Algorithms with Prediction Portfolios

Anonymous Author(s)
Affiliation
Address
email

Abstract

The research area of algorithms with predictions has seen recent success showing how to incorporate machine learning into algorithm design to improve performance when the predictions are correct, while retaining worst-case guarantees when they are not. Most previous work has assumed that the algorithm has access to a single predictor. However, in practice, there are many machine learning methods available, often with incomparable generalization guarantees, making it hard to pick a best method a priori. In this work we consider scenarios where multiple predictors are available to the algorithm and the question is how to best utilize them.

Ideally, we would like the algorithm's performance to depend on the quality of the *best* predictor. However, utilizing more predictions comes with a cost, since we now have to identify which prediction is best. We study the use of multiple predictors for a number of fundamental problems, including matching, load balancing, and non-clairvoyant scheduling, which have been well-studied in the single predictor setting. For each of these problems we introduce new algorithms that take advantage of multiple predictors, and prove bounds on the resulting performance.

1 Introduction

An exciting recent line of research attempts to go beyond traditional worst-case analysis of algorithms by equipping algorithms with *machine-learned predictions*. The hope is that these predictions allow the algorithm to circumvent worst case lower bounds when the predictions are good, and approximately match them otherwise. The precise definitions and guarantees vary with different settings, but there have been significant successes in applying this framework for many different algorithmic problems, ranging from very general online problems to classical graph algorithms (see Section 1.2 for a more detailed discussion of related work, and [29] for a survey). In all of these settings it turns out to be possible to define a "prediction" where the "quality" of the algorithm (competitive ratio, running time, etc.) depends the "error" of the prediction. Moreover, in at least some of these settings, it has been further shown that this prediction is actually learnable with a small number of samples, usually via standard ERM methods [15].

Previous work has shown the power of accurate predictions, and there are numerous examples showing improved performance in both theory and practice. However, developing accurate predictors remains an art, and a single predictor may not capture all of the subtleties of the instance space. Recently, researchers have turned to working with *portfolios of predictors*: instead of training a single model, we instead train multiple models, with the hope that one of them will give good guarantees.

It is easy to see why the best predictor in a portfolio may be *significantly* better than a one-size fits all predictor. First, many of the modern machine learning methods come with a slew of hyperparameters that require tuning. Learning rate, mini-batch size, optimizer choice, all of these have significant impact on the quality of the final solution. Instead of commiting to a single setting, one can instead try to cover the parameter space, with the hope that some of the predictors will generalize better than

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

others. Second, problem instances themselves may come from complex distributions, consisting of many latent groups or clusters. A single predictor is forced to perform well on average, whereas multiple predictors can be made to "specialize" to each cluster.

In order to take advantage of the increased accuracy provided by the portfolio approach, we must adapt algorithms with predictions to take advantage of multiple predictions. To recoup the gains in performance, the algorithm must perform as if equipped with the best predictor, auto-tuning to use the best one available in the portfolio. However, it is easy to see that there should be a cost as the size of the portfolio grows. In the extreme, one can add every possible prediction to the portfolio, providing no additional information, yet now requiring high performance from the algorithm. Therefore, we must aim to minimize the dependence on the number of predictions in the portfolio.

We remark that the high level set up may be reminiscent of expert- or bandit-learning literature. However, there is a critical distinction. In expert and bandit learning, we are given a sequence of problem instances, and the goal is to compete (minimize regret) with respect to the best prediction *averaged* over the whole sequence. On the other hand, in our setup, we aim to compete with the best predictor on a *per-instance* basis.

Previous work on multiple predictions. Anand, Ge, Kumar, and Panigrahi recently initiated the study of algorithms with multiple learned predictions in [6], proving strong bounds for important online covering problems including online set cover, weighted caching, and online facility location. However, their techniques and results are limited to online covering problems. Moreover, they do not discuss the learning aspects at all: they simply assume that they are given k predictions, and their goal is to have competitive ratios that are based on the minimum error of any of the k predictions. (They actually compete against a stronger dynamic benchmark, but for our purposes this distinction is not important).

