# Leveraging Speed Predictions for Online Energy-Efficient Scheduling

**Eric Balkanski** **Jingwei Li** **Clifford Stein** **Cherlin Zhu**
Department of Industrial Engineering and Operations Research
Columbia University
eb3224@columbia.edu, jl6639@columbia.edu
cliff@ieor.columbia.edu, cz2740@columbia.edu

## Abstract

We consider the scheduling problem of online speed scaling where the goal is to minimize the energy consumption of a machine that controls the speed at which jobs are processed. Recent work has leveraged the learning-augmented framework, where the algorithm is provided with predictions about jobs that will arrive in the future, to manage power usage more efficiently. This paper proposes a novel prediction model for speed scaling where the predictions are about the machine speed (the output), instead of the jobs (the input). Machine speed predictions have multiple advantages: they are succinct, admit strong PAC-learnability guarantees, can be provided dynamically, and lead to a natural definition of smoothness. We give an algorithm for dynamic machine speed predictions that is $(1+\epsilon)$-consistent and $O(1)$-robust. For offline machine speed predictions and job speed predictions, we provide an algorithm that achieves the stronger guarantee of $(1+\epsilon)$-smoothness, while maintaining $O(1)$-robustness. These guarantees are comparable to previous work, but do not require predicting the entire input.

## 1 Introduction

Massive data centers that power modern artificial intelligence engines are placing significant pressure on power grids and sustainability goals. The electricity consumption from data centers, artificial intelligence (AI), and the cryptocurrency sector is experiencing tremendous growth and is projected to double during the 2024-26 period (19). Consequently, energy is becoming one of the limiting factors for AI's scale and deployment, and a central challenge is to minimize the energy consumption caused by this growing demand for AI, while still achieving high performance.

Energy consumption is typically controlled by the speed at which machines are run. In speed scaling problems, the operating system/algorithm designer controls the speed of a machine, and the power usage of a machine is typically modeled as the speed to some power $\alpha > 1$. The total energy consumption is then power integrated over time. The goal is to scale the speed at which jobs run in order to minimize total energy consumption while satisfying certain performance requirements, such as meeting deadlines.

Many variants of speed scaling problems have been studied in the online setting where jobs arrive over time and future jobs are unknown (see, e.g., (2; 20; 10; 3; 12)). Yao et al. (25), who introduced the problem of speed scaling, proposed the Optimal Available (OA) algorithm for the online setting with deadlines, which was later shown to achieve a competitive ratio of $\alpha^\alpha$ by Bansal et al. (9). At each job arrival, OA recomputes an optimal schedule for all released, unfinished jobs using their remaining work, and follows it until the next arrival. The best-known lower bound is $e^{\alpha-1}/\alpha$ (11). Since treating future jobs as entirely unknown is often overly pessimistic, there has been recent

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

interest in using the framework of algorithms with predictions to achieve improved performance guarantees in scheduling, including in online speed scaling with deadlines (8; 4; 7).

A crucial modeling choice in algorithms with predictions is what should be predicted. This paper argues for a new framework in speed scaling where the predictions are about the machine speed (the output of the algorithm), instead of the parameters of the jobs (the input to the algorithm). Three major advantages of speed predictions are that they provide succinct predictions, suggest a natural definition of smoothness, and can be dynamic. Our scheduling algorithms and their guarantees do not depend on how the predictions are generated; in particular, they do not require the PAC-learning procedure in Section 5. Thus, an existing heuristic speed controller can supply the predictions, and our framework provides a worst-case robustness guarantee even if the controller has none.

**Succinct predictions.** A job consists of at least three parameters: an arrival time, a deadline, and an amount of work to be completed. In some models, it can also have a weight, a speed limit, or a job-specific power factor for energy consumption. With job predictions, previous work required predictions for all the parameters of all the jobs. Instead, speed predictions only provide predictions regarding the machine speed and are more succinct. In Section 5, we show that speed predictions contain strictly less information than job predictions and prove they admit PAC-learning guarantees.

**Smoothness.** The competitive ratio of an algorithm with predictions is often called smooth if it smoothly interpolates between its consistency (competitive ratio when predictions are accurate) and robustness (competitive ratio when predictions are arbitrarily wrong) as a function of a measure of the prediction error. Existing measures of prediction error are often incomparable, lack a consensus on how they should be defined, and can be relatively complex (especially in speed scaling since each job consists of multiple parameters). Recent work, which also uses output predictions, has considered an alternate approach where smoothness is defined as the approximation ratio with respect to the cost of the output prediction (see, e.g., (15; 18; 6)). The main benefit of this approach, which we also use, is that it does not require defining a measure of prediction error, while providing meaningful guarantees not only when the predictions are exactly accurate (which is required for consistency) but also when they are approximately accurate.

**Dynamic predictions.** In prior work on speed scaling with predictions, all predictions are provided at time $t = 0$. Speed predictions can instead be provided dynamically: at every time $t$, the algorithm receives a prediction of the speed at which the machine should run at that time. This allows predictions to incorporate data observed about the instance up to time $t$, rather than being provided entirely up-front, and thus to adapt and improve in real time. For comparison, current workload measures, such as the total remaining work of released jobs, are already known at time $t$. The optimal speed at $t$ can nevertheless depend on unknown future jobs, so predicting the current speed can provide additional information.

Our two main contributions are:

1. **Novel speed prediction models** for speed scaling. We call them the dynamic machine speed, offline machine speed, and dynamic job speed prediction models. While our definitions of consistency and robustness are standard, we propose a definition for smoothness (c.f. Definition 2.3) that does not require defining a measure of the prediction error and allows for infeasible predictions.

2. **Algorithms.** We give two main algorithms. The first algorithm achieves, for any constant $\epsilon > 0$, $(1 + \epsilon)$-consistency and $O(1)$-robustness with dynamic machine speed predictions (Theorem 3.4). The second obtains the stronger guarantees of $(1+\epsilon)$-smoothness and $O(1)$-robustness with either offline machine speed or dynamic job speed predictions (Theorem 4.1 and Theorem 4.2).

In comparison, Bamas et al. (8); Antoniadis et al. (4); Balkanski et al. (7) also achieve $(1 + \epsilon)$-consistency and $O(1)$-robustness guarantees, but for the easier problem with the more powerful predictions about the entire input. In restricted cases, they also achieve smoothness guarantees, but for a different notion of smoothness that is as a function of a prediction error (which is defined differently in each of these prior works). We note that our smoothness guarantee requires offline machine speed or dynamic job speed predictions. Finding a smooth algorithm for the dynamic machine speed prediction model is an intriguing open problem.

Finally, in Section 6, we provide empirical evaluations on synthetic and real data that demonstrate that our algorithms outperform the best-known algorithm without predictions when the predictions are reasonably accurate, while also being robust to poor predictions.

**Additional discussion of related work.** A recent paper by Lindermayr et al. (22) shares a similar title but studies a different scheduling problem. They consider speed-oblivious scheduling with predictions of job- and machine-dependent processing rates, whereas we consider predictions of the machine speed profile in energy-efficient speed scaling. Balkanski et al. (7) considered a more general speed scaling setting that also includes the energy plus flow time minimization problem. Similar to their algorithm called TPE, we partition jobs based on whether they should be processed according to the prediction or instead run by an online algorithm that ignores predictions. The key difference lies in how this partition is constructed, as our more succinct speed predictions require a more delicate algorithm and analysis than the full-instance predictions used in TPE. Most notably, our algorithm may start processing a job according to the prediction but then reassign it mid-processing to the online algorithm, which complicates the analysis (see, e.g., Lemma 3.1).

We also compare against TPE empirically, and our algorithms achieve comparable or better performance even on instances designed to be favorable to TPE. Although output predictions have not previously been considered for online speed scaling, they have been studied in the algorithms with predictions literature, including for problems such as metrical task systems (5; 15), list update (6), non-clairvoyant scheduling (21), clock auctions (18), and facility location (1; 16).

## 2 Preliminaries

In **online speed scaling**, the input is a set $J$ of $n$ jobs where each job $j \in J$ has a release date $r_j$, a deadline $d_j > r_j$, and an amount of work $w_j$ required to be completed. The jobs are independent and are processed on a single machine. Preemption is allowed without overhead: a job may be interrupted and resumed later. The jobs are revealed online according to their release date $r_j$. Time is continuous, and the machine speed satisfies $s(t) \in [0, \infty)$. At each time $t$, an algorithm determines a machine speed $s(t)$ and processes at most one released, unfinished job. The job-selection function $\sigma(t)$ identifies the job being processed at that time, with $\sigma(t) = \perp$ when no job is processed. For the analysis, we also use a fractional representation: $s_j(t) \geq 0$ denotes the processing rate allocated to job $j$ at time $t$, and $s(t) = \sum_{j \in J} s_j(t)$ is the aggregate rate. This representation is an analytical device for sharing processing capacity among jobs. We refer to $(s, \sigma)$ as a machine speed schedule and $\{s_j\}_{j \in J}$ as a job speed schedule. With cost-free preemption, fractional processing can be implemented by single-job time-sharing while preserving feasibility and energy cost; see Appendix A.1. The work processed for job $j$ under a job speed schedule $\{s_j\}_{j \in J}$ is $\int_{r_j}^{d_j} s_j(t)dt$. A job speed schedule is feasible if all jobs are completed by their deadlines: $\int_{r_j}^{d_j} s_j(t)dt \geq w_j$ for all $j \in J$. The power consumption at time $t$ is $s(t)^\alpha$, where $\alpha > 1$ is the energy parameter, which is assumed to be constant. Thus, the energy cost of a job speed schedule is

$$\text{cost}(\{s_j\}_{j \in J}) := \int_0^\infty \Big( \sum_{j \in J} s_j(t) \Big)^\alpha dt = \int_0^\infty s(t)^\alpha \, dt. \tag{1}$$

The goal is to find a feasible schedule that minimizes energy cost. A machine speed schedule $(s, \sigma)$ has energy cost $\int_0^\infty s(t)^\alpha \, dt$ and is feasible if it processes at least $w_j$ units of work on each job $j$ during $[r_j, d_j]$. Given two machine speed schedules $s$ and $s'$, $s \leq s'$ if $s(t) \leq s'(t)$ for all $t$.

There exists an optimal schedule in which each job $j$ is processed at a single constant speed $\ell_j^\star$ whenever it executes, and has processing rate zero otherwise; $\ell_j^\star$ is called its *level speed* (25). The main challenge for an online algorithm is to control the tradeoff between running at a slow speed in case no jobs arrive in the future and running at a fast speed in case a large workload arrives in the near future.

**Example.** Consider two jobs with $(r_1, d_1, w_1) = (0, 4, 2)$ and $(r_2, d_2, w_2) = (1, 3, 2)$. A feasible schedule runs at speed 1 on $[0, 4)$ and speed 0 otherwise. Earliest-deadline-first (EDF) scheduling always selects a released, unfinished job with the earliest deadline. In this example, EDF processes $j_1$ on $[0, 1)$, preempts it to process $j_2$ on $[1, 3)$ because $d_2 < d_1$, and resumes $j_1$ on $[3, 4)$. Thus, $\sigma(t)$ follows this sequence, each job receives two units of work, and both jobs have level speed 1, although $j_1$ has processing rate zero while preempted. This schedule has energy cost 4 and is optimal by convexity; see Appendix A.1.

**The predictions.** Jobs arrive online in all models in Table 1. In the *dynamic machine speed* model, $\hat{s}(t)$ is revealed at time $t$; in the *offline machine speed* model, the entire function $\hat{s}$ is revealed at

Table 1: Guarantees with a $c_{\text{on}}$-competitive online subroutine, for fixed $\alpha > 1$ and $\varepsilon \in (0, 1)$. The $O(\cdot)$ notation hides only constants depending on $\alpha$; OA gives $c_{\text{on}} = \alpha^\alpha$.

| Prediction model | Prediction revealed | Result | Consistency | Robustness | Smoothness |
|---|---|---|---|---|---|
| Dynamic machine speed prediction | $\hat{s}(t)$ at each time $t$ | Theorem 3.4 | $1+\varepsilon$ | $O\left(\frac{c_{\text{on}}}{\varepsilon^\alpha}\right)$ | — |
| Offline machine speed prediction | Entire profile $\hat{s}$ at time $t = 0$ | Theorem 4.1 | $1+\varepsilon$ | $O\left(\frac{c_{\text{on}}}{\varepsilon^\alpha}\right)$ | $1+\varepsilon$ |
| Dynamic job speed prediction | $\hat{\ell}_j$ at each job release $r_j$ | Theorem 4.2 | $1+\varepsilon$ | $O\left(\frac{c_{\text{on}}}{\varepsilon^\alpha}\right)$ | $1+\varepsilon$ |
| Full-input prediction (TPE) | All predicted jobs' release times, deadlines, and work at $t = 0$ | Balkanski et al. (7), Corollary 3.5 | $1+\varepsilon$ | $O\left(\frac{c_{\text{on}}^\alpha}{\varepsilon^\alpha}\right)$ | Error-based |

The TPE row uses an optimal offline solver and the confidence parameter for $(1+\varepsilon)$-consistency in Balkanski et al. (7, Corollary 3.5). Our smoothness bounds are relative to MAKEFEASM and MAKEFEASJ, respectively. TPE's error-dependent bound (7, Theorem 3.4) uses a different notion of smoothness. A dash indicates no guarantee given here.

$t = 0$. In the *dynamic job speed* model, a prediction $\hat{\ell}_j \in (0, \infty)$ of job $j$'s optimal level speed $\ell_j^\star$ is revealed at $r_j$. These job predictions induce a machine speed schedule $\hat{s}$ by processing jobs in EDF order at speeds $\hat{\ell}_j$.

We let $\text{ALG}(J, \hat{s})$ be the schedule returned by an algorithm ALG over instance $J$ and prediction $\hat{s}$, which can be dynamic or offline. The optimal offline schedule, which knows $J$ at $t = 0$, is denoted by $\text{OPT}(J)$. For independent jobs with arbitrary preemption and a fixed machine speed function, if some assignment of jobs to the available processing capacity meets all deadlines, then EDF does as well (25). Since energy depends only on the speed function, we sometimes omit $\sigma$ and refer to a machine speed schedule by $s$, with EDF as the implicit job-selection rule. In particular, we let $\text{OPT}(J)$ denote the machine speed function $s^\star$ such that $s^\star$ with the EDF job selection $\sigma$ is an optimal machine speed schedule for $J$. $s^\star$ is unique by Yao et al. (25). Consistency is the competitive ratio when predictions are accurate, while robustness refers to the competitive ratio when predictions are arbitrarily wrong.

**Definition 2.1.** An algorithm ALG is $\gamma$-consistent and $\beta$-robust if

$$\max_J \frac{\text{cost}(\text{ALG}(J, \hat{s} = \text{OPT}(J)))}{\text{cost}(\text{OPT}(J))} \le \gamma \qquad \text{and} \qquad \max_{J, \hat{s}} \frac{\text{cost}(\text{ALG}(J, \hat{s}))}{\text{cost}(\text{OPT}(J))} \le \beta.$$

Another performance metric, which has been referred to as smoothness[1], is the approximation ratio achieved by an algorithm against the benchmark that simply outputs the prediction. Smoothness is stronger than consistency since it allows achieving strong competitive ratio guarantees not only when the predictions are exactly accurate, but also when they are approximately accurate and the output prediction performs well. This notion of smoothness was first considered in Mahdian et al. (24) and more recently in the context of output predictions.

An important modeling choice for smoothness in the context of speed scaling is how to define it when the prediction is infeasible, which occurs when the predicted machine speed is not fast enough to complete all jobs by their deadlines. One simple option is to define the cost of infeasible predictions to be infinity. This option is undesirable, as it does not provide meaningful guarantees when the predicted speed is both infeasible and near-optimal. This scenario happens, for example, when the prediction did not anticipate one small job. Instead, we propose a stronger and more challenging benchmark where, in such scenarios, the goal is to compete against a feasible solution that is close to the infeasible prediction. This feasible solution is obtained with a feasibility projection of the prediction $\hat{s}$, which we define next.

[1] Azar et al. (6) call it smoothness, Christianson et al. (15) consistency, and Gkatzelis et al. (18) consistency$^\infty$.

**Definition 2.2.** An algorithm $\textsc{MakeFeas}(J, \hat{s})$ is a feasibility projection if it returns a machine speed schedule such that, for all instances $J$ and machine speed prediction $\hat{s}$:

- $\textsc{MakeFeas}(J, \hat{s})$ is feasible,
- if $\hat{s}$ is feasible, then $\textsc{MakeFeas}(J, \hat{s}) = \hat{s}$, and
- $\textsc{MakeFeas}$ is an online algorithm, i.e., the speed at time $t$ of schedule $\textsc{MakeFeas}(J, \hat{s})$ depends only on jobs and predictions that are known at time $t$.

**Definition 2.3.** An algorithm ALG is $\zeta$-smooth with respect to feasibility projection $\textsc{MakeFeas}$ if for all instances $J$ and prediction $\hat{s}$, we have $\frac{\text{cost}(\text{ALG}(J,\hat{s}))}{\text{cost}(\textsc{MakeFeas}(J,\hat{s}))} \leq \zeta$.

This guarantee compares directly with the cost of the prediction after applying the specified feasibility projection, without requiring a numerical measure of prediction error. When the prediction $\hat{s}$ is feasible, then $\textsc{MakeFeas}(J, \hat{s}) = \hat{s}$, and ALG must be a $\zeta$-approximation to the cost $\text{cost}(\hat{s})$ of the prediction. We note that 1-smoothness implies 1-consistency. In addition, since $\textsc{MakeFeas}(J, \hat{s})$ is an online algorithm, a trivial approach is to follow $\textsc{MakeFeas}(J, \hat{s})$, which achieves 1-smoothness but has unbounded robustness. Consistency, robustness, feasibility projection, and smoothness extend to dynamic job speed predictions via the induced machine speed schedule.

