheduling with Predictions: As alluded to previously, scheduling has become a primary application for ML-augmented algorithms. Various objectives were considered. For example, total completion time [15, 26, 37, 46], total flow time [3, 40], makespan minimization [4, 32, 36], and minimizing energy consumption [2, 5].

I Further experimental details and results

I.1 Algorithm details

We provide further details of the algorithms.

1. **SNAP** (coupled with PMLF): Recall that the SNAP algorithm aims to process each job j by $v_{j,k}$ units in epoch k. While a machine i could technically idle after meeting the specific processing targets $v_{j,k}$ for its assigned jobs, such idling results in an unnecessary loss of efficiency.

   Therefore, we introduce an engineering adaptation: while jobs are assigned to machines following Step 4, we execute the **PMLF** (Preemptive Multi-Level Feedback) policy on each individual machine to process its assigned jobs. An epoch concludes once a β fraction of jobs are exhausted.

   This approach serves as a practical adaptation that maintains the underlying spirit and analysis of SNAP. Each epoch is designed to exhaust a constant fraction of jobs, mimicking the schedule of **PF** (Proportional Fairness). This behavior is indirectly achieved through

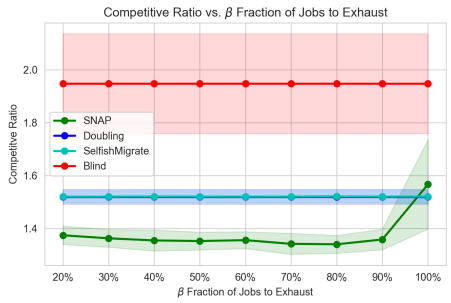


Figure 2: Competitive ratio of SNAP vs. exhaustion parameter β. Performance remains stable until reaching a 100% exhaustion rate.

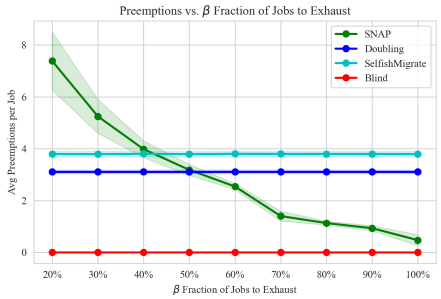


Figure 3: Preemption frequency of SNAP for different values of the exhaustion parameter β.

PMLF; we know Step 4 processes jobs without exhausting more than βn_k jobs within $4l_k$ time steps. Given $v_{j,k} = l_k y_j^{(k)}$ for each job $j \in F_k$ where $|F_k| = (1-\beta)n_k$, exhausting more than βn_k jobs ensures we are effectively mimicking PF without being constrained by rigid checkpoints. This provides the intuition for using PMLF.

2. **Blind Algorithm**: This policy is non-migratory, ensuring that no job is ever processed on multiple machines throughout its duration. The algorithm is equivalent to that of [1] assuming that $p_j = \hat{p}_j$ for all jobs j. The algorithm operates under the assumption of 100% trust in the provided predictions. Let t denote the current time and $q_j(t)$ represent the volume of job j processed prior to t. The predicted remaining size of job j is thus given by $\hat{p}_j - q_j(t)$.

   On each machine i, we execute the Shortest Job First (SJF) policy (or equivalently SRPT). We define the *residual estimate* for machine i as the total completion time that would be incurred by running SJF on that machine based on these predicted remaining sizes.

   Upon the arrival of a new job with a predicted size, we compute the potential increase in the residual estimate for every machine. The job is then assigned to the machine that yields the minimum marginal increase to the total residual estimate.

3. **Doubling** (Coupled With Immediate-Dispatch): Initially, each job's size is assumed to be equal to its predicted size $\hat{p}_j$. If a job's actual processing time exceeds this estimate, we double the predicted size (i.e., $\hat{p}_j \leftarrow 2\hat{p}_j$). In this event, we treat the original job as completed and consider a new copy of the job to have arrived. This copy is dispatched to the machine that results in the minimum increase to the residual estimate.

4. **Hybrid SNAP**: Initially, all jobs are initially in Group 1. Jobs in Group 1 are dispatched to machines using the Doubling strategy. To loosely emulate the Blind strategy, we utilize large milestones: $\{c(1+\delta)^i \hat{p}_j \mid i \geq 1\}$ for a large parameter $c \geq 1$. Any job that reaches its first milestone permanently migrates to Group 2. Group 2 jobs are assigned to machines using the SNAP algorithm. Finally, we execute PMLF on individual machines to process their assigned jobs.