On the other hand Balcan et al. [12] look at this problem through a data driven algorithm lens and study the sample complexity and generalization error of working with k (as opposed to 1) parameter settings. The main difference from our work is that they also aim learn a selector, which selects one of the k parameters *prior* to beginning to solve the problem instance. In contrast, in this work we make the selection during the course of the algorithm, and sometimes switch back and forth while honing in on the best predictor.

1.1 Our Results and Contributions

In this paper we study three fundamental problems, min-cost perfect matching, online load balancing, and non-clairvoyant scheduling for total completion time, in this new setting. Each of these has seen significant success in the single-prediction model but is not covered by previous multiple-prediction frameworks.

For each of these we develop algorithms whose performance depends on the error of the *best* prediction, and explore the effect of the number of predictions, k. Surprisingly, in the case of matching and scheduling we show that using a limited number of multiple predictions is essentially free, and has *no* asymptotic impact on the algorithm's performance. For load balancing, on the other hand, we show that the cost of multiple predictions grows *logarithmically* with k, again implying a tangible benefit of using multiple predictions. We now describe these in more detail.

Min-Cost Perfect Matching. We warm up to showcase our approach with the classical min-cost perfect matching problem in Section 3. This problem was recently studied by [14, 15] to show that it is possible to use learned predictions to improve *running times* of classical optimization problems. In particular, [15] showed it is possible to speed up the classical Hungarian algorithm by predicting dual values, and moreover that it is possible to efficiently (PAC-)learn the best duals. We show that simple modifications of their ideas lead to similar results for multiple predictions. Interestingly, we show that as long as $k \leq O(\sqrt{n})$, the extra "cost" (running time) of using k predictions is negligible compared to the cost of using a single prediction, so we can use up to $\sqrt{n}$ predictions "for free" while still getting running time depending on the best of these predictions. Moreover, since in this setting running time is paramount, we go beyond sample complexity to show that it is also computationally efficient to learn the best k predictions.

Online Load Balancing with Restricted Assignments. We continue in Section 4 with the fundamental load balancing problem. In this problem there are m machines, and n jobs which appear in online fashion. Each job has a size, and a subset of machines that it can be assigned to. The goal is to minimize the maximum machine load (i.e., the makespan). This problem has been studied extensively in the traditional scheduling and online algorithms literature, and recently it has also been the subject of significant study given a single prediction [22–24]. In particular, Lattanzi, Lavastida, Moseley, and Vassilvitskii [22] showed that there exist per machine "weights" and an allocation function so that the competitive ratio of the algorithm depends logarithmically on the maximum error of the predictions. We show that one can use k predictions and incur an additional $O(\log k)$ factor in the competitive ratio, while being competitive with the error of the *best* prediction. Additionally, we show that learning the best k predicted weights (in a PAC sense) can be done efficiently.

Non Clairvoyant Scheduling Finally, in Section 5 we move to the most technically complex part of this paper. We study the problem of scheduling n jobs on a single machine, where all jobs are released at time 0, but where we do not learn the length of a job until it actually completes (the *non-clairvoyant* model). Our objective is to minimize the sum of completion times. This problem has been studied extensively, both with and without predictions [20, 26, 31, 32]. Most recently, Lindermayr and Megow [26] suggested that we use an *ordering* as the prediction (as opposed to the more obvious prediction of job sizes), and use the difference between the cost induced by the predicted ordering and the cost induced by the instance-optimal ordering as the notion of "error". In this case, simply following the predicted ordering yields an algorithm with error equal to the prediction error.

We extend this to the multiple prediction setting, which turns out to be surprisingly challenging. The algorithm of [26] is quite simple: follow the ordering given by the prediction (and run a 2-competitive algorithm in parallel to obtain a worst-case backstop). But we obviously cannot do this when we are given multiple orderings! So we must design an algorithm which considers all k predictions to build a schedule that has error comparable to the error of the *best* one. Slightly more formally, we prove that we can get sum of completion times bounded by at most $(1+\epsilon)\mathrm{OPT}$ plus $\mathrm{poly}(1/\epsilon)$ times the error of the best prediction, under the mild assumption that no set of at most $\log\log n$ jobs has a large contribution to OPT.