## 3 Dynamic Machine Speed Predictions

This section treats dynamic machine speed predictions (Table 1, first row). For any constant $\epsilon > 0$, the algorithm is $(1 + \epsilon)$-consistent and $O(1)$-robust and runs in polynomial time. Missing proofs are in Appendix A unless otherwise stated.

**The algorithm.** The algorithm, defined in Algorithm 1, splits the jobs into two sets $J^{\text{p}}$ and $J^{\text{on}}$. It uses the machine speed predictions $\hat{s}(t)$ to process jobs in $J^{\text{p}}$. For the jobs in $J^{\text{on}}$, it ignores the predictions and processes them according to an arbitrary algorithm $\textsc{Online}$ for online speed scaling without predictions. For example, the Optimal Available (OA) algorithm can be used as $\textsc{Online}$. Our guarantees are expressed in terms of the competitive ratio $c_{\text{on}}$ of $\textsc{Online}$, so improvements to this subroutine carry over to our bounds. The jobs in $J^{\text{p}}$ are for the consistency guarantee and those in $J^{\text{on}}$ for the robustness guarantee. The crux of the algorithm is the construction of $J^{\text{p}}$ and $J^{\text{on}}$.

The algorithm first considers the set $\Sigma$ of all schedules that, depending on potential future jobs, could be an optimal schedule for the online instance, assuming accurate predictions so far. If $\Sigma = \emptyset$ then the predictions must have been wrong, so we stop trusting them and assign all future jobs to $J^{\text{on}}$. Otherwise, we rely on a schedule $s_{\text{F}}$, the *forecast schedule*, defined as the minimum cost schedule in $\Sigma$ (computed without enumerating $\Sigma$ by $\textsc{Forecast}$, Algorithm 7 in Appendix E). When a job is released at $t$, the algorithm uses the $\textsc{Assign}$ subroutine (Algorithm 4) to decide if it should be added to $J^{\text{on}}$ or $J^{\text{p}}$.

The forecast schedule $s_{\text{F}}$ can have a cost that is too large to ensure robustness, so the algorithm maintains a schedule $s_{\text{R}}$ called the robust forecast schedule. This is a forecast schedule produced in the past whose cost is sufficiently small, compared to the cost of the jobs $J_{\leq t} := \{j \in J : r_j \leq t\}$ that have been released, so that robustness can be guaranteed (Line 8). A main challenge is that if $s_{\text{F}}(t) > s_{\text{R}}(t)$ (Line 9), then following the robust forecast $s_{\text{R}}(t)$ could be too slow to complete jobs in $J^{\text{p}}$. In that case, we use a subroutine called $\textsc{Reassign}$ (Algorithm 5) to reassign a job from $J^{\text{p}}$ to $J^{\text{on}}$. At any time $t$, we run jobs in $J^{\text{on}}$ according to $\textsc{Online}$ (Line 11). Simultaneously, we run jobs in $J^{\text{p}}$ according to the robust schedule $s_{\text{R}}$ (Line 13): we run the job that $s_{\text{R}}$ runs at time $t$ if it is in $J^{\text{p}}$.

The loop over all $t \geq 0$ specifies a continuous-time policy. Under the piecewise-constant representations used in Appendix E, an event-driven execution integrates processing between events and must account for releases, deadlines, prediction and schedule breakpoints, job completions, and changes in the algorithm's branch conditions.

The pseudocode for the subroutines $\textsc{Assign}$ and $\textsc{Reassign}$ are provided in Appendix A.2. The subroutine $\textsc{Assign}$ assigns job $j$ to $J^{\text{on}}$ if doing so maintains that the cost of $J^{\text{on}}$ is bounded by the scaled cost of the forecast schedule $s_{\text{F}}$, which is needed for the consistency guarantee. The scaling is controlled by the parameter $\lambda$, which represents how much the algorithm trusts the predictions (with smaller $\lambda$ meaning higher trust).

**Algorithm 1:** with dynamic machine speed predictions

1: **Input:** jobs $J$ released online, dynamic machine speed predictions $\hat{s}$, parameter $\lambda \in [0, 1]$
2: Initialize $J^{\mathrm{p}}, J^{\mathrm{on}} \leftarrow \emptyset$
3: **for** all $t \geq 0$ **do**
4: $\quad \Sigma \leftarrow \{s : s(t') = \hat{s}(t')$ for all $t' \leq t, \exists\, J'$ s.t. $s = \mathrm{OPT}(J')$ and $J'_{\leq t} = J_{\leq t}\}$
5: $\quad$ **if** $|\Sigma| > 0$ **then**
6: $\quad\quad s_{\mathrm{F}} \leftarrow \arg\min_{s \in \Sigma} \mathrm{cost}(s)$
7: $\quad\quad$ **if** a job $j$ has release date $r_j = t$ **then** $J^{\mathrm{on}}, J^{\mathrm{p}} \leftarrow \textsc{Assign}(j, s_{\mathrm{F}}, \lambda, J^{\mathrm{on}}, J^{\mathrm{p}})$
8: $\quad\quad$ **if** $\lambda \cdot \mathrm{cost}(s_{\mathrm{F}}) \leq \mathrm{cost}(\mathrm{OPT}(J_{\leq t}))$ **then** update $s_{\mathrm{R}} \leftarrow s_{\mathrm{F}}$
9: $\quad\quad$ **else if** $s_{\mathrm{F}}(t) > s_{\mathrm{R}}(t)$ **then** $J^{\mathrm{on}}, J^{\mathrm{p}} \leftarrow \textsc{Reassign}(s_{\mathrm{F}}, J^{\mathrm{on}}, J^{\mathrm{p}}, t)$
10: $\quad$ **else if** a job $j$ has release date $r_j = t$ **then** add $j$ to $J^{\mathrm{on}}$
11: $\quad$ Run jobs $J^{\mathrm{on}}$ at speed $\textsc{Online}(J^{\mathrm{on}})$
12: $\quad$ Let $j$ be the job that $s_{\mathrm{R}}$ runs at time $t$
13: $\quad$ **if** $j \in J^{\mathrm{p}}$ **then** run $j$ at speed $s_{\mathrm{R}}(t)$

The subroutine REASSIGN is called when a job $j$ needs to be reassigned from $J^{\mathrm{p}}$ to $J^{\mathrm{on}}$ to ensure that the robust forecast schedule $s_{\mathrm{R}}$ can complete jobs in $J^{\mathrm{p}}$. The job $j$ reassigned is the job that the forecast schedule $s_{\mathrm{F}}$ would have run at time $t$ when running jobs in EDF order. This job is split into two jobs $j'$ and $j''$ where job $j'$ has work equal to the work that has already been completed for $j$. Job $j'$ corresponds to the part of $j$ that has already been completed and it remains in $J^{\mathrm{p}}$ to reflect that it was processed while in $J^{\mathrm{p}}$. Job $j''$, which corresponds to the remaining part of $j$ that needs to be processed, is moved to $J^{\mathrm{on}}$ as a new job, so its release time is the current time $t$.

**Analysis of Algorithm 1.** We let $\mathrm{ALG}(J, \hat{s})$ denote the job speed schedule of Algorithm 1. Fixing instance $J$ and prediction $\hat{s}$, we take $J^{\mathrm{on}}$ and $J^{\mathrm{p}}$ to be defined as at the end of Algorithm 1. To bound the cost of $\mathrm{ALG}(J, \hat{s})$, we separately bound the cost due to running jobs in $J^{\mathrm{on}}$ (Line 11) and $J^{\mathrm{p}}$ (Line 13) with respect to the cost of the whole instance. A main challenge is that $J^{\mathrm{on}}$, through reassignment, contains jobs whose release dates have been increased, because for $j \in J$ and $j''$ defined in Algorithm 5, $r_{j''} > r_j$ may hold. A job with a later release date (but same deadline) may need to run faster to complete, which could make the cost of $J^{\mathrm{on}} \cup J^{\mathrm{p}}$ exceed that of $J$.

The following three lemmas bound the cost of $\mathrm{ALG}(J, \hat{s})$: the first shows that the reassignment of jobs does not increase the cost, and the next two bound the cost of running jobs in $J^{\mathrm{on}}$ and $J^{\mathrm{p}}$, respectively. For a schedule $\mathcal{S} = \{s_j\}_{j \in J}$ and a subset $J' \subseteq J$, we let $\mathcal{S}[J'] = \{s_j\}_{j \in J'}$ denote the subschedule over $J'$. Let us define $J^{\mathrm{on}}_t$ and $J^{\mathrm{p}}_t$ as $J^{\mathrm{on}}$ and $J^{\mathrm{p}}$ in the algorithm at time $t$.

**Lemma 3.1.** *Let $t^\star$ be the earliest time at which $\Sigma = \emptyset$. For any $J$, dynamic machine speed prediction $\hat{s}$, and $t < t^\star$, cost*$(\mathrm{OPT}(J^{on}_t \cup J^{p}_t)) =$ *cost*$(\mathrm{OPT}(J_{\leq t}))$.

**Lemma 3.2.** *For any $\lambda \in (0, 1)$, jobs $J$, dynamic machine speed prediction $\hat{s}$, and $c_{on}$-competitive algorithm* ONLINE*, cost*$(\mathrm{ALG}(J, \hat{s})[J^{on}]) \leq 2^{\alpha+1} \cdot c_{on} \cdot$ *cost*$(\mathrm{OPT}(J))$. *If $\hat{s} = \mathrm{OPT}(J)$,* *cost*$(\mathrm{ALG}(J, \hat{s})[J^{on}]) \leq c_{on} \cdot \lambda \cdot$ *cost*$(\mathrm{OPT}(J))$.

**Lemma 3.3.** *For any $\lambda \in (0, 1)$, $J$, $\hat{s}$,* *cost*$(\mathrm{ALG}(J, \hat{s})[J^{p}]) \leq \min\{$*cost*$(\hat{s}),$ *cost*$(\mathrm{OPT}(J))/\lambda\}$.

Finally, we can state the main result. The consistency and robustness guarantees come from Lemmas 3.2 and 3.3. See Appendix E for the runtime proof.

**Theorem 3.4.** *For any $\epsilon \in (0, 1)$, Algorithm 1 with $\lambda = \frac{1}{c_{on}} \left(\frac{\varepsilon}{2^\alpha}\right)^\alpha$ is $(1 + \varepsilon)$-consistent and $c_{on} \cdot (2^{\frac{\alpha+1}{\alpha}} + \frac{2^\alpha}{\varepsilon})^\alpha$ robust, where $c_{on}$ is the competitive ratio of* ONLINE*. Furthermore, Algorithm 1 is implementable with runtime polynomial in $n$.*

Since $c_{\mathrm{on}} = \alpha^\alpha$ for OA (9), Theorem 3.4 implies that Algorithm 1 with ONLINE = OA is $(1 + \epsilon)$-consistent and $O(1)$-robust for any constant $\epsilon > 0$.

## 4 The Smooth Algorithm

This section treats offline machine speed and dynamic job speed predictions (Table 1, second and third rows), using a shared framework with separate guarantees below.

Offline machine speed predictions reveal the future speed profile, while dynamic job speed predictions provide each job's predicted level speed at release. Dynamic machine speed predictions reveal only the current aggregate speed, leaving both future speeds and their allocation among jobs unspecified. This missing information complicates constructing a feasibility projection for the smoothness benchmark.

**Shared framework.** To unify notation, let $\mathcal{P}$ be a prediction model and $P \in \mathcal{P}$ be a prediction. Let $P_{\leq t}$ denote what is known about $P$ at time $t$. For the offline machine speed prediction model, $P = P_{\leq t} = \hat{s}$ for all $t$. For the dynamic job speed prediction model, $P = \{\hat{\ell}_j\}_{j \in J}$ and $P_{\leq t} = \{\hat{\ell}_j\}_{j \in J : r_j \leq t}$.

The algorithm, described in Algorithm 2, uses a feasibility projection MAKEFEAS (recall Definition 2.2). This feasibility projection will be defined next and the smoothness guarantee of Algorithm 2 will be with respect to this feasibility projection. As in the algorithm in the previous section, Algorithm 2 partitions jobs into two sets $J^{\text{on}}$ or $J^{\text{p}}$. When job $j$ is released at time $t$, we decide if it is added to $J^{\text{on}}$ or $J^{\text{p}}$ depending on whether we can maintain the cost of $J^{\text{on}}$ being at most the cost of MAKEFEAS$(J_{\leq t}, P_{\leq t})$, scaled by $\lambda$ (Line 5). If we can maintain this bound, we assign $j$ to $J^{\text{on}}$ (Line 5). Otherwise, we assign $j$ to $J^{\text{p}}$ (Line 6). In contrast to Algorithm 1, Algorithm 2 compares the cost of $J^{\text{on}}$ to MAKEFEAS$(J_{\leq t}, P_{\leq t})$ instead of the forecast schedule $\arg\min_{s \in \Sigma} \text{cost}(s)$. We run jobs in $J^{\text{on}}$ according to an arbitrary online speed scaling algorithm (without predictions) ONLINE (Line 7) and jobs in $J^{\text{p}}$ according to MAKEFEAS$(J^{\text{p}}, P_{\leq t})$ (Line 7).

**Algorithm 2:**

1: **Input:** jobs $J$ released online, prediction $P$, $\lambda \in [0, 1]$
2: Initialize $J^{\text{p}}, J^{\text{on}} \leftarrow \emptyset$
3: **for** all $t \geq 0$ **do**
4: **if** a job $j$ has release date $r_j = t$ **then**
5: **if** cost(OPT$(J^{\text{on}} \cup \{j\})$) $\leq \lambda \cdot$ cost(MAKEFEAS$(J_{\leq t}, P_{\leq t})$) **then** add $j$ to $J^{\text{on}}$
6: **else** add $j$ to $J^{\text{p}}$
7: Run jobs $J^{\text{on}}$ at speed ONLINE$(J^{\text{on}})$ and jobs $J^{\text{p}}$ at speed MAKEFEAS$(J^{\text{p}}, P_{\leq t})$

**Offline machine speed predictions.** For this model (Table 1, second row), use MAKEFEASM (Algorithm 3). At every time $t$ it keeps track of the earliest deadline among available unfinished jobs, the available unfinished jobs that share this earliest deadline, and the total remaining work over these jobs (Lines 4 to 6). If following the predicted speeds up to the next deadline is not feasible for these jobs, the jobs are run at the maximum of the predicted speed and the minimum speed needed to complete these jobs by the deadline (Line 8), which is $\frac{W^u}{d-t}$. Otherwise, we just follow the prediction (Line 9).

**Algorithm 3:** MAKEFEASM

1: **Input:** Instance $J$, offline machine speed prediction $\hat{s}$
2: Initialize $t_c \leftarrow 0$, $J^u \leftarrow \emptyset$
3: **for** $t \leq \max_{j \in J} d_j$ **do**
4: $d \leftarrow$ earliest deadline for unfinished released jobs
5: $J^u \leftarrow$ unfinished released jobs with deadline $d$
6: $W^u \leftarrow$ remaining work of $J^u$
7: **if** $J^u = \emptyset$ **then** idle at speed $\hat{s}(t)$
8: **else if** $\int_t^d \hat{s}(\tau)\, d\tau < W^u$ **then** run the EDF job at speed $\max\{\hat{s}(t), \frac{W^u}{d-t}\}$
9: **else** run the EDF job at speed $\hat{s}(t)$

We note that if $\hat{s}(t) > 0$ when no jobs are pending, i.e., $W^u = 0$, then MAKEFEASM still incurs the energy cost associated with $\hat{s}(t)$, penalizing predicted speeds that are too large. Also note that for all $J$ and $\hat{s}$, (1) MAKEFEASM$(J, \hat{s})$ is feasible, (2) if $\hat{s}$ is feasible then MAKEFEASM$(J, \hat{s}) = \hat{s}$, and (3) MAKEFEASM is an online algorithm. Thus, MAKEFEASM is a feasibility projection.

For this instantiation, let ALG$(J, \hat{s})$ denote the resulting job speed schedule. The guarantee below is proved in Appendix B.1.

**Theorem 4.1.** *Fix any $\varepsilon \in (0,1)$. In the offline machine speed prediction model, Algorithm 2 with* MAKEFEAS = MAKEFEASM*, a $c_{on}$-competitive algorithm* ONLINE*, and $\lambda = \frac{1}{c_{on}}\left(\frac{\varepsilon}{2^\alpha}\right)^\alpha$ achieves $(1+\varepsilon)$-smoothness w.r.t.* MAKEFEASM *and $c_{on} \cdot (1+\frac{2^\alpha}{\varepsilon})^\alpha$-robustness.*

**Dynamic job speed predictions.** For this model (Table 1, third row), use MAKEFEASJ (Algorithm 6); the proof is in Appendix B.2.

**Theorem 4.2.** *Fix any constant $\varepsilon \in (0,1)$. In the dynamic job speed prediction model, Algorithm 2 with* MAKEFEAS = MAKEFEASJ *and a $c_{on}$-competitive algorithm* ONLINE *achieves, with $\lambda = \frac{1}{c_{on}}\left(\frac{\varepsilon}{2^\alpha}\right)^\alpha$, $1+\varepsilon$-smoothness and $c_{on} \cdot (1+\frac{2^\alpha}{\varepsilon})^\alpha$-robustness.*

With ONLINE = OA, both theorems give $(1+\varepsilon)$-smoothness and $O(1)$-robustness for fixed $\alpha > 1$ and $\varepsilon \in (0,1)$.