I.2 Further experimental results

We conduct further experiments to evaluate the performance of our algorithms. The first experiment investigates the competitive ratio and preemption frequency of SNAP as a function of the exhaustion parameter β, with $\delta = 1$ and $R = 512$. Interestingly, SNAP exhibits a remarkably stable competitive ratio for most values of β; the performance degrades sharply only when β reaches 100%. In contrast, the preemption frequency is highly sensitive to β, decreasing significantly as β increases. Note that since other algorithms do not utilize the β parameter, their performance remains constant across these values.

The second experiment compares the algorithms across various values of δ. In our implementation, δ defines the checkpoints for SNAP and serves as a parameter for the PMLF subprocedure. Intuitively, a larger δ implies a higher threshold for processing before a preemption occurs. Similarly, our implementation of SelfishMigrate utilizes δ within its SNAP subprocedure. In the Doubling approach,

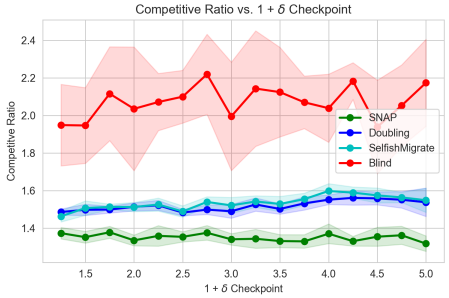


Figure 4: Competitive ratio of the evaluated algorithms for different values of δ.

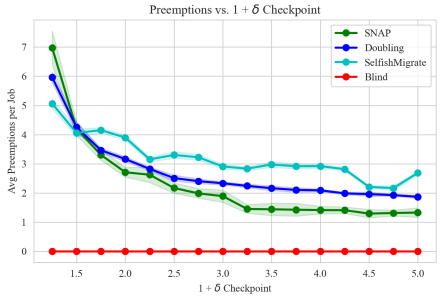


Figure 5: The preemption frequency of the evaluated algorithms for different values of δ.

Table 1: Competitive ratio across special job and machine percentages for different algorithms ($\delta = 1$, $\beta = 0.7$, $R = 256$). The Hybrid SNAP (c) algorithm is categorized by a parameter c for larger initial milestones.

Algorithm	**0%**	**10%**	**20%**	**30%**	**40%**	**50%**
Blind	**1.3076**	1.7923	1.8279	1.6468	1.6101	1.575
Doubling	1.6983	1.5098	1.5047	1.5375	1.5438	1.606
Hybrid SNAP (1)	1.5166	1.4216	**1.3274**	1.389	1.383	1.6093
Hybrid SNAP (2)	1.5353	1.6287	1.4522	1.3347	1.4919	**1.3347**
Hybrid SNAP (4)	1.4559	1.4178	1.4824	1.4894	1.3912	1.3448
Hybrid SNAP (6)	1.5221	1.3954	1.4255	**1.2963**	1.3854	1.3554
Hybrid SNAP (8)	1.3879	1.3994	1.3813	1.3924	**1.3005**	1.3927
SNAP	1.4943	**1.3562**	1.359	1.4599	1.5124	1.5315

we simulate a job completion and the release of a new job whenever the processing reaches a power of $(1+\delta)$. Thus, while the algorithms (with the exception of Blind) utilize δ differently, the parameter serves a similar purpose across all of them. For this experiment, we set $\beta = 0.6$ and $R = 512$. As δ increases, the competitive ratios of Doubling and SelfishMigrate increase mildly, while that of SNAP decreases slightly. However, the impact of δ on preemption frequency is more pronounced: the frequency drops consistently as δ increases across all applicable algorithms.

Our final experiment evaluates Hybrid SNAP by comparing the competitive ratios of the algorithms across different special job and machine frequencies, with parameters set at $\beta = 0.7$ and $R = 256$. We observe that in the absence of special jobs, the Blind algorithm performs effectively. This is because the machines are essentially identical in this scenario, allowing even a blind strategy to achieve efficient load balancing across resources. However, once heterogeneity is introduced (across all tested frequencies from 10% to 50%), SNAP and Hybrid SNAP consistently yield the superior competitive ratios. In particular, Hybrid SNAP with $c = 8$ significantly outperforms both Blind and Doubling. This experiment demonstrates the robust performance of our proposed algorithms across varying degrees of system heterogeneity.

J Reproducibility and code implementation

Initial implementations of the experimental code and plotting scripts were generated using Gemini 3 Pro. All generated code was subsequently refactored, manually verified, and rigorously tested to ensure the technical accuracy of the algorithm's implementation and the reproducibility of the results. The full source code is provided in the supplemental material.