## 5 Learnability of Speed Predictions

In this section, we formalize the learnability comparison between the speed predictions that we introduce and the job predictions from previous work. All missing results and proofs are in Appendix D. The PAC result below concerns a fixed time grid and bounded speeds and workloads under a Lipschitz loss; these restrictions are specific to the learning analysis and are not assumptions of our scheduling guarantees. It provides a theoretical comparison of prediction representations, while Section 6 evaluates their use in scheduling rather than predictor training.

**Information-theoretic comparison.** We first note that speed predictions contain strictly less information than job predictions in the following sense. Given job predictions that correspond to an instance $J$, it is possible to compute a predicted optimal speed schedule for that instance by using the optimal algorithm for offline speed scaling of Yao et al. (25). The converse fails. A single optimal speed schedule is consistent with infinitely many distinct job instances, so it carries strictly less information than any one such instance. In Section D.1, we provide a construction which, given any optimal speed schedule, finds infinitely many distinct sets of jobs consistent with it.

**PAC-learnability.** Our main result in this section is a sample complexity bound for PAC learning of machine speed predictions. We assume there are $k$ time steps $0 \le t_1 < t_2 < \cdots < t_k$ such that each job has a release date $t_i$ and a deadline $t_j$ for some $i, j \in [k]$. Our goal is to bound the sample complexity of learning machine speed predictions as a function of $k$.

We first analyze the number of values that need to be learned in order to learn an optimal speed schedule (output prediction) or a set of jobs (input prediction). We note that an optimal schedule must be piecewise constant on the $k-1$ intervals $[t_1, t_2], \ldots, (t_{k-1}, t_k]$ since optimal speeds can only change when a job is released or at a job's deadline. Thus, learning an optimal speed schedule requires learning $k-1$ values: speeds $s_1, \ldots, s_{k-1}$. An input set of $n$ jobs can always be described with $3n$ values (work $w_j$, release date $r_j$, and deadline $d_j$). Alternatively, note that when two jobs have the same release date and deadline, it is equivalent to merge them into one job with workload that is the sum of the two jobs' workload. Thus, as a function of $k$, learning an input instance requires learning an aggregate workload $W_{i,j}$ for each release date-deadline pair $(t_i, t_j)$, so $\binom{k}{2} = \Theta(k^2)$ values. Thus, when $n = \Omega(k^2)$,[2] speed predictions require learning a linear (in $k$) number of values, while job predictions require learning a quadratic number of values.

We leverage this to derive sample complexity bounds via covering numbers (see Section D.2) and Rademacher complexity. In Section D.3, we use a generalization bound based on Dudley's entropy integral (Theorem D.2) to convert these covering bounds into the stated sample complexities. Assume sampled instances $J_1, \ldots, J_N$ are drawn i.i.d. from a distribution over $\mathcal{J}$, with risk $R(h) = \mathbb{E}_J[\ell(h; J)]$ and empirical risk $\hat{R}_N(h) = \frac{1}{N}\sum_{i=1}^N \ell(h; J_i)$. Let $\hat{h}$ denote the empirical risk minimizer over $\mathcal{H}$; its *excess risk* is $R(\hat{h}) - \inf_{h \in \mathcal{H}} R(h)$.

**Theorem 5.1.** *Let $\ell : \mathcal{H} \times \mathcal{J} \to [0,1]$ be L-Lipschitz in its first argument under $\|\cdot\|_\infty$. Assume the workloads satisfy $W_{i,j} \le B$ for all $i, j \in [k]$ and speeds satisfy $s_i \le S$ for all $i \in [k]$. To achieve*

[2]This assumption is natural when job arrivals are batched into discrete slots and deadlines align with system-level epochs.

*excess risk $\eta$ with probability at least $1-\delta$, it suffices to use*

$$N_{\text{speed}} = O\left(\frac{L^2S^2k + \log(1/\delta)}{\eta^2}\right) \quad \textit{and} \quad N_{\text{work}} = O\left(\frac{L^2B^2k^2 + \log(1/\delta)}{\eta^2}\right)$$

*samples for machine speed predictions and job (input) predictions respectively.*

Theorem 5.1 shows that a linear number of samples in $k$ suffices to PAC-learn machine speed predictions. The same approach yields a quadratic dependence on $k$ for job (input) predictions.

# 6 Experiments

## 6.1 Experimental setup

**Algorithms and parameters.** We evaluate the implementation of three algorithms studied in the paper: (i) **Alg 1** is a simplified version of Algorithm 1 for the dynamic machine speed prediction. (See Appendix C for implementation details), (ii) **Alg 4 - M** implements Algorithm 2 for the offline machine speed prediction model, and (iii) **Alg 4 - J** for the dynamic job speed prediction model. For all three, we use OA (25) as our ONLINE subroutine. In this section, we fix the power parameter $\alpha = 3$. Additional experiments with varying $\lambda$ values are provided in Appendix C.

**Dataset generation.** We test our algorithms on both synthetic and real datasets. Our synthetic dataset consists of instances with $n = 100$ jobs over time horizon $T$ with equally spaced release times and deadlines chosen uniformly random from $\{T/2, T\}$. The work of job $j$ is generated as in Balkanski et al. (7): $w_j = M(1 - p(a))$ where $p(a)$ is drawn from a power law distribution parameterized by $a = 100$, and where $M = 500$ is the scaling parameter. This results in a large number of small jobs and a small number of very large jobs, which is a common pattern in real-world systems and tends to be challenging to schedule.

The real dataset is the Brightkite social network dataset from the SNAP database (14), which was also used by Lykouris and Vassilvitskii (23) and, for speed scaling with predictions, by Bamas et al. (8). Using 851 days of check-in data, we define jobs by aggregating the number of check-ins in each ten minute interval, each interval corresponding to a job. We assign each job a deadline chosen uniformly at random within the same day. An instance is then comprised of all the jobs for a single day. These instances combine real check-in activity with synthetic deadlines, so the evaluation concerns scheduling behavior on a derived workload rather than performance in a deployed computing system.

**Prediction generation.** In order to produce comparable predictions for the three different prediction models, we derive the different speed predictions from the same predicted instance $\hat{J}$. For synthetic data, $\hat{J}$ perturbs each job's workload independently with probability $0.2$: $\hat{w}_j = w_j + \mathcal{N}(0, W(J) \cdot \eta)$ with noise level $\eta$. This sparse perturbation model is deliberately favorable to TPE, which does not leverage the predicted information about a job unless all three parameters (release, deadline, work) are exactly correct and otherwise falls back to a blackbox online algorithm (in this case OA). Perturbing a larger fraction of jobs would degrade TPE towards OA, thus we adopt this sparse model so TPE remains a meaningful benchmark. For real data, we use the preceding day's trace as a naive prediction. This previous-day predictor is a data-derived heuristic. Given $\hat{J}$, we derive our two prediction models: $\hat{s} = \text{OPT}(\hat{J})$ for machine speed models, and $\hat{\ell}_j$ equal to job $j$'s speed in $\text{OPT}(\hat{J})$ for the job speed model.

**Benchmarks.** We compare against several benchmarks. The first is the offline optimal algorithm (OPT), which assumes all information is known at the time $t = 0$. To compare against the scenario where no prior information is known, we use Optimal Available algorithm (OA), the online algorithm without predictions with the best-known competitive ratio (9). In contrast, MAKEFEASM from Section 4 follows the predictions while enforcing feasibility, serving as an indicator of prediction quality. Finally, we compare against TPE (7), which receives the full predicted instance $\hat{J}$ at $t = 0$. It first runs the online subroutine until the offline cost of released jobs exceeds a trust threshold; it then combines an offline schedule for exactly predicted jobs arriving after the switch with the online subroutine for the remaining jobs. With no exact matches, it reduces to the online subroutine, OA in our experiments. The guarantees of TPE-S, which tolerates deviations in job parameters, require an additional smoothness assumption on the quality-of-service cost that fails for hard deadlines (7, Section 6). We therefore use TPE, whose guarantees cover this setting.

### 6.2 Results

**Synthetic instances.** The synthetic experiment is designed to isolate the effect of prediction quality on algorithm performance. We set $\lambda = 0.005$. Figure 1 shows the resulting average energy cost over increasing prediction perturbation $\eta$ values. For small perturbation $\eta \leq 0.08$, all of our algorithms improve over OA. On the other hand, for large perturbation $\eta \geq 0.07$, they outperform MAKEFEAS. Observe that, while our algorithms are close to the better of these two benchmarks for all $\eta$, there is a range of values $[0.07, 0.08]$ for which our algorithms actually outperform *both* benchmarks. Our three algorithms, and in particular the two dynamic ones, also closely match the performance of TPE. Note that this comparison is conservative, as the sparse perturbation prediction model was chosen to keep TPE competitive, and our algorithms match it without requiring the full instance prediction at $t = 0$.

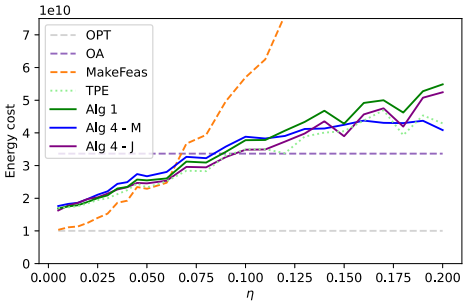


Figure 1: Average energy cost, over 20 instances, of our algorithms against OPT, OA, MAKEFEAS, and TPE as a function of prediction perturbation $\eta$. Note that MakeFeas continues to increase, hitting 2e11 at $\eta = 0.2$.

As $\eta$ grows, observe the gradual performance degradation of our three algorithms. The two dynamic prediction algorithms (Alg 1 and Alg 4 - J) maintain very similar performance. We also ran experiments for very large $\eta$ (up to 2), and while the cost of MakeFeas continues to grow linearly (incurring energy cost up to 12 times that of OA by $\eta = 0.4$), our algorithms' energy cost never exceeds 6.5e10, demonstrating their robustness.

Table 2: Energy cost statistics (scaled by 1e-6) over the full Brightkite dataset.

| Algorithm | **avg** | min | max | stdev |
|---|---|---|---|---|
| OPT | **9.56** | 0.08 | 46.75 | 9.11 |
| *Alg 1* | **15.79** | 0.13 | 195.63 | 15.84 |
| *Alg 4 – M* | **17.05** | 0.14 | 346.42 | 19.23 |
| *Alg 4 – J* | **15.09** | 0.13 | 307.98 | 17.13 |
| OA | **35.03** | 0.20 | 201.62 | 34.18 |
| MakeFeas | **17.95** | 0.09 | 528.19 | 29.76 |
| TPE | **35.03** | 0.20 | 201.62 | 34.17 |

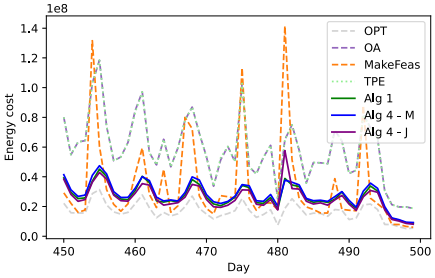


Figure 2: Energy cost of our algorithms and benchmarks on a snapshot of 50 days of the Brightkite dataset.

**Real instances.** Table 2 summarizes the performance of our algorithms and benchmarks on real data. Our three algorithms achieve a lower average energy than all three online benchmarks. This shows that we can leverage even a coarse previous-day prediction to improve over OA. TPE closely tracks OA because the previous-day predictions rarely contain exact job matches, which are difficult to obtain in reality. For comparison, in Appendix C.3 we include additional experiments that augment TPE's predictions with exact jobs with fixed probability, as in our synthetic setting. Our algorithms remain competitive despite receiving substantially less prediction information.

MakeFeas achieves a competitive average and the lowest minimum among online methods, but its maximum and standard deviation are substantially larger than ours. Among our three algorithms, Alg 4 - J has the best average performance, followed by Alg 1 and Alg 4 - M.

The per day view in (Figure 2) explains the aggregate pattern. On days where MakeFeas spikes, our algorithms remain stable, demonstrating empirical robustness. On stretches where the predictions are accurate, our algorithms track MAKEFEAS closely, demonstrating empirical smoothness. This asymmetry is what drives the average behavior: MakeFeas is closer to OPT on typical days, but the costly peaks dominate its mean, while our algorithms sacrifice a small amount on accurate days to bound the damage on inaccurate ones.

## Acknowledgments

Eric Balkanski and Cherlin Zhu were supported by NSF grants CCF-2210501 and IIS-2147361. Cliff Stein and Jingwei Li were supported in part by NSF grant CCF-2218677 and ONR grant ONR-13533312. Cliff Stein was also supported partly by the Wai T. Chang Chair in Industrial Engineering and Operations Research. The authors declare no competing interests.

## References

[1] Priyank Agrawal, Eric Balkanski, Vasilis Gkatzelis, Tingting Ou, and Xizhi Tan. Learning-augmented mechanism design: Leveraging predictions for facility location. In *Proceedings of the 23rd ACM Conference on Economics and Computation*, pages 497–528, 2022.

[2] Susanne Albers and Hiroshi Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms (TALG)*, 3(4):49–es, 2007.

[3] Lachlan LH Andrew, Adam Wierman, and Ao Tang. Optimal speed scaling under arbitrary power functions. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

[4] Antonios Antoniadis, Peyman Jabbarzade, and Golnoosh Shahkarami. A novel prediction setup for online speed-scaling. In *18th Scandinavian Symposium and Workshops on Algorithm Theory*, 2022.

[5] Antonios Antoniadis, Christian Coester, Marek Eliáš, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. *ACM transactions on algorithms*, 19(2):1–34, 2023.

[6] Yossi Azar, Shahar Lewkowicz, and Varun Suriyanarayana. List update with prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 15436–15444, 2025.

[7] Eric Balkanski, Noemie Perivier, Clifford Stein, and Hao-Ting Wei. Energy-efficient scheduling with predictions. In *Advances in Neural Information Processing Systems*, volume 36, pages 79012–79023, 2023.

[8] Étienne Bamas, Andreas Maggiori, Lars Rohwedder, and Ola Svensson. Learning augmented energy minimization via speed scaling. *Advances in Neural Information Processing Systems*, 33:15350–15359, 2020.

[9] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1):1–39, 2007.

[10] Nikhil Bansal, Ho-Leung Chan, and Kirk Pruhs. Speed scaling with an arbitrary power function. In *Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms*, pages 693–701. SIAM, 2009.

[11] Nikhil Bansal, Ho-Leung Chan, Kirk Pruhs, and Dmitriy Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. In *International Colloquium on Automata, Languages, and Programming*, pages 144–155. Springer, 2009.

[12] Nikhil Bansal, Kirk Pruhs, and Cliff Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2010.

[13] Peter L. Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: risk bounds and structural results. *J. Mach. Learn. Res.*, 3(null):463–482, March 2003. ISSN 1532-4435.

[14] Eunjoon Cho, Seth A Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1082–1090, 2011.

[15] Nicolas Christianson, Junxuan Shen, and Adam Wierman. Optimal robustness-consistency tradeoffs for learning-augmented metrical task systems. In *International Conference on Artificial Intelligence and Statistics*, pages 9377–9399. PMLR, 2023.

[16] George Christodoulou, Alkmini Sgouritsa, and Ioannis Vlachos. Mechanism design augmented with output advice. *Advances in Neural Information Processing Systems*, 37:41934–41953, 2024.

[17] R.M Dudley. The sizes of compact subsets of hilbert space and continuity of gaussian processes. *Journal of Functional Analysis*, 1(3):290–330, 1967. ISSN 0022-1236. doi: https://doi.org/10.1016/0022-1236(67)90017-1. URL https://www.sciencedirect.com/science/article/pii/0022123667900171.

[18] Vasilis Gkatzelis, Daniel Schoepflin, and Xizhi Tan. Clock auctions augmented with unreliable advice. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2629–2655. SIAM, 2025.

[19] IEA. Electricity 2024, analysis and forecast to 2026. https://www.iea.org/reports/electricity-2024, 2024. [Accessed 27-01-2026].

[20] Sandy Irani, Sandeep Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms (TALG)*, 3(4):41–es, 2007.

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

[22] Alexander Lindermayr, Nicole Megow, and Martin Rapp. Speed-oblivious online scheduling: Knowing (Precise) speeds is not necessary. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 21312–21334. PMLR, 2023. URL https://proceedings.mlr.press/v202/lindermayr23a.html.

[23] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *Journal of the ACM (JACM)*, 68(4):1–25, 2021.

[24] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. Allocating online advertisement space with unreliable estimates. In *Proceedings of the 8th ACM conference on Electronic commerce*, pages 288–294, 2007.

[25] Frances Yao, Alan Demers, and Scott Shenker. A scheduling model for reduced cpu energy. In *Proceedings of IEEE 36th annual foundations of computer science*, pages 374–382. IEEE, 1995.

# A Additional Preliminaries and Missing Proofs from Section 3

## A.1 Additional details for the preliminaries

**Implementing fractional processing.** With cost-free preemption, fractional processing can be viewed as time-sharing over arbitrarily short intervals, split at job releases and deadlines. Within each interval $I$, run the jobs in turn at the aggregate speed $s(t)$, giving job $j$ the amount of work $\int_I s_j(t)\,dt$. These allocations fit in $I$ because their sum is $\int_I s(t)\,dt$, so time-sharing respects each job's release time and deadline and preserves the energy cost, while executing only one job at a time. For constant rates with $s > 0$, this amounts to assigning job $j$ a fraction $s_j/s$ of the interval at speed $s$.

**Optimality of the two-job example.** In the example in Section 2, every feasible schedule must process four units of work within $[0, 4]$. Convexity gives

$$\int_0^4 s(t)^\alpha\,dt \geq 4\left(\frac{1}{4}\int_0^4 s(t)\,dt\right)^\alpha \geq 4.$$

The schedule described there attains this bound and is therefore optimal.

## A.2 Subroutines for the dynamic machine speed algorithm

We provide the pseudocode for the ASSIGN and REASSIGN subroutines used by Algorithm 1.

**Algorithm 4:** ASSIGN

**Input:** job $j$, schedule $s_{\mathrm{F}}$, $\lambda \in [0, 1]$, jobs $J^{\mathrm{on}}$ and $J^{\mathrm{p}}$
**if** $\mathrm{cost}(\mathrm{OPT}(J^{\mathrm{on}} \cup \{j\})) \leq \lambda \cdot \mathrm{cost}(s_{\mathrm{F}})$ **then** add $j$ to $J^{\mathrm{on}}$
**else** add $j$ to $J^{\mathrm{p}}$
**return** $J^{\mathrm{on}}$, $J^{\mathrm{p}}$

**Algorithm 5:** REASSIGN

**Input:** schedule $s_{\mathrm{F}}$, jobs $J^{\mathrm{on}}$ and $J^{\mathrm{p}}$, time $t$
Let $j$ be the job that $s_{\mathrm{F}}$ runs at time $t$
**if** $j \in J^{\mathrm{p}}$ **then**
    Let $w^{\mathrm{c}}$ be the work completed by time $t$ for $j$
    Define $r_{j'} = r_j$, $d_{j'} = d_j$, $w_{j'} = w^{\mathrm{c}}$ ; define $r_{j''} = t$, $d_{j''} = d_j$, $w_{j''} = w_j - w^{\mathrm{c}}$
    Replace $j$ with $j'$ in $J^{\mathrm{p}}$, add $j''$ to $J^{\mathrm{on}}$
**return** $J^{\mathrm{on}}$, $J^{\mathrm{p}}$

## A.3 Proof of Lemma 3.1

**Lemma 3.1.** *Let $t^\star$ be the earliest time at which $\Sigma = \emptyset$. For any $J$, dynamic machine speed prediction $\hat{s}$, and $t < t^\star$, cost*$(\mathrm{OPT}(J_t^{on} \cup J_t^{p})) =$ *cost*$(\mathrm{OPT}(J_{\leq t}))$.

*Proof.* For a machine speed schedule $s$ let $w_j^c(s, \tau)$ denote the amount of work completed by $s$ on job $j$ by time $\tau$. Observe that since $t < t^*$ and by definition of $\Sigma$ in Line 4, there exists $J'$ such that $J'_{\leq t} = J_{\leq t}$ and $s_{\mathrm{F}} = \mathrm{OPT}(J')$. Thus we have that $\mathrm{OPT}(J_{\leq t}) \leq s_F$. This implies, for any $j \in J_{\leq t}$ and time $\tau$, that

$$w_j^c(s_{\mathrm{F}}, \tau) \geq w_j^c(\mathrm{OPT}(J_{\leq t}), \tau). \tag{2}$$

We can then construct a schedule for $J_t^{\mathrm{on}} \cup J_t^{\mathrm{p}}$ from $\mathrm{OPT}(J_{\leq t})$. For any $j \in J_{\leq t}$, exactly one case holds: (1) $j \in J_{\leq t}$ was never reassigned, and (2) $j \in J_{\leq t}$ was reassigned at time $\tau_j \geq r_j$.

For case 1, the job can be scheduled exactly as in $\mathrm{OPT}(J_{\leq t})$. For case 2, observe that $j$ is replaced in Algorithm 5 by job $j'$ in $J^{\mathrm{p}}$ and $j''$ in $J^{\mathrm{on}}$. Let $w_j^c = w_j^c(s_{\mathrm{F}}, \tau_j)$, and by Equation 2 we have $w_j^c \geq w_j^c(\mathrm{OPT}(J_{\leq t}), \tau_j)$. This means that $\mathrm{OPT}(J_{\leq t})$ schedules at least $w_j - w_j^c$ units of work on $j$ within $[\tau_j, d_j]$. Then we can feasibly schedule $j''$ with its $w_j - w_j^c$ units of work (Line 5 in

REASSIGN) at times $T \subseteq [\tau_j, d_j]$, which is within the release and deadline times of $j''$, at the same speed that $\text{OPT}(J_{\leq t})$ runs $j$ on $T$.

Then $\text{OPT}(J_{\leq t})$ runs $w_j^c$ units of work within $[r_j, d_j] \backslash T$, and we schedule job $j'$ at these times and at the same speeds. This is feasible for $j'$ by definition of $r_{j'} = r_j, d_{j'} = d_j, w_{j'} = w_j^c$ (Line 5). This schedule is feasible for all jobs in $J_{\leq t}$ and has cost equal to $\text{cost}(\text{OPT}(J_{\leq t}))$. □

### A.4 Proof of Lemma 3.2

We now need to bound $\text{cost}(J^{\text{on}})$ in terms of $\lambda$ and the cost of the prediction in the consistency case.

**Lemma A.1.** *For any instance $J$ and speed prediction $\hat{s} = \text{OPT}(J)$, we have that cost$(\text{OPT}(J^{on})) \leq \lambda \cdot$ cost$(\hat{s})$.*

*Proof.* First note that $t^\star = \infty$, as $\hat{s} \in \Sigma$ holds at all $t$.

We will show that $\text{cost}(\text{OPT}(J_t^{\text{on}})) \leq \lambda \cdot \text{cost}(\hat{s})$ holds for all time $t$ by induction. For the base case of $t = 0$, this holds because $J_0^{\text{on}} = \emptyset$. Assume then that the hypothesis holds for $t'$. Observe that $\text{cost}(\text{OPT}(J_t^{\text{on}}))$ only increases from $\text{cost}(\text{OPT}(J_{t'}^{\text{on}}))$ in two ways: (1) when jobs get added to $J_t^{\text{on}}$ in Line 7 with ASSIGN (Algorithm 4) and (2) when jobs are reassigned to $J_t^{\text{on}}$ in Line 9 with REASSIGN (Algorithm 5). Then let $t > t'$ be the first time one of these occurs.

In the first case, let $j$ be the job released at $t$. Observe that $j$ gets assigned to $J^{\text{on}}$ by ASSIGN if the condition in Line 2 holds, so $\text{cost}(\text{OPT}(J_{t'}^{\text{on}} \cup \{j\})) \leq \lambda \cdot \text{cost}(s_{\text{F}})$. Then we get the following:

$$\text{cost}(J_t^{\text{on}}) \leq \lambda \cdot \text{cost}(s_{\text{F}}) \leq \lambda \cdot \text{cost}(\hat{s}),$$

where the last inequality is because $\hat{s} \in \Sigma$ and $s_{\text{F}}$ is by definition the minimum cost schedule in $\Sigma$.

In the second case, first observe that $\text{cost}(\text{OPT}(J_t^{\text{on}})) \leq \text{cost}(J_{\leq t})$ by Lemma 3.1 and that $t < t^*$. We also have that REASSIGN is only called when the condition in Line 8 in Algorithm 1 fails, meaning $\text{cost}(\text{OPT}(J_{\leq t})) < \lambda \cdot \text{cost}(s_{\text{F}})$. Thus, we have that $\text{cost}(J_t^{\text{on}}) \leq \lambda \cdot \text{cost}(s_{\text{F}}) \leq \lambda \cdot \text{cost}(\hat{s})$ as desired.

Thus, the induction hypothesis holds for all $t$. □

Finally, we need the following fact about combining jobs.

**Observation A.2.** *For any set of jobs $J$ and $J'$, it holds that*

$$\textit{cost}(\text{OPT}(J \cup J')) \leq 2^\alpha \cdot (\textit{cost}(\text{OPT}(J)) + \textit{cost}(\text{OPT}(J'))).$$

*Proof.* Let $s = \text{OPT}(J \cup J')$, $s_J = \text{OPT}(J)$, and $s_{J'} = \text{OPT}(J')$. Note that $s(t) \leq \max\{2s_J(t), 2s_{J'}(t)\}$ for any $t$. Thus, the result follows directly from these computations:

$$\text{cost}(\text{OPT}(J \cup J')) = \int_0^\infty s(t)^\alpha dt \leq \int_0^\infty \max\{(2s_J(t))^\alpha, (2s_{J'}(t))^\alpha\} dt$$
$$\leq 2^\alpha \int_0^\infty [s_J(t)^\alpha + s_{J'}(t)^\alpha] dt = 2^\alpha \cdot (\text{cost}(\text{OPT}(J)) + \text{cost}(\text{OPT}(J'))).$$ □

Now we can complete the proof of Lemma 3.2.

**Lemma 3.2.** *For any $\lambda \in (0, 1)$, jobs $J$, dynamic machine speed prediction $\hat{s}$, and $c_{on}$-competitive algorithm* ONLINE*, cost$(\text{ALG}(J, \hat{s})[J^{on}]) \leq 2^{\alpha+1} \cdot c_{on} \cdot$ cost$(\text{OPT}(J))$. If $\hat{s} = \text{OPT}(J)$, cost$(\text{ALG}(J, \hat{s})[J^{on}]) \leq c_{on} \cdot \lambda \cdot$ cost$(\text{OPT}(J))$.*

*Proof.* Let us first assume that $t^* < \infty$. Note that this is always the case if $\hat{s} \neq \text{OPT}(J)$ because $\text{OPT}(J)$ is unique. Then let $t$ be the the last time before $t^*$ where $J^{\text{on}}$ changed. We have that

$$\begin{aligned}\text{cost}(\text{ALG}[J^{\text{on}}]) &= \text{cost}(\text{ONLINE}(J_t^{\text{on}} \cup (J^{\text{on}} \backslash J_t^{\text{on}})) \\ &\leq c_{\text{on}} \cdot \text{cost}(J_t^{\text{on}} \cup (J^{\text{on}} \backslash J_t^{\text{on}})) \\ &\leq c_{\text{on}} \cdot 2^\alpha (\text{cost}(\text{OPT}(J_t^{\text{on}})) + \text{cost}(\text{OPT}(J^{\text{on}} \backslash J_t^{\text{on}})))\end{aligned}$$

where the second line is by the competitive ratio of ONLINE and the last line is by Observation A.2. We know that $\text{cost}(\text{OPT}(J^{\text{on}}_t)) \le \text{cost}(\text{OPT}(J_{\le t})) \le \text{cost}(\text{OPT}(J))$ by Lemma 3.1 and that $\text{cost}(\text{OPT}(J^{\text{on}} \setminus J^{\text{on}}_t)) \le \text{cost}(\text{OPT}(J))$ by $(J^{\text{on}} \setminus J^{\text{on}}_t) \subseteq J$. Thus,

$$\text{cost}(\text{ALG}[J^{\text{on}}]) \le c_{\text{on}} \cdot 2^{\alpha+1} \cdot \text{cost}(\text{OPT}(J)).$$

For the case of $\hat{s} = \text{OPT}(J)$, we get that $\text{cost}(\text{ALG}[J^{\text{on}}]) = \text{cost}(\text{ONLINE}(J^{\text{on}})) \le c_{\text{on}} \cdot \text{cost}(\text{OPT}(J^{\text{on}}))$. Applying Lemma A.1, this is at most $c_{\text{on}} \cdot \lambda \cdot \text{cost}(\hat{s})$, which by definition of $\hat{s}$ equals $c_{\text{on}} \cdot \lambda \cdot \text{cost}(\text{OPT}(J))$. □

### A.5 Proof of Lemma 3.3

We now bound $\text{cost}(\text{ALG}[J^{\text{p}}])$. Let us first make the following observation about the monotonicity of $s_{\text{F}}$ in time.

**Observation A.3.** *Fix an instance $J$ and prediction $\hat{s}$. For any $t' \le t$ with $\Sigma_t \neq \emptyset$, let $s^{t'}_F$ and $s^t_F$ be the corresponding forecast schedules. Then $s^{t'}_F \le s^t_F$.*

This follows from Corollary E.6. Once $\Sigma_t$ becomes empty, Algorithm 1 keeps its last forecast schedule. We use this observation to prove Lemma 3.3.

**Lemma 3.3.** *For any $\lambda \in (0,1)$, $J$, $\hat{s}$, $cost(\text{ALG}(J,\hat{s})[J^p]) \le \min\{cost(\hat{s}), cost(\text{OPT}(J))/\lambda\}$.*

*Proof.* Take $s_{\text{R}}$ as defined at the end of the algorithm. Let us define $\mathcal{S}|_J$ as the schedule $\mathcal{S}$ restricted to jobs in $J$. Observe that $\text{cost}(\text{ALG}[J^{\text{p}}]) \le \text{cost}(\text{EDF}(J_{\le t}, s_{\text{R}})|_{J^{\text{p}}})$ at any time $t$ because $s_{\text{R}}$ is pointwise monotone in $t$ by Observation A.3 and the fact that $s_{\text{R}}$ is equal to some $s_{\text{F}}$. Then we will show by induction on time that

$$\lambda \cdot \text{cost}(\text{EDF}(J_{\le t}, s_{\text{R}})|_{J^{\text{p}}}) \le \text{cost}(\text{OPT}(J_{\le t})).$$

At the base case of $t = 0$, this holds because $J_{\le 0} = \emptyset$ and $s_{\text{R}}(t) = 0$ for all $t$, and therefore both sides are equal to 0. Observe that $\text{cost}(\text{OPT}(J_{\le t}))$ is nondecreasing, so it suffices to show that the claim holds at every $t$ where $\text{cost}(\text{EDF}(J_{\le t}, s_{\text{R}})|_{J^{\text{p}}})$ increases. This only occurs when $s_{\text{R}}$ increases or jobs get added to $J^{\text{p}}$. Assume the claim holds for $t'$ and let $t > t'$ be the first time one of these conditions is met.

Let us first consider $t < t^*$. The first case is that $s_{\text{R}}$ increases. This only occurs in Line 8 when $s_R$ is updated to $s_F$, which implies that the condition $\lambda \cdot \text{cost}(s_{\text{F}}) \le \text{cost}(\text{OPT}(J_{\le t}))$ in Line 8 is met. Thus,

$$\lambda \cdot \text{cost}(\text{EDF}(J_{\le t}, s_{\text{R}})|_{J^{\text{p}}}) \le \lambda \cdot \text{cost}(s_{\text{R}}) = \lambda \cdot \text{cost}(s_{\text{F}}) \le \text{cost}(\text{OPT}(J_{\le t})).$$

The second case is that some job $j$ is added to $J^{\text{p}}$ at $t$. This only happens when ASSIGN is called in Line 7. For $j$ to be assigned to $J^{\text{p}}$, the condition in Line 2 of ASSIGN must fail, meaning $\lambda \cdot \text{cost}(s_{\text{F}}) < \text{cost}(\text{OPT}(J^{\text{on}}_{\le t'} \cup \{j\})))$. Thus we have that

$$\begin{aligned}\lambda \cdot \text{cost}(\text{EDF}(J_{\le t}, s_{\text{R}})|_{J^{\text{p}}}) &\le \lambda \cdot \text{cost}(s_{\text{R}}) \le_{(1)} \lambda \cdot \text{cost}(s_{\text{F}}) \\ &< \text{cost}(\text{OPT}(J^{\text{on}}_{\le t'} \cup \{j\})) \le_{(2)} \text{cost}(\text{OPT}(J_{\le t})),\end{aligned}$$

where (1) is by the fact that $s_{\text{R}}$ is equal to some $s_{\text{F}}$ defined at some $t' \le t$ and Observation A.3 and (2) is by Lemma 3.1. Then the induction hypothesis also holds in this case.

For the case of $t \ge t^*$, observe that $s_{\text{R}}$ and $J^{\text{p}}$ do not change after $t^*$, therefore the claim holds for all $t$. □

### A.6 Proof of Theorem 3.4

We use the following result by Balkanski et al. (7) to bound the cost of simultaneously running schedules $\text{ALG}(J,\hat{s})[J^{\text{on}}]$ and $\text{ALG}(J,\hat{s})[J^{\text{p}}]$. We let $s_1 + s_2$ be the machine speed schedule that runs at speed $s_1(t) + s_2(t)$ at time $t$.

**Lemma A.4** ((7))**.** *For any machine speed schedule $s_1$ and machine speed schedule $s_2$, we have $cost(s_1 + s_2) \le (cost(s_1)^{1/\alpha} + cost(s_2)^{1/\alpha})^\alpha$.*

Then we can prove the main theorem.

**Theorem 3.4.** *For any $\epsilon \in (0,1)$, Algorithm 1 with $\lambda = \frac{1}{c_{on}}\left(\frac{\varepsilon}{2^\alpha}\right)^\alpha$ is $(1+\varepsilon)$-consistent and $c_{on} \cdot (2^{\frac{\alpha+1}{\alpha}} + \frac{2^\alpha}{\varepsilon})^\alpha$ robust, where $c_{on}$ is the competitive ratio of* ONLINE. *Furthermore, Algorithm 1 is implementable with runtime polynomial in $n$.*

*Proof.* Let us fix our parameter $\lambda \in (0,1)$. We will first provide bounds on consistency and robustness that depend on $\lambda$. Fix instance $J$ and predicted speed schedule $\hat{s}$. Let $J^{\text{on}}$ and $J^{\text{off}}$ be as defined in the algorithm when all jobs are finished.

Observe that cost of our algorithm is equal to $\text{cost}(\text{ALG}[J^{\text{on}}] + \text{ALG}[J^{\text{p}}])$. Apply Lemma A.4 to bound this above by

$$(\text{cost}(\text{ALG}[J^{\text{on}}])^{1/\alpha} + \text{cost}(\text{ALG}[J^{\text{p}}])^{1/\alpha})^\alpha.$$

For the consistency case, assume that $\hat{s} = \text{OPT}(J)$. Then by Lemma 3.2 and Lemma 3.3, the cost of the algorithm is bound above by

$$((c_{\text{on}} \cdot \lambda \cdot \text{cost}(\text{OPT}(J)))^{1/\alpha} + \text{cost}(\hat{s})^{1/\alpha})^\alpha \le (1 + c_{\text{on}}^{1/\alpha} \cdot \lambda^{1/\alpha})^\alpha \cdot \text{cost}(\text{OPT}(J)).$$

We now show the robustness case. Observe that $\text{cost}(\text{ALG}[J^{\text{on}}]) \le 2^{\alpha+1} \cdot c_{\text{on}} \cdot \text{cost}(\text{OPT}(J))$ by Lemma 3.2. Also note that $\text{cost}(\text{ALG}[J^{\text{p}}]) \le 1/\lambda \cdot \text{cost}(\text{OPT}(J))$ by Lemma 3.3. Thus, the cost of the algorithm is bound by

$$\begin{aligned}&((2^{\alpha+1} \cdot c_{\text{on}} \cdot \text{cost}(\text{OPT}(J)))^{1/\alpha} + 1/\lambda \cdot \text{cost}(\text{OPT}(J))^{1/\alpha})^\alpha \\ &\le ((2^{\alpha+1} \cdot c_{\text{on}})^{1/\alpha} + (1/\lambda^{1/\alpha}))^\alpha \cdot \text{cost}(\text{OPT}(J)).\end{aligned}$$

It suffices then to set $\lambda = \frac{1}{c_{\text{on}}}\left(\frac{\varepsilon}{2^\alpha}\right)^\alpha$. Thus, we get consistency

$$\begin{aligned}(1 + (c_{\text{on}} \cdot \lambda)^{1/\alpha})^\alpha &\le 1 + 2^\alpha (c_{\text{on}} \cdot \lambda)^{1/\alpha} \\ &= 1 + 2^\alpha \left(c_{\text{on}} \cdot \frac{1}{c_{\text{on}}}\left(\frac{\varepsilon}{2^\alpha}\right)^\alpha\right)^{1/\alpha} \\ &= 1 + \varepsilon,\end{aligned}$$

where the inequality is by the convex of $f(u) = (1+u)^\alpha$. To see this, observe that for $u \in [0,1]$, we have that $f(u) \le (1-u) \cdot f(0) + u \cdot f(1) = (1-u) + 2^\alpha u \le 1 + 2^\alpha u$. We can then set $u = (c_{\text{on}} \cdot \lambda)^{1/\alpha}$ to get the desired inequality.

For robustness, plugging $\lambda$ in gives us $c_{\text{on}} \cdot (2^{\frac{\alpha+1}{\alpha}} + \frac{2^\alpha}{\varepsilon})^\alpha$- robustness. □

# B Missing Results and Proofs from Section 4

## B.1 Offline machine speed predictions

We prove Theorem 4.1 for offline machine speed predictions (Table 1, second row), starting with monotonicity of MAKEFEASM.

**Lemma B.1.** *For any offline machine speed prediction $\hat{s}$ and jobs $J' \subseteq J$, we have*

$$\textit{cost}(\text{MAKEFEASM}(J', \hat{s})) \le \textit{cost}(\text{MAKEFEASM}(J, \hat{s})).$$

*Proof.* It suffices to prove the lemma when $J = J' \cup \{j\}$ differs from $J'$ by a single arbitrary job $j$, and the general case will follow inductively. Now we fix $J = J' \cup \{j\}$ and let $s := \text{MAKEFEASM}(J, \hat{s})$ and $s' := \text{MAKEFEASM}(J', \hat{s})$.

For any schedule produced by MAKEFEASM, consider any time interval on which (i) there is a set of available unfinished jobs with the earliest common deadline $d$, and (ii) no job with deadline $< d$ is released during this interval. On such an interval, MAKEFEASM always executes jobs of deadline $d$ (EDF), and its speed is

$$s(t) = \begin{cases} \hat{s}(t), & \text{if } \int_t^d \hat{s}(\tau)\, d\tau \ge W(t), \\ \max\{\hat{s}(t),\, W(t)/(d-t)\}, & \text{otherwise,} \end{cases}$$

where $W(t)$ is the remaining work at time $t$ of the released unfinished jobs of deadline $d$. Define the "density ratio" $\rho(t) := W(t)/(d-t)$ on this interval. The execution of deadline-$d$ jobs reduces $W(t)$ at the speed of $s(t) \geq \rho(t)$. Therefore, whenever the deadline $d$ remains the minimum, $\rho$ is nonincreasing:

$$\frac{d}{dt}\rho(t) = \frac{(d-t)\cdot \frac{d}{dt}W(t) + W(t)}{(d-t)^2} = \frac{-(d-t)\,s(t) + W(t)}{(d-t)^2} \leq 0. \qquad (\star)$$

In particular, during any such ***fixed-deadline phase*** for deadline $d$, the value of $\rho(t)$ never exceeds its value at the start of this phase.

At some time $t$, we refer to a job $j$ being among the released unfinished jobs having the minimum deadline as a ***current*** job. Let $t_0$ be the earliest time $t \geq r_j$ at which, in the run on $J$, job $j$ becomes a current job. By definition of $t_0$, on $[0, t_0)$, the extra job $j$ is either unreleased or has a deadline larger than the current minimum, so the set of current jobs is identical in the two runs $s$ and $s'$. MAKEFEASM makes identical choices, which means $s(t) = s'(t)$ for all $t \in [0, t_0)$. At time $t_0$, in the run on $J$, the current minimum deadline equals $d_j$. In the run on $J'$, the current minimum deadline at $t_0$ is some $d \geq d_j$. We discuss the following two cases.

**Case 1: $d = d_j$.** At time $t_0$, both runs $s$ and $s'$ have the same minimum deadline $d_j$ among their current jobs. Let $W(t)$ and $W'(t)$ be the remaining work at time $t$ of the current jobs with deadline $d_j$ in the runs on $J$ and $J'$ respectively. At $t = t_0$, the only difference between the two sets is the additional job $j$, hence $W(t_0) = W'(t_0) + w_j \geq W'(t_0)$. Consider the maximal time interval $I$ starting at $t_0$ that is a *fixed-deadline phase* with deadline $d_j$. On $I$, both runs execute only deadline-$d_j$ jobs. Moreover, we have for all $t \in I$ that

$$s(t) = \max\{\hat{s}(t),\, W(t)/(d_j - t)\} \;\geq\; \max\{\hat{s}(t),\, W'(t)/(d_j - t)\} = s'(t),$$

which is by the above discussion on fixed-deadline intervals. Therefore, the energy cost incurred during $I$ for $J$ is lower bounded by that for $J'$:

$$\int_I s(t)^\alpha \, dt \;\geq\; \int_I s'(t)^\alpha \, dt.$$

At the end of $I$, either (a) all currently released deadline-$d_j$ jobs finish, or (b) a job with deadline $< d_j$ is released. In either case, the two runs face identical future arrivals, and the same argument can be re-applied from the next divergence point (or otherwise we immediately conclude the argument if both runs coincide thereafter). Summing over all such phases yields $\text{cost}(s) \geq \text{cost}(s')$.

**Case 2: $d > d_j$.** Let $C$ denote the set of current jobs in the run on $J'$ at time $t_0$, with the deadline equals $d$, and let $W_C$ be their total remaining work at time $t_0$ in the run on $J'$. By EDF, the run on $J'$ will execute jobs from $C$ whenever no job with deadline $< d$ is pending; the run on $J$ cannot execute any job from $C$ while $j$ (deadline $d_j < d$) remains unfinished and is the minimum-deadline job.

Let $t_1 \geq t_0$ be the time at which the run on $J$ first enters a phase, after $t_0$, whose minimum deadline equals $d$. By EDF, the run on $J$ does not process any job from $C$ on $[t_0, t_1)$. Therefore, at time $t_1$ the remaining work of $C$ in the run on $J$ is still $W_C$, whereas in the run on $J'$ it is some $W'_C(t_1) \leq W_C$, since it has been processing jobs in $C$. Now consider the maximal fixed-deadline-$d$ phase $T$ of the run on $J$ starting at $t_1$, and the corresponding fixed-deadline-$d$ phase $T'$ of the run on $J'$ that contains time $t_1$ (possibly starting earlier). On these phases, we have discussed that the density threshold used for deadline $d$ is controlled by the phase-start densities. Since at the start of $T$ the run on $J$ has remaining work $W_C$ for deadline $d$ jobs, while at the start of $T'$ the run on $J'$ has remaining work at most $W_C$, we get that the density thresholds (and thus speeds) satisfy

$$s(t) \;\geq\; s'(t) \qquad \forall t \in T \cap T'.$$

Thus, any potential advantage in energy cost of the run on $J$ can only come from times before $t_1$, when it needed to process $j$ while the run on $J'$ was already processing $C$. However, such an advantage will be compensated for by the higher energy cost later on. Formally, by the argument above, once the run on $J$ begins servicing deadline-$d$ jobs at $t_1$ and onwards, its speed is pointwise at least the speed of the run on $J'$ on the corresponding deadline-$d$ phases, and by convexity, since the

run on $J$ completes at least the same amount of total work, it has higher energy cost. Note that both runs will also exhibit identical behavior on any middle phases with a minimum deadline $< d$, since those jobs will be common to both runs.

Therefore, summing costs phase-by-phase up through the completion of all jobs with deadline at most $d$ yields

$$\int_{t_0}^{\infty} s(t)^\alpha \, dt \;\geq\; \int_{t_0}^{\infty} s'(t)^\alpha \, dt.$$

Together with the equality on $[0, t_0)$, this implies $\text{cost}(s) \geq \text{cost}(s')$.

In conclusion, in both cases, adding one job cannot decrease the energy cost of MAKEFEASM. By the initial reduction, the claim follows for all $J' \subseteq J$. □

We now provide smoothness and robustness guarantees for our prediction models and feasibility projection, starting with smoothness. Recall that $c_{\text{on}}$ is the competitive ratio of the online algorithm ONLINE without predictions.

**Lemma B.2.** *Algorithm 2 with* MAKEFEAS = MAKEFEASM *is* $((c_{on}\lambda)^{1/\alpha} + 1)^\alpha$*-smooth with respect to* MAKEFEASM.

*Proof.* Consider an instance $J$ and an offline machine speed prediction $\hat{s}$. At time $t$, Line 5 of Algorithm 2 only adds a job $j$ to $J^{\text{on}}$ if $\text{cost}(\text{OPT}(J^{\text{on}} \cup \{j\})) \leq \lambda \cdot \text{cost}(\text{MAKEFEASM}(J_{\leq t}, \hat{s}))$. Combined with the fact that $\text{cost}(\text{MAKEFEASM}(J_{\leq t}, \hat{s}))$ is monotonically increasing with respect to $t$ by Lemma B.1, we have that $\text{cost}(\text{OPT}(J^{\text{on}})) \leq \lambda \cdot \text{cost}(\text{MAKEFEASM}(J, \hat{s}))$ when the algorithm terminates. Thus, since ONLINE is a $c_{\text{on}}$-competitive algorithm, we have that upon termination, $\text{cost}(\text{ONLINE}(J^{\text{on}})) \leq c_{\text{on}} \cdot \lambda \cdot \text{cost}(\text{MAKEFEAS}(J, P))$. We get

$$\begin{aligned}
\text{cost}(\text{ALG}(J, \hat{s})) &=_{(1)} \text{cost}(\text{ONLINE}(J^{\text{on}}) + \text{MAKEFEAS}(J^{\text{p}}, \hat{s})) \\
&\leq_{(2)} (\text{cost}(\text{ONLINE}(J^{\text{on}}))^{\frac{1}{\alpha}} + \text{cost}(\text{MAKEFEAS}(J^{\text{p}}, \hat{s}))^{\frac{1}{\alpha}})^\alpha \\
&\leq_{(3)} ((c_{\text{on}}\lambda)^{\frac{1}{\alpha}} \text{cost}(\text{MAKEFEAS}(J, \hat{s}))^{\frac{1}{\alpha}} + \text{cost}(\text{MAKEFEAS}(J^{\text{p}}, \hat{s}))^{\frac{1}{\alpha}})^\alpha \\
&\leq_{(4)} ((c_{\text{on}}\lambda)^{\frac{1}{\alpha}} \text{cost}(\text{MAKEFEAS}(J, \hat{s}))^{\frac{1}{\alpha}} + \text{cost}(\text{MAKEFEAS}(J, \hat{s}))^{\frac{1}{\alpha}})^\alpha \\
&\leq ((c_{\text{on}}\lambda)^{1/\alpha} + 1)^\alpha \cdot \text{cost}(\text{MAKEFEAS}(J, \hat{s})),
\end{aligned}$$

where (1) is by definition of Algorithm 2 and since ONLINE and MAKEFEAS are online algorithms, (2) by Lemma A.4, (3) by the argument above, and (4) is by Lemma B.1. □

Next, we show the robustness bound.

**Lemma B.3.** *Algorithm 2 with* MAKEFEAS = MAKEFEASM *is* $(c_{on}^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}})^\alpha$*-robust.*

*Proof.* Let $t^\star$ be the last time Algorithm 2 adds a job to $J^{\text{p}}$. Let $j^\star$ be this last job added to $J^{\text{p}}$. By Line 5 of Algorithm 2,

$$\text{cost}(\text{OPT}(J^{\text{on}} \cup \{j^\star\})) > \lambda \cdot \text{cost}(\text{MAKEFEAS}(J_{\leq t^\star}, P_{\leq t^\star})),$$

which implies

$$\text{cost}(\text{MAKEFEAS}(J_{\leq t^\star}, P_{\leq t^\star})) < \text{cost}(\text{OPT}(J)) \cdot \frac{1}{\lambda}.$$

The cost of Algorithm 2 is then bounded as follows:

$$\begin{aligned}
\text{cost}(\text{ALG}(J, \hat{s})) &=_{(1)} \text{cost}(\text{ONLINE}(J^{\text{on}}) + \text{cost}(\text{MAKEFEAS}(J^{\text{p}}, \hat{s}))) \\
&\leq_{(2)} (\text{cost}(\text{ONLINE}(J^{\text{on}}))^{1/\alpha} + \text{cost}(\text{MAKEFEAS}(J^{\text{p}}, \hat{s}))^{1/\alpha})^\alpha \\
&\leq_{(3)} (\text{cost}(\text{ONLINE}(J^{\text{on}}))^{1/\alpha} + \text{cost}(\text{MAKEFEAS}(J_{\leq t^\star}, \hat{s}))^{1/\alpha})^\alpha \\
&\leq_{(4)} (c_{\text{on}}^{1/\alpha} \cdot \text{cost}(\text{OPT})^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}} \cdot \text{cost}(\text{OPT})^{1/\alpha})^\alpha \\
&\leq (c_{\text{on}}^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}})^\alpha \text{cost}(\text{OPT}),
\end{aligned}$$

where (1) is by definition of Algorithm 2 and since ONLINE and MAKEFEAS are online algorithms, (2) by Lemma A.4, (3) by Lemma B.1, and (4) by the argument above and the competitive ratio for the online algorithm. □

We can then present our main result for this section, which follows directly from Lemma B.2 and Lemma B.3.

**Theorem 4.1.** *Fix any $\varepsilon \in (0,1)$. In the offline machine speed prediction model, Algorithm 2 with* MAKEFEAS $=$ MAKEFEASM*, a $c_{on}$-competitive algorithm* ONLINE*, and $\lambda = \frac{1}{c_{on}}\left(\frac{\varepsilon}{2^\alpha}\right)^\alpha$ achieves $(1+\varepsilon)$-smoothness w.r.t.* MAKEFEASM *and $c_{on} \cdot (1+\frac{2^\alpha}{\varepsilon})^\alpha$-robustness.*

### B.2 Dynamic job speed predictions

For dynamic job speed predictions (Table 1, third row), we define MAKEFEASJ in Algorithm 6 and prove its monotonicity. Theorem 4.2 then follows from the shared bounds in Appendix B.3.

**Algorithm 6:** MAKEFEASJ

1: **Input:** Instance $J$, job speed predictions $\{\hat{\ell}_j\}_{j\in J}$
2: $t_c \leftarrow 0$, $J^u \leftarrow \emptyset$
3: **for** $t \le \max_{j\in J} d_j$ **do**
4: $\quad d \leftarrow$ earliest deadline for unfinished released jobs
5: $\quad J^u \leftarrow$ unfinished released jobs with deadline $d$
6: $\quad \{w_j^u\}_{j\in J^u} \leftarrow$ remaining work for $J^u$
7: $\quad$ **if** $J^u = \emptyset$ **then**
8: $\quad\quad$ Idle at speed 0
9: $\quad$ **else if** $\sum_{j\in J^u} w_j^u/\hat{\ell}_j > d - t$ **then**
10: $\quad\quad$ Run the EDF job $j$ (break ties by higher predicted speed) at speed $\max\{\hat{\ell}_j, \frac{\sum_{j\in J^u} w_j^u}{d-t}\}$
11: $\quad$ **else**
12: $\quad\quad$ Run the EDF job $j$ at speed $\hat{\ell}_j$

We note that for all $J$ and $\{\hat{\ell}_j\}_{j\in J}$, (1) MAKEFEASJ$(J, \{\hat{\ell}_j\}_{j\in J})$ is feasible, (2) if all jobs $J$ can be run feasibly in EDF order at speeds $\{\hat{\ell}_j\}_{j\in J}$ then MAKEFEASJ$(J, \{\hat{\ell}_j\}_{j\in J})$ is equal to that schedule, and (3) MAKEFEASJ is an online algorithm. Thus, MAKEFEASJ is a feasibility projection.

**Lemma B.4.** *For any dynamic job speed prediction $\{\hat{\ell}_j\}_{j\in J}$ and jobs $J' \subseteq J$, we have*

$$\textit{cost}(\text{MAKEFEASJ}(J', \{\hat{\ell}_j\}_{j\in J'})) \le \textit{cost}(\text{MAKEFEASJ}(J, \{\hat{\ell}_j\}_{j\in J})).$$

*Proof.* Similarly, it suffices to prove the claim for $J = J' \cup \{j\}$ for an arbitrary additional job $j$; The general statement then follows by iterating this one-job extension.

We first make the following local monotonicity observation for a fixed active deadline. Fix a time $t$ and a deadline $d > t$. Let $J^u$ be the set of currently unfinished released jobs with deadline $d$ at time $t$, with remaining works $\{w_j^u\}_{j\in J^u}$ and predicted speeds $\{\hat{\ell}_j\}_{j\in J^u}$. Let $T \subseteq [t,d]$ be the times where $d$ is the earliest deadline. Consider running (only) these jobs in EDF order during $T$ until all jobs in $J^u$ finish. Then the additional energy incurred by Algorithm MAKEFEASJ to finish jobs $J^u$ by time $d$ is *nondecreasing* if either we increase $w_j$ for some $j \in J^u$ or we begin processing $J^u$ at some time $t' > t$. Specifically, if $\sum_{j\in J^u} w_j^u/\hat{\ell}_j \le d - t$, the algorithm runs each job at its predicted speed (Line 12), and increasing work or reducing slack cannot decrease energy. If the inequality fails, the algorithm switches to Line 10 and enforces the minimum average speed needed to finish by $d$, namely $\sum_{j\in J^u} w_j^u/(d-t)$, which is monotone in the remaining work $\sum_{j\in J^u} w_j^u$ and inverse slack $d - t$, and therefore by convexity of $s^\alpha$ prevents energy from dropping.

Let $s :=$ MAKEFEASJ$(J, \{\hat{\ell}_j\}_{j\in J})$ and $s' :=$ MAKEFEASJ$(J', \{\hat{\ell}_j\}_{j\in J'})$. Let $t_0$ be the first time in the execution of Algorithm MAKEFEASJ on $J$ such that $j$ belongs to the current earliest-deadline set $J^u$. By definition of $t_0$, for all $t < t_0$ MAKEFEASJ on $J$ and $J'$ makes identical choices and therefore $s(t) = s'(t)$.

Then consider time periods $T$ later than $t_0$ where there exists job $j'$ with $d_{j'} < d_j$. First observe that $r_{j'} > t_0$ must hold, so $j'$ has not been run prior to $t_0$. Then for any $t \in T$, $j \notin J^u$ and $s, s'$ behave identically on these periods as well.

Thus, we focus on times where $d_j$ is the earliest deadline. At time $t_0$, let $d$ be the minimum deadline among unfinished released jobs by $s'$ for $J'$. We denote these jobs $J^u$ and the work left $w^u_{j'}$ for $j' \in J^u$. Then there are two cases.

**Case 1:** $d = d_j$**.** At time $t_0$, the active set for $s$ is equal to $J^u \cup \{j\}$ is exactly the active set in $s'$ plus the additional job $j$, and the remaining work is equal to $\sum_{j' \in J^u} w^u_{j'} + w_j$. By the local monotonicity claim, since we have increased total work, the cost for $s$ to complete all jobs in $J^u \cup \{j\}$ is at least the cost for $s'$ to complete all jobs in $J^u$.

Let $t_1$ be the time that $s$ completes jobs in $J^u$ and $t_2$ the time that $s'$ completes jobs in $J^u$. Schedule $s$ has the same remaining jobs and corresponding work at time $t_1$ as $s'$ at time $t_2$. However, $t_1 \geq t_2$. To see this, observe that $t_1 < t_2$ implies that following the prediction for $J^u \cup \{j\}$ is always feasible, which means following the prediction for just $J^u$ is also feasible. However, this means both $s$ and $s'$ follow the predicted speeds, and therefore it takes the same amount of time to complete jobs in $J^u$, which makes $t_1 < t_2$ impossible. Therefore, for every later deadline phase, schedule $s$ starts the work for that phase no earlier than $s'$, to which we can apply the same local monotonicity property. Summing over all such phases yields $\text{cost}(s) \geq \text{cost}(s')$.

**Case 2:** $d > d_j$**.** Then in the execution on $J'$ at time $t_0$, the current minimum deadline is some $d > d_j$. In the execution on $J$, job $j$ is active with the strictly smaller deadline $d_j$ at time $t_0$, so by EDF, $s$ must process jobs of deadline at most $d_j$ before it can process jobs of deadline $d$. Similarly to the previous lemma, let $t_1$ be the first time in $s$ at which the current minimum deadline becomes $d$. By a similar argument as Lemma B.1, $t_1 \geq t_0$, and during $[t_0, t_1)$ the schedule $s$ does not process any job of deadline $d$, while $s'$ may have already reduced the remaining work in that deadline group. Hence, at $t_1$, the $d$-deadline group in $s$ has *at least* as much remaining work as in $s'$, and also an earlier starting time to work on the deadline $d$, since $t_1$ is later. Applying the local monotonicity claim to this deadline-$d$ phase shows that the energy that $s$ spends to complete this phase is at least the energy that $s'$ spends on the corresponding phase. The same reasoning then applies inductively to all subsequent deadlines/phases.

Combining the above with the equality on the prefix, we have $\text{cost}(s) \geq \text{cost}(s')$, proving the lemma. □

### B.3 Bounds shared by both models

The following bounds apply to both offline machine speed and dynamic job speed predictions (Table 1, second and third rows), with their respective projections.

**Lemma B.5.** *In the offline machine speed prediction model, Algorithm 2 with* MAKEFEAS = MAKEFEASM *is* $((c_{on}\lambda)^{1/\alpha} + 1)^\alpha$*-smooth with respect to* MAKEFEASM.

*Moreover, in the dynamic job speed prediction model, Algorithm 2 with* MAKEFEAS = MAKEFEASJ *is also* $((c_{on}\lambda)^{1/\alpha} + 1)^\alpha$*-smooth with respect to* MAKEFEASJ.

*Proof.* Let $\mathcal{P}^m$ and $\mathcal{P}^j$ be the spaces of all predictions for the offline machine speed and dynamic job speed predictions models. Let $(\text{MF}, P) \in \{(\text{MAKEFEASM}, \hat{s}) : \hat{s} \in \mathcal{P}^m\} \cup \{(\text{MAKEFEASJ}, \{\hat{\ell}_j\}_{j \in J}) : \{\hat{\ell}_j\}_{j \in J} \in \mathcal{P}^j\}$.

Consider an instance $J$. At time $t$, Line 5 of Algorithm 2 only adds a job $j$ to $J^{\text{on}}$ if $\text{cost}(\text{OPT}(J^{\text{on}} \cup \{j\})) \leq \lambda \cdot \text{cost}(\text{MF}(J_{\leq t}, P_{\leq t}))$. Combined with the fact that $\text{cost}(\text{MF}(J_{\leq t}, P_{\leq t}))$ is monotonically increasing with respect to $t$ (by Lemma B.1 for MF = MAKEFEASM and Lemma B.4 for MF = MAKEFEASJ), we have that $\text{cost}(\text{OPT}(J^{\text{on}})) \leq \lambda \cdot \text{cost}(\text{MF}(J, P))$ when the algorithm terminates. Thus, since ONLINE is a $c_{\text{on}}$-competitive algorithm, we have that upon termination

$$\text{cost}(\text{ONLINE}(J^{\text{on}})) \leq c_{\text{on}} \cdot \text{cost}(\text{OPT}(J^{\text{on}})) \leq c_{\text{on}} \cdot \lambda \cdot \text{cost}(\text{MF}(J, P)).$$

We conclude that

$$\begin{aligned}
\text{cost}(\text{ALG}(J,P)) &=_{(1)} \text{cost}(\textsc{Online}(J^{\text{on}}) + \text{MF}(J^{\text{p}},P)) \\
&\leq_{(2)} (\text{cost}(\textsc{Online}(J^{\text{on}}))^{\frac{1}{\alpha}} + \text{cost}(\text{MF}(J^{\text{p}},P))^{\frac{1}{\alpha}})^{\alpha} \\
&\leq_{(3)} ((c_{\text{on}}\lambda)^{\frac{1}{\alpha}}\text{cost}(\text{MF}(J,P))^{\frac{1}{\alpha}} + \text{cost}(\text{MF}(J^{\text{p}},P))^{\frac{1}{\alpha}})^{\alpha} \\
&\leq_{(4)} ((c_{\text{on}}\lambda)^{\frac{1}{\alpha}}\text{cost}(\text{MF}(J,P))^{\frac{1}{\alpha}} + \text{cost}(\text{MF}(J,P))^{\frac{1}{\alpha}})^{\alpha} \\
&\leq ((c_{\text{on}}\lambda)^{1/\alpha}+1)^{\alpha}\cdot\text{cost}(\text{MF}(J,P)),
\end{aligned}$$

where (1) is by definition of Algorithm 2 and since ONLINE and MAKEFEAS are online algorithms, (2) by Lemma A.4, (3) by the argument above, and (4) is by Lemma B.1 for MF = MAKEFEASM and Lemma B.4 for MF = MAKEFEASJ. □

**Lemma B.6.** *In the offline machine speed prediction model, Algorithm 2 with* MAKEFEAS = MAKEFEASM *is* $(c_{on}^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}})^{\alpha}$*-robust.*

*Moreover, in the dynamic job speed prediction model, Algorithm 2 with* MAKEFEAS = MAKEFEASJ *is also* $(c_{on}^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}})^{\alpha}$*-robust.*

*Proof.* Let $\mathcal{P}^m$ and $\mathcal{P}^j$ be the spaces of all predictions for the offline machine speed and dynamic job speed predictions models. Let $(\text{MF},P) \in \{(\textsc{MakeFeasM},\hat{s}) : \hat{s} \in \mathcal{P}^m\} \cup \{(\textsc{MakeFeasJ},\{\hat{\ell}_j\}_{j\in J}) : \{\hat{\ell}_j\}_{j\in J} \in \mathcal{P}^j\}$.

Consider an instance $J$. Let $t^\star$ be the last time Algorithm 2 adds a job to $J^{\text{p}}$. Let $j^\star$ be this last job added to $J^{\text{p}}$. By Line 5 of Algorithm 2,

$$\text{cost}(\text{OPT}(J^{\text{on}} \cup \{j^\star\})) > \lambda \cdot \text{cost}(\text{MF}(J_{\leq t^\star}, P_{\leq t^\star})),$$

which implies

$$\text{cost}(\text{MF}(J_{\leq t^\star}, P_{\leq t^\star})) < \frac{\text{cost}(\text{OPT}(J))}{\lambda}.$$

The cost of Algorithm 2 is then bounded as follows:

$$\begin{aligned}
\text{cost}(\text{ALG}(J,P)) &=_{(1)} \text{cost}(\textsc{Online}(J^{\text{on}}) + \text{cost}(\text{MF}(J^{\text{p}},P)) \\
&=_{(1)} \text{cost}(\textsc{Online}(J^{\text{on}}) + \text{MF}(J^{\text{p}},P)) \\
&\leq_{(3)} (\text{cost}(\textsc{Online}(J^{\text{on}}))^{1/\alpha} + \text{cost}(\text{MF}(J_{\leq t^\star},P))^{1/\alpha})^{\alpha} \\
&\leq_{(4)} (c_{\text{on}}^{1/\alpha}\cdot\text{cost}(\text{OPT})^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}}\cdot\text{cost}(\text{OPT})^{1/\alpha})^{\alpha} \\
&\leq (c_{\text{on}}^{1/\alpha} + \frac{1}{\lambda^{1/\alpha}})^{\alpha}\text{cost}(\text{OPT}),
\end{aligned}$$

where (1) is by definition of Algorithm 2 and since ONLINE and MAKEFEAS are online algorithms, (2) by Lemma A.4, (3) by Lemma B.1 for MF = MAKEFEASM and Lemma B.4 for MF = MAKEFEASJ, and (4) by the argument above and the competitive ratio for the online algorithm. □

Theorem 4.2 follows from Lemmas B.5 and B.6 with its stated choice of $\lambda$.

# C Additional Experimental Results and Description of Setup

## C.1 Implementation details of Algorithm 1

Given instance $J$ and dynamic machine speed predictions $\hat{s}$, we modify Algorithm 1 as follows. Rather than calculate at every time $t$ the forecast schedule $s_{\text{F}}$, which may not exist for inaccurate $\hat{s}$, we replace it with a naive heuristic $s'$, similarly recalculated at every $t$. At time $t$, $s'$ is set to the predicted speed made feasible by ensuring it does not fall behind OA. Then $s'(\tau) = s'(t)$ for $\tau \in [t, d_{j^{\text{last}}}]$ where $j^{\text{last}} \in J_{\leq t}$ is the job with the latest deadline. We also remove the reassignment that depends on (potentially nonexistent) $s_{\text{F}}$. Then we run jobs in $J^{\text{p}}$ according to $s'$.

### C.2 Additional results

We present plots for the performance of our algorithms as a function of the confidence parameter $\lambda$ (Figure 3). This illustrates how $\lambda$ controls the trade-off between trusting the prediction and falling back to the online algorithm. We use the same method as Section 6.1 to generate instances data. We set the perturbation rate to always be 1 since TPE is not part of this experiment. The plots show a regime of low prediction error relative to OA ($\eta = 0.05$), a regime where the predictions have performance relatively close to that of OA ($\eta = 0.08$), a regime where the predictions perform much worse than OA ($\eta = 0.1$).

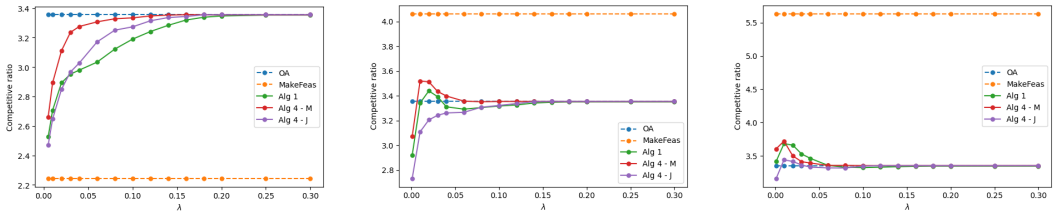


Figure 3: We plot the average competitive ratio (over 20 instances) of our algorithms against different $\lambda$ values. The left figure sets $\eta = 0.05$, the center figure sets $\eta = 0.08$, and the right figure sets $\eta = 0.1$.

When the prediction is strong ($\eta = 0.05$, where MAKEFEAS clearly outperforms OA), all three of our algorithms achieve their lowest competitive ratios for small $\lambda$, closely tracking MAKEFEAS. As $\lambda$ increases, performance degrades smoothly toward OA, consistent with the algorithms assigning more jobs to $J^{\text{on}}$ and incurring cost close to that of OA. Among the three, Alg 1 attains the lowest ratio at small $\lambda$, while Alg 4 - M lies slightly above the other two, mirroring the same ordering observed in the synthetic experiments of Section 6.

When the prediction performance is roughly comparable to OA ($\eta = 0.08$), neither benchmark dominates and the gap our algorithms can exploit is small. Accordingly, all three curves concentrate near OA over a broad range of $\lambda$ (within roughly $0.5$ deviation in competitive ratio), reflecting that the algorithms cannot extract large gains from a prediction whose cost is not decisively better than the baseline. The non-monotonic dip at very small $\lambda$ for Alg 4 - J reflects that aggressive trust can occasionally help even when MAKEFEAS and OA are close.

When the predictions deviate significantly from OPT and MAKEFEAS has high competitive ratio ($\eta = 0.1$), small $\lambda$ leads to slightly worse performance than larger $\lambda$, as the former enforces that the algorithms follow the costly prediction more closely. Crucially, even at the smallest $\lambda$, the competitive ratios of all three algorithms remain bounded around $4$, well below MAKEFEAS, which exceeds $5.5$, and decrease back toward OA as $\lambda$ grows. This nicely demonstrates the robustness of our algorithms across the entire range of $\lambda$.

### C.3 Real-data comparison with exact job predictions

We supplement the real-data experiments with a comparison that gives TPE additional exact job predictions. Each job in the realized instance is added to TPE's predicted instance with probability $p$, revealing its exact release time, deadline, and work at $t = 0$. Our algorithms retain their original speed predictions derived from the previous day's trace, and all other experimental settings remain unchanged. This augmentation provides TPE with a substantial information advantage. The fixed-probability provision of exact jobs is not intended to model a realistic prediction process; it serves as a controlled comparison under conditions favorable to TPE.

Figure 4 shows the effect of this additional information. With $p = 0.5$ (left), TPE substantially improves over OA, but still incurs higher energy cost than each of our three algorithms on most of the displayed days. With $p = 0.8$ (right), matching the exact-job probability in our synthetic prediction setting, TPE achieves slightly lower energy cost on many days, while its performance is generally comparable to ours. Thus, our algorithms remain competitive even when TPE receives substantial additional exact job information, using only the much less informative speed predictions derived from the previous day's trace.

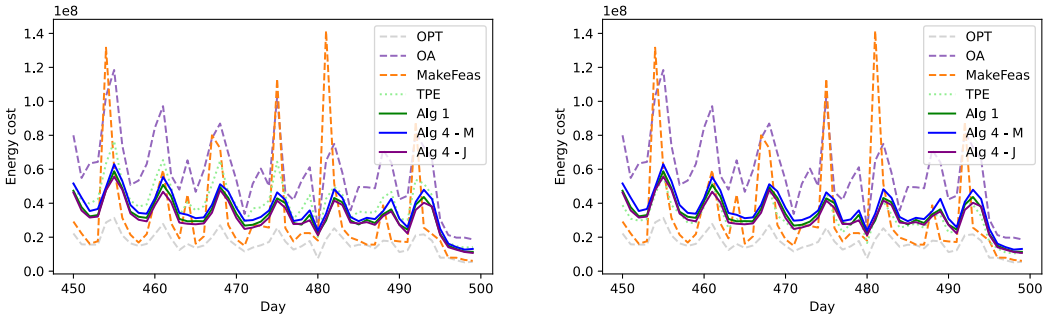


Figure 4: Energy cost of our algorithms and benchmarks on the Brightkite dataset, with each realized job added to TPE's predictions with probability $p = 0.5$ (left) and $p = 0.8$ (right). Our algorithms retain their original previous-day-based speed predictions.

All experiments are lightweight numerical simulations and were run on a single standard laptop CPU. No GPU or cluster is required. Each synthetic instance with $n = 100$ jobs runs in under a second, and the full Brightkite sweep over 851 days completes in under an hour.

# D Missing Results and Proofs from Section 5

## D.1 Non-uniqueness of instances given an optimal speed schedule

A non-trivial optimal speed schedule does not uniquely determine the underlying job instance, because any such schedule is consistent with infinitely many distinct instances. Consider any non-empty instance $J$ with $\text{cost}(\text{OPT}(J)) > 0$. Pick any job $j \in J$ with $w_j > 0$, which exists since $\text{cost}(\text{OPT}(J)) > 0$. For each $k \geq 1$, let $J_k$ be the instance obtained by replacing $j$ with $k$ jobs of work $w_j/k$, each having release date $r_j$ and deadline $d_j$. The $J_k$'s are pairwise distinct. Since these $k$ new jobs share $j$'s release date and deadline, any schedule for $J$ can be turned into a schedule for $J_k$ with the same machine speed profile, and vice versa, by splitting or merging the work assigned to $j$. Hence $\text{OPT}(J)$ and $\text{OPT}(J_k)$ have the same cost, so $\text{OPT}(J_k) = \text{OPT}(J)$.

## D.2 Covering numbers

We compare our two hypothesis classes of predictors over the k-discrete regime by calculating their respective covering numbers.

**Definition D.1.** Let $\mathcal{H}$ be a hypothesis class and $\varepsilon > 0$. The covering number $\mathcal{N}(\varepsilon, \mathcal{H}, \|\cdot\|_\infty)$ is the minimum number of balls of radius $\varepsilon$ needed to cover $\mathcal{H}$ under the $\|\cdot\|_\infty$ metric.

Let us define $\mathcal{H}_W = \{\{W_{ab}\}_{1 \leq a < b \leq k} : W_{ab} \leq B\}$ and $\mathcal{H}_S = \{\{s_i\}_{i \in [k-1]} : s_i \leq S\}$ as the hypothesis classes for workload predictions and speed predictions respectively, with $B$ and $S$ as general upper bounds on the values. $\mathcal{H}_W$ has covering number $\mathcal{N}(\varepsilon, \mathcal{H}_W, \|\cdot\|_\infty) = \Theta((B/\varepsilon)^{\binom{k}{2}})$ and $\mathcal{H}_S$ has covering number $\mathcal{N}(\varepsilon, \mathcal{H}_S, \|\cdot\|_\infty) = \Theta((S/\varepsilon)^{k-1})$.

## D.3 From covering number to sample complexity

We can now derive our sample complexity bounds from the covering numbers with the following result.

**Theorem D.2.** *Let $\mathcal{H}$ be either $\mathcal{H}_W$ or $\mathcal{H}_S$. Assume sample instances $J_1, \ldots, J_N$ are drawn i.i.d. from a distribution over $\mathcal{J}$, and let $\ell : \mathcal{H} \times \mathcal{J} \to [0, 1]$ be an $L$-Lipschitz loss in its first argument under $\|\cdot\|_\infty$. Let $\hat{h}$ be an empirical risk minimizer over $\mathcal{H}$. If*

$$\log \mathcal{N}(\varepsilon, \mathcal{H}, \|\cdot\|_\infty) \;\leq\; C\, d\, \log \tfrac{A}{\varepsilon}$$

*for all $\varepsilon \in (0, A]$, then with probability at least $1 - \delta$,*

$$R(\hat{h}) - \inf_{h \in \mathcal{H}} R(h) \;\leq\; C' \left( LA\sqrt{\tfrac{d}{N}} + \sqrt{\tfrac{\log(1/\delta)}{N}} \right),$$

*for a constant $C' > 0$.*

*Proof.* By $L$-Lipschitzness of $\ell$ we have that,

$$\log \mathcal{N}(\varepsilon, \ell \circ \mathcal{H}, L_\infty) \leq \log \mathcal{N}(\varepsilon/L, \mathcal{H}, \|\cdot\|_\infty) \leq Cd \log \frac{LA}{\varepsilon}.$$

Then we can apply Dudley's entropy integral (17) and bound the Rademacher complexity of $\mathfrak{R}_N(\ell \circ \mathcal{H})$ and show that it is bound above by

$$C' \int_0^{LA} \sqrt{\frac{\log \mathcal{N}(\varepsilon, \ell \circ \mathcal{H})}{N}}\, d\varepsilon \leq \frac{C'}{\sqrt{N}} \int_0^{LA} \sqrt{Cd \log(LA/\varepsilon)}\, d\varepsilon = O\left(LA\sqrt{\frac{d}{N}}\right).$$

Applying the standard Rademacher generalization bound for empirical risk minimization (13), with probability at least $1 - \delta$,

$$R(\hat{h}) - \inf_{h \in \mathcal{H}} R(h) \leq 2\mathfrak{R}_N(\ell \circ \mathcal{H}) + \sqrt{\frac{\log(1/\delta)}{N}} = O\left(LA\sqrt{\frac{d}{N}} + \sqrt{\frac{\log(1/\delta)}{N}}\right).$$

as desired. □

With this result, we can prove our main sample complexity result.

**Theorem 5.1.** *Let $\ell : \mathcal{H} \times \mathcal{J} \to [0, 1]$ be $L$-Lipschitz in its first argument under $\|\cdot\|_\infty$. Assume the workloads satisfy $W_{i,j} \leq B$ for all $i, j \in [k]$ and speeds satisfy $s_i \leq S$ for all $i \in [k]$. To achieve excess risk $\eta$ with probability at least $1 - \delta$, it suffices to use*

$$N_{\text{speed}} = O\left(\frac{L^2 S^2 k + \log(1/\delta)}{\eta^2}\right) \quad \textit{and} \quad N_{\text{work}} = O\left(\frac{L^2 B^2 k^2 + \log(1/\delta)}{\eta^2}\right)$$

*samples for machine speed predictions and job (input) predictions respectively.*

*Proof.* Assume we have samples $J_1, \ldots, J_N$ drawn i.i.d from a distribution over $\mathcal{J}$ and let $\hat{h}_W$ and $\hat{h}_S$ be the respective empirical risk minimizers of $\mathcal{H}_W$ and $\mathcal{H}_S$. Applying the fact that $\log \mathcal{N}(\varepsilon, \mathcal{H}_W, \|\cdot\|_\infty) = O(k^2 \log(B/\varepsilon))$ and $\log \mathcal{N}(\varepsilon, \mathcal{H}_S, \|\cdot\|_\infty) = O(k \log(S/\varepsilon))$ to Theorem D.2 gives us that, with probability at least $1 - \delta$,

$$R(\hat{h}_{\text{speed}}) - \inf_{h \in \mathcal{H}_S} R(h) \;=\; O\left(LS\sqrt{\tfrac{k}{N}} + \sqrt{\tfrac{\log(1/\delta)}{N}}\right),$$

$$R(\hat{h}_{\text{work}}) - \inf_{h \in \mathcal{H}_W} R(h) \;=\; O\left(LB\tfrac{k}{\sqrt{N}} + \sqrt{\tfrac{\log(1/\delta)}{N}}\right).$$

Equivalently, to achieve excess risk at most $\eta$, it suffices to take number of samples

$$N_{\text{speed}} = O\Big(\tfrac{L^2 S^2 k + \log(1/\delta)}{\eta^2}\Big), \qquad N_{\text{work}} = O\Big(\tfrac{L^2 B^2 k^2 + \log(1/\delta)}{\eta^2}\Big).$$

□

# E Polynomial-time construction of the forecast schedule

We give an explicit implementation of FORECAST and prove that it returns the minimum-cost schedule consistent with the observations. Fix a time $t \leq t^\star$, write $J = J_{\leq t}$, and recall that

$$\Sigma_t = \big\{ s : \; s(\tau) = \hat{s}(\tau) \text{ for } \tau \leq t, \quad s = \mathrm{OPT}(J') \text{ for some finite } J' \text{ with } J'_{\leq t} = J \big\}.$$

Here OPT denotes the optimal machine-speed function, and it will be constant between consecutive releases and deadlines. In particular, every job in $J' \setminus J$ must have release time strictly greater than $t$.

Assume that jobs have $0 \leq r_j < d_j$ and $w_j > 0$, and that the observed prediction is a nonnegative piecewise-constant function. We fix the EDF tie-breaking rule.

### E.1 Construction

Replay EDF on the arrived jobs at speed $\hat{s}$ over $[0, t]$. Let $p_j$ be job $j$'s remaining workload at time $t$, and define

$$P = \{j \in J : p_j < w_j\}, \qquad R = \{j \in J : p_j > 0\}, \qquad \mu_j = \min_{\tau \in (r_j, \min\{t, d_j\})} \hat{s}(\tau).$$

Thus $P$ is the started jobs, including completed jobs, while $R$ is the unfinished jobs. The values $p_j$ remain fixed during the construction. We use $\min \emptyset = +\infty$ and $\max \emptyset = 0$.

Let $D_0 = t < D_1 < \cdots < D_m$ list $t$ and the distinct deadlines in $J$ greater than $t$. For $0 \le i \le m$, set

$$B_i = \max_{i<k\le m} \frac{\sum_{j \in R:\, D_i < d_j \le D_k} p_j}{D_k - D_i}, \qquad F_i = \max_{j \in P:\, d_j > D_i} \mu_j. \tag{3}$$

The quantity $B_i$ is the minimum constant speed needed to complete the remaining workloads $p_j$ of all jobs $j \in R$ with $d_j > D_i$ by their deadlines, starting at $D_i$. In the next subsection we will show that in an optimal schedule, once job $j$ has been processed at speed $\mu_j$, the speed cannot fall below $\mu_j$ before its deadline. Thus $F_i$ is the largest $\mu_j$ among started jobs whose deadlines are after $D_i$.

**Algorithm 7:** FORECAST

**Input:** Current time $t$, arrived jobs $J = J_{\le t}$, and prediction prefix $\hat{s}_{\le t}$.
**Output:** A minimum-cost $s_F \in \Sigma_t$, or $\emptyset$ if $\Sigma_t = \emptyset$.

Replay EDF on $[0, t]$ and set $p_j$ to job $j$'s remaining load at time $t$
$P \leftarrow \{j \in J : p_j < w_j\}$; $R \leftarrow \{j \in J : p_j > 0\}$
Compute $\mu_j$, the deadline grid $(D_i)$, and $B_i, F_i$ as defined above
**if** *EDF misses a deadline, idles at positive speed, or processes some $j$ at speed $\neq \mu_j$* **then return** $\emptyset$
Initialize $s_F(\tau) \leftarrow \hat{s}(\tau)$ for $\tau \le t$ and $s_F(\tau) \leftarrow 0$ for $\tau > t$
**if** $R = \emptyset$ **then**
  **if** $F_0 > 0$ **then return** $\emptyset$
  **return** $s_F$
**end**
$d_* \leftarrow \min_{j \in R} d_j$; $v_0 \leftarrow \max\{B_0, F_0\}$; $v \leftarrow v_0$
**for** $i = 0, \ldots, m-1$ **do**
  $v \leftarrow \min\{v, \max\{B_i, F_i, v_0 \mathbf{1}_{\{D_i < d_*\}}\}\}$
  Set $s_F(\tau) \leftarrow v$ for $\tau \in (D_i, D_{i+1}]$
**end**
**if** *$s_F(d_j^-) > \mu_j$ for some $j \in R$, or $s_F(d_j^-) < \mu_j$ for some $j \in P \cap R$* **then return** $\emptyset$
**return** $s_F$

Algorithm 7 is the pseudocode of our construction. Here we give a description of what the algroithm does. We keep the observed schedule up to $t$ and construct its continuation. The construction uses the following optimality condition, proved below: a job can be processed only at the minimum speed over its window. We first check that the EDF execution of the observed prefix misses no deadline, never idles at positive speed, and processes each started job only at speed $\mu_j$. If any check fails, we return $\emptyset$.

The continuation must both finish the remaining work and respect the processing already performed. The quantity $B_i$ is the minimum constant speed that would suffice if the remaining workloads $p_j$ of all jobs with $d_j > D_i$ were postponed until $D_i$. The quantity $F_i$ accounts for the past: once a job has been processed at speed $\mu_j$, the speed cannot fall below $\mu_j$ before its deadline, since this would violate the optimality condition. Thus $F_i$ is the largest such requirement among jobs whose deadlines are after $D_i$.

If unfinished jobs remain, we start at speed $v_0 = \max\{B_0, F_0\}$. We keep this speed until the earliest remaining deadline $d_*$. This ensures that the first constant-speed interval contains the deadline of an unfinished job. In the proof, we first process the remaining work of arrived jobs whose deadlines lie in this interval, starting at time $t$. If part of the interval remains unused, we fill it with a new job released when that work finishes, strictly after $t$. At each subsequent deadline $D_i$, we lower the speed to $\max\{B_i, F_i\}$ if this is smaller than the current speed. Otherwise, we keep the current speed.

Finally, because the continuation is nonincreasing, its minimum before $d_j$ is $s_F(d_j^-)$. For an unfinished job, this value must be at most $\mu_j$; otherwise, all future processing would occur above a speed already observed in its window. If the job has also started, equality is required so that its past and future processing can occur at the same minimum speed. We reject if these conditions fail. If no unfinished jobs remain, we instead use a zero-speed continuation, provided that $F_0 = 0$.

One special note for handeling the first interval: in the first constant-speed interval, we must start by processing a job that has already arrived, since any jobs added in the proof are released strictly after $t$. Let $d_* = \min_{j \in R} d_j$ be the earliest deadline among unfinished jobs. If the speed dropped before $d_*$, every unfinished job would have a lower speed available before its deadline, so none could be processed at $v_0$ in an optimal schedule. We therefore keep the speed at $v_0$ until $d_*$.

### E.2 Correctness

We show that, in an optimal schedule, each job is processed only at the minimum machine speed between its release time and deadline.

**Lemma E.1.** *Fix a finite job instance and a nonnegative machine speed function $s$. If some schedule meets all deadlines using speed at most $s(\tau)$ at each time $\tau$, then preemptive EDF running at speed $s(\tau)$ whenever a job is pending also meets all deadlines.*

*Proof.* Suppose that EDF misses a deadline $d$. Let $(a, d)$ be the longest interval ending at $d$ throughout which EDF always has a released but unfinished job with deadline at most $d$. By the choice of $a$, all jobs released before $a$ with deadlines at most $d$ have been completed by $a$. Let $S = \{j : r_j \geq a,\ d_j \leq d\}$. Throughout $(a, d)$, EDF therefore uses all available speed to process jobs in $S$. Since some of their work remains unfinished at $d$, we have $\sum_{j \in S} w_j > \int_a^d s(\tau)\, d\tau$. However, every job in $S$ must be processed entirely within $[a, d]$, so no schedule whose speed is bounded by $s$ can complete all of them on time. This contradicts the assumed existence of a feasible schedule. □

**Lemma E.2.** *Consider a feasible schedule with machine speed $s(\tau) = \sum_j x_j(\tau)$. This schedule minimizes energy if and only if every job $j$ is processed only when the machine speed equals $\ell_j$, where*

$$\ell_j = \min_{\tau \in (r_j, d_j)} s(\tau). \tag{4}$$

*Every finite job instance has an optimal EDF schedule whose speed changes only finitely many times.*

*Proof.* Split time into intervals at every release time, deadline, and moments of changes in machine speed. Within each interval, spread each job's processing evenly over the interval. This preserves feasibility and, by Jensen's inequality, does not increase energy. Suppose for the sake of contradiction that job $j$ receives positive work on an interval of speed $a$, and another interval in its window has speed $b < a$. Move $\delta$ units of $j$'s work from the first interval to the second, distributing the change uniformly within each interval. The change is feasible for sufficiently small $\delta > 0$, and the change of its energy cost from that at $\delta = 0$ is $\alpha(b^{\alpha-1} - a^{\alpha-1}) < 0$, contradicting optimality.

Conversely, let $\{y_j\}$ be any competing feasible schedule, with machine speed $z$. Convexity and (4) give

$$\begin{aligned}
\operatorname{cost}(z) - \operatorname{cost}(s) &\geq \alpha \int s(\tau)^{\alpha-1}(z(\tau) - s(\tau))\, d\tau \\
&= \alpha \sum_j \int s(\tau)^{\alpha-1}(y_j(\tau) - x_j(\tau))\, d\tau \\
&\geq \alpha \sum_j \ell_j^{\alpha-1} \left( \int y_j - \int x_j \right) = 0.
\end{aligned}$$

The last inequality uses $s \geq \ell_j$ throughout $j$'s window and $s = \ell_j$ wherever $x_j > 0$. The above proves the optimality of $s$. □

Now we show the desired properties of this speed forecast, $s_F$. Suppose the initial checks of the algorithm pass and some jobs remain unfinished. Let $v_i$ be the speed on $(D_i, D_{i+1}]$. By the

algorithm's update rule,

$$v_i = \min_{0 \le h \le i} \max\{B_h, F_h, v_0 \mathbf{1}_{\{D_h < d_*\}}\}. \tag{5}$$

Thus $s_F$ is nonincreasing after $t$. Since $F_i$ is nonincreasing, the update also gives

$$v_i \ge F_i, \qquad v_i = v_0 \quad \text{if } D_i < d_*. \tag{6}$$

The first inequality is because both arguments of the update's outer minimum are at least $F_i$. The second follows in the same way from the $v_0$ term and the upper bound $v_i \le v_0$.

**Lemma E.3.** *If $\Sigma_t \neq \emptyset$, the algorithm does not reject. Moreover, its constructed schedule $s_F$ satisfies $s_F(\tau) \le s(\tau)$ for every $s \in \Sigma_t$ and every $\tau$.*

*Proof.* Fix $s \in \Sigma_t$ and an instance $J'$ with $s = \mathrm{OPT}(J')$ and $J'_{\le t} = J$. The EDF execution of $J'$ must be the same for $s_F$ and $s$ on $[0, t]$, since no job in $J' \setminus J$ is available there. It misses no deadline and does not idle at positive speed. By Lemma E.2, every job has speed equal to its minimum speed over its full window. For a job ever processed before $t$, therefore, its speed is exactly $\mu_j$. This proves that the prefix checks pass and that

$$s(\tau) \ge \mu_j \qquad (j \in P,\ t < \tau < d_j). \tag{7}$$

In the case $R = \emptyset$, all jobs released by time $t$ have been completed for both $J'$ and $J$. If $F_0 > 0$, (7) requires $s$ to be positive immediately after $t$. This will contradict the optimality of $s$ for $J'$. Therefore $F_0 = 0$ and subsequently the algorithm sets $s_F(\tau) = 0$ for all $\tau > t$. Therefore we have $s_F(\tau) \le s(\tau)$ for all $\tau$. For the rest of the proof, we assume $R \neq \emptyset$.

Let $e$ be the job with the earliest deadline in $R$, and we have $d_e = d_*$. Choose a short interval immediately after $t$, ending before $d_*$, during which no job is released and $s$ is constant. Let $c$ be its speed. If $c = 0$, the minimum of $s$ between $r_e$ and $d_e$ would be zero. Lemma E.2 would then allow $e$ to be processed only at speed zero, which is impossible because $p_e > 0$. Thus $c > 0$, and EDF of $s$ starts by processing $e$ at speed $c$. The same lemma implies that $c$ is the minimum of $s$ between $r_e$ and $d_e$, so $s(\tau) \ge c$ for $t < \tau < d_*$.

For every job $j \in R$, this initial interval lies between $r_j$ and $d_j$. The minimum of $s$ between its release and deadline is therefore at most $c$. By Lemma E.2, these jobs can be processed only when the machine speed is at most $c$. All their remaining work due by $D_k$ must be completed between $t$ and $D_k$, so

$$\sum_{j \in R:\, d_j \le D_k} p_j \le c(D_k - t) \qquad (1 \le k \le m).$$

Hence by the definition we have $B_0 \le c$. Equation (7) also gives $F_0 \le c$. Since $v_0 = \max\{B_0, F_0\}$, we obtain

$$s(\tau) \ge v_0 \qquad (t < \tau < d_*). \tag{8}$$

We next show that $s_F(\tau) \le s(\tau)$ before the algorithm performs its final checks. Suppose instead that $s_F(\tau_0) > s(\tau_0)$ at some time $\tau_0 > t$, and let $v = s_F(\tau_0) > 0$. Let $a$ be the start of the first interval after $t$ on which $s$ is constant and less than $v$, and let $c < v$ be its speed. Then $a \le \tau_0$ and $s(\tau) \ge v$ for $t < \tau < a$. Since $s_F$ is zero after $D_m$, we have $a \le \tau_0 < D_m$. Find $i$ such that $D_i \le a < D_{i+1}$.

For every job $j \in R$ with $d_j > a$, the speed $s$ equals $c$ for some time between $r_j$ and $d_j$. Its minimum speed over that time window is therefore $\le c$, so it can be processed only when $s \le c$. Since $s \ge v > c$ throughout $(t, a)$, the job receives no processing between $t$ and $a$ and still has $p_j$ units of work left at $a$.

There is no deadline in $(D_i, a]$. Thus, for every $k > i$, all jobs in $R$ with $D_i < d_j \le D_k$ still have their full remaining workloads $p_j$ at time $a$. They must complete this work by $D_k$, and they can be processed only when the machine speed is at most $c$. Therefore,

$$\sum_{j \in R:\, D_i < d_j \le D_k} p_j \le c(D_k - a) \le c(D_k - D_i).$$

This gives $B_i \le c$. Every job in $P$ with deadline after $D_i$ also has deadline after $a$, so (7) gives $F_i \le c$. If $D_i < d_*$, then $D_{i+1} \le d_*$ because $d_*$ is one of the listed deadlines. Hence $a < d_*$, and (8) gives $v_0 \le c$. Since $s_F$ does not increase after $t$ and $a \le \tau_0$, the algorithm's update gives

$$v = s_F(\tau_0) \le v_i \le \max\{B_i, F_i, v_0 \mathbf{1}_{\{D_i < d_*\}}\} \le c < v,$$

which is a contradiction. Thus $s_F(\tau) \le s(\tau)$ for all $\tau$.

Finally, we show that neither of the algorithm's final two rejection conditions can hold. Suppose that $s_F(d_j^-) > \mu_j$ for some $j \in R$. Since $s_F$ does not increase after $t$, the inequality just proved gives

$$s(\tau) \ge s_F(\tau) \ge s_F(d_j^-) > \mu_j \qquad (t < \tau < d_j).$$

However, the minimum of $s$ between $r_j$ and $d_j$ is at most $\mu_j$, since $s$ agrees with $\hat{s}$ up to time $t$. By Lemma E.2, job $j$ can be processed only when $s \le \mu_j$. It therefore is not processed at all between $t$ and $d_j$, contradicting $p_j > 0$ and the feasibility of $s$. For the other condition, (6) already gives $s_F(d_j^-) \ge \mu_j$ for every $j \in P \cap R$. Thus neither condition causes the algorithm to return $\emptyset$.

□

**Lemma E.4.** *If the algorithm returns a schedule $s_F$, then $s_F \in \Sigma_t$. Moreover, there is a feasible execution of $J$ using $s_F$ in which each job is processed only at the minimum speed of $s_F$ between its release time and deadline.*

*Proof.* Keep the execution of $J$ on $[0, t]$ computed by the algorithm, which runs EDF at speed $\hat{s} = s_F$.

First suppose $R = \emptyset$. Every job has already been completed at speed $\mu_j > 0$. Because the algorithm returns a schedule, we have $F_0 = 0$. If any job $j$ had a deadline after $t$, it would give $F_0 \ge \mu_j > 0$, a contradiction. Thus every deadline is at most $t$, and $\mu_j$ is the minimum of $s_F$ between $r_j$ and $d_j$. The algorithm's checks in the if condition ensure that every job was processed only at this speed and that the machine never idled while $s_F > 0$. Since $s_F = 0$ after $t$, Lemma E.2 gives $s_F = \mathrm{OPT}(J)$, proving both claims in this case.

Now suppose $R \neq \emptyset$. We construct an execution after $t$ and add jobs to obtain a finite instance $J'$ for which $s_F$ is optimal. Divide the times after $t$ at which $s_F > 0$ into intervals of constant speed, merging adjacent intervals whenever their speeds are equal. Consider one such interval $(a, b]$, with speed $v > 0$, and write $a = D_h$. For the first interval, $v = v_0 \ge B_0$. At the start of any later interval, the speed decreases, so the algorithm sets it to $\max\{B_h, F_h, v_0 \mathbf{1}_{\{D_h < d_*\}}\}$. Thus $v \ge B_h$ for every interval.

On $(a, b]$, process the remaining $p_j$ units of each job $j \in R$ with $a < d_j \le b$. Start at $a$, use speed $v$, and process these jobs one after another in increasing order of deadline. All these jobs have arrived by $t \le a$. For each of their deadlines $d$, the definition of $B_h$ gives

$$\sum_{j \in R:\, a < d_j \le d} p_j \le B_h(d - a) \le v(d - a). \tag{9}$$

Therefore every one of these jobs finishes by its deadline and is processed at speed $v = s_F(d_j^-)$.

Every job in $R$ has its deadline in one of these intervals. Indeed, suppose $s_F$ first becomes zero at $D_h$ while some job in $R$ has a deadline after $D_h$. Then $B_h > 0$. Since the preceding speed is also positive, the update cannot set the new speed to zero. Thus the construction above completes the remaining work of every job in $R$.

Let $W = \sum_{j \in R:\, a < d_j \le b} p_j$. The work assigned to $(a, b]$ finishes at $c = a + W/v \le b$. If $c < b$, add a job released at $c$ with deadline $b$ and workload $v(b - c)$, and process it at speed $v$ throughout $(c, b]$. Every added job is released strictly after $t$. If $a > t$, this follows from $c \ge a$. For the first interval, where $a = t$, (6) keeps $s_F$ equal to $v_0$ until $d_*$. Hence $b \ge d_*$, so at least one job in $R$ has its deadline in this interval. This gives $W > 0$ and therefore $c > t$.

Let $J'$ consist of $J$ and the added jobs. There are finitely many intervals, so $J'$ is finite. All added jobs are released after $t$, so $J'_{\le t} = J$. Together with the execution on $[0, t]$, the construction gives a feasible execution of $J'$ that never idles when $s_F > 0$. We now show that every job is processed only at the minimum speed of $s_F$ between its release time and deadline.

For every $j \in P$, the algorithm processed $j$ at speed $\mu_j$ before $t$. By the definition of $\mu_j$ and (6), we have $s_F(\tau) \ge \mu_j$ for $r_j < \tau < d_j$. Thus $\mu_j$ is the minimum speed over this interval. If $j \in P \cap R$, the final checks also give $s_F(d_j^-) = \mu_j$, so the construction processes its remaining work at the same speed.

For $j \in R \setminus P$, no work was processed before $t$, and the final check gives $s_F(d_j^-) \le \mu_j$. Before $t$, the speed within the job's release-to-deadline interval is at least $\mu_j$. After $t$, $s_F$ does not increase, so its minimum before $d_j$ is $s_F(d_j^-)$. Therefore $s_F(d_j^-)$ is the minimum over the entire interval $(r_j, d_j)$, and this is exactly the speed at which the construction processes $j$.

Each added job is processed at speed $v$, and $s_F$ equals $v$ throughout the interval between its release time and deadline. Thus it also satisfies the required condition. Lemma E.2 now gives $s_F = \mathrm{OPT}(J')$, so $s_F \in \Sigma_t$. Removing the added jobs from this execution leaves a feasible execution of $J$ in which every job is processed only at the minimum speed of $s_F$ between its release time and deadline. □

Consequently, EDF on $J$ at speed $s_F$ is feasible as well, by the EDF argument in Lemma E.1. Finally, we put everything together and get the following desired results:

**Theorem E.5.** *Algorithm 7 returns $\emptyset$ if and only if $\Sigma_t = \emptyset$. Otherwise, it returns $s_F \in \Sigma_t$ such that*

$$s_F \le s \quad \textit{for every } s \in \Sigma_t, \qquad s_F = \arg\min_{s \in \Sigma_t} \mathrm{cost}(s).$$

*The minimum-cost speed function is unique, apart from endpoint values.*

*Proof.* Lemma E.3 rules out rejection when $\Sigma_t$ is nonempty, and Lemma E.4 rules out acceptance when it is empty. The pointwise speed bound implies the cost bound. □

**Corollary E.6.** *For $t_1 < t_2$ with $\Sigma_{t_2} \neq \emptyset$, let $s_F^{t_1}$ and $s_F^{t_2}$ be the respective outputs. Then*

$$s_F^{t_1}(\tau) \le s_F^{t_2}(\tau) \quad \textit{for all } \tau.$$

*In particular, $s_F^{t_1}(\tau) \le \hat{s}(\tau)$ for $t_1 < \tau \le t_2$.*

*Proof.* Every instance and schedule consistent with the observations at $t_2$ is consistent with those at $t_1$, so $\Sigma_{t_2} \subseteq \Sigma_{t_1}$. Apply Theorem E.5 at $t_1$ to $s_F^{t_2} \in \Sigma_{t_1}$. The last assertion uses $s_F^{t_2} = \hat{s}$ through $t_2$. □

### E.3 Running time

**Theorem E.7.** *Let $n = |J|$ and let $K$ be the number of constant pieces in the observed prediction. One call to* FORECAST *uses*

$$O\big(nK + n^2 + (n+K)\log(n+K+2)\big)$$

*arithmetic operations. Its output has $K + O(n+1)$ constant pieces.*

*Proof.* An EDF replay splits the observed prefix at prediction changes, releases, deadlines, and completions. There are $O(n+K)$ such events. Sorting them and maintaining pending jobs by deadline takes $O((n+K)\log(n+K+2))$ operations. The replay records residual work, missed deadlines, positive-speed idling, and each job's execution speeds. Scanning the $K$ prediction pieces for each window computes all $\mu_j$ in $O(nK)$ operations; the recorded execution speeds then give the remaining prefix check in $O(n+K)$ operations.

Group the residual loads by deadline and form their prefix sums. Every numerator in (3) is then a difference of two prefix sums. Enumerating pairs $(i,k)$ computes all $B_i$ in $O(n^2)$ operations. Grouping the $\mu_j$ of started jobs by deadline and taking suffix maxima computes all $F_i$ in $O(n)$ additional operations. The construction loop and the final checks take $O(n)$ operations after each job is associated with its deadline index. The output consists of the observed $K$ pieces, at most $m \le n$ future deadline intervals, and a zero tail.

□

## 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 two contributions stated in Section 1, namely the speed prediction models and the two algorithms with the corresponding consistency, robustness, and smoothness guarantees, are formally established in Sections 3 and 4, in Theorems 3.4 and 4.1. The PAC-learnability claim from the introduction is established in Section 5. The empirical claims are supported by the experiments in Section 6.

   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: Limitations are discussed throughout the paper. For example, in Section 1, we explicitly state that finding a smooth algorithm for the dynamic machine-speed prediction model remains an open problem. The setting is clearly stated in Section 2, and the assumptions used in the PAC-learnability analysis are stated in Section 5, for example, the discretized release times/deadlines.

   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: All theorems and lemmas are formally stated, with assumptions made explicit. Proof sketches accompany the main results in Sections 3 and 4, with full proofs deferred to Appendix A, Appendix B, Appendix D, and Appendix E.

   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 paper provides sufficient details, including dataset sources, parameter values, and descriptions of algorithmic implementations, to reproduce the experiments. Algorithms 1 and 2 are fully described in pseudocode, including all of their subroutines formally presented as well. Section 6.1 specifies all parameter choices, the deadline distribution, and how the three speed prediction types are derived from a common predicted instance $\hat{J}$. The Brightkite dataset is publicly available from the SNAP repository, and our preprocessing of the data is fully described in that section as well. Implementation details specific to Algorithm 1 are given in Appendix C.1. The OA and YDS subroutines are taken without modification from (9) and (25), and the TPE baseline is taken from (7).

   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: [No]

   Justification: Our experiments serve as a proof of validity for the theoretical guarantees established in Sections 3 and 4, which stand on their own as the main theoretical contribution. We do not release code with this submission, but we are happy to release it upon publication.

   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 6.1 specifies all parameter choices and the procedure used to derive the three different prediction types from a common predicted instance $\hat{J}$. Appendix C.2 reports an additional sensitivity analysis over $\lambda$.

   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: [No]

   Justification: For the synthetic experiments, each reported point is the average energy/competitive ratio over 20 i.i.d. instances drawn from the same generative process. For the real data experiments, Table 2 reports the average, minimum, maximum, and standard deviation of the energy across the 851 days of the Brightkite trace.

   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: [Yes]

   Justification: All experiments are lightweight numerical simulations of our algorithms and were run on a single standard laptop CPU. Each synthetic instance with $n = 100$ jobs runs in under a second, and the full Brightkite sweep over 851 days completes in under an hour. We report this in Appendix C.2.

   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: We have reviewed the NeurIPS Code of Ethics and conform to it. The paper presents theoretical and empirical work on online scheduling algorithms; it does not involve human subjects, personally identifying data, or deployed systems. The Brightkite dataset is publicly available and used here only as an aggregate workload trace.

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: [Yes]

Justification: As discussed in Section 1 the paper is motivated by the growing energy demand of data centers and AI workloads, and the algorithms aim to reduce energy consumption while meeting deadlines. The potential positive societal impact, namely reduced energy use and associated emissions in computing infrastructure, is therefore aligned with the paper's stated goals.

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 any pretrained models, generative models, or scraped data.

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: [Yes]

    Justification: The Brightkite dataset is cited to its original source (14) and accessed via the publicly available SNAP repository (`https://snap.stanford.edu/data/loc-brightkite.html`). The Optimal Available algorithm and the YDS algorithm we use as subroutines are properly attributed to (9) and (25), and the TPE baseline is attributed to (7).

    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 only new asset released with the paper is our implementation of the proposed algorithms (Algorithms 1, 2, and the subroutines) and the experiment scripts.

    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 or any research with human subjects.

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 research with human subjects, so IRB approval is 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: [N/A]

Justification: LLMs are not part of the methodology of this paper.

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.