To do this, we first use sampling techniques similar to those of [20] to estimate the size of the approximately ϵn'th smallest job without incurring much cost. We then use even more sampling and partial processing to determine for each prediction whether its ϵn prefix has many jobs that should appear later (a bad sequence) or has very few jobs that should not be in the prefix (a good sequence). If all sequences are bad then every prediction has large error, so we can use a round robin schedule and charge the cost to the prediction error. Otherwise, we choose one of the good orderings and follow it for its ϵn prefix (being careful to handle outliers). We then recurse on the remaining jobs.

1.2 Related Work

As discussed, the most directly related papers are Anand et al. [6] and Balcan, Sandholm, and Vitercik [12]; these give the two approaches (multiple predictions and portfolio-based algorithm selection) that are most similar to our setting. The single prediction version of min-cost bipartite matching was studied in [14, 15], the single prediction version of our load balancing problem was considered by [22–24] (and a different though related load balancing problem was considered by [4]), and the single prediction version of our scheduling problem was considered by [26] with the same prediction that we use (an ordering) and earlier with different predictions by [20, 32]. Online scheduling with estimates of the true processing times was considered in [9, 10].

More generally, there has been an enormous amount of recent progress on algorithms with predictions. This is particularly true for online algorithms, where the basic setup was formalized by [27] in the context of caching. For example, the problems considered include caching [21, 27, 33], secretary problems [7, 17], ski rental [5, 32], and set cover [13]. There has also been recent work on going beyond traditional online algorithms, including work on running times [14, 15], algorithmic game theory [2, 18, 28], and streaming algorithms [1, 16, 19].

The above is only a small sample of the work on algorithms with predictions. We refer the interested reader to a recent survey [29], as well as a recently set up website which maintains a list of papers in the area [25].

2 Learnability

When designing new methods in the algorithms with predictions setting, the predictions under consideration must satisfy two constraints. First, they should be useful to the algorithm, so that using the predictions allows the algorithm to achieve better running time, competitive ratio, or some other performance measure. Second, they must be *learnable*; that is it must be feasible to find good predictions given a set of problem instances.

To rigorously prove learnability, we follow previous work [11, 15, 30] and focus on proving a bound on the sample complexity of finding the best predictions that generalize.

Our main result shows that for a given problem, the pseudo-dimension of finding k predictions is $\tilde{O}(k)$[1] factor larger than that for finding a single best predictor. We state the formal Theorem below, but defer the proof to the supplementary material.

Theorem 2.1. *Let $\mathcal{F}$ be a class of functions $f : \mathcal{X} \to \mathbb{R}$ with pseudo-dimension d and let $\mathcal{F}^k := \{F(x) = \min_{\ell \in [k]} f^\ell(x) \mid f^1, f^2, \ldots, f^k \in \mathcal{F}\}$. Then the pseudo-dimension of $\mathcal{F}^k$ is at most $\tilde{O}(dk)$.*

Note that this directly implies that the sample complexity when looking for k predictions is a factor of k larger than that of a single predictor.

3 Minimum Cost Bipartite Matching with Predicted Duals

In this section we study the minimum cost bipartite matching problem with multiple predictions. The case of a single prediction has been considered recently [14, 15], where they used dual values as a prediction and showed that the classical Hungarian algorithm could be sped up by using appropriately learned dual values. Our goal in this section is to extend these results to multiple predictions, i.e., multiple duals. In particular, in Section 3.2 we show that we can use k duals and get running time comparable to the time we would have spent if we used the single best of them in the algorithm of [15], with no loss whatsoever if k is at most $O(\sqrt{n})$. Then in Section 3.3 we show that k predictions can be learned with not too many more samples (or running time) than learning a single prediction.

3.1 Problem Definition and Predicted Dual Variables

In minimum cost bipartite matching there is a bipartite graph $G = (V, E)$ (letting $n = |V|$ and $m = |E|$) with edge costs $c \in \mathbb{Z}^E$. The objective is to output a perfect matching $M \subseteq E$ which minimizes the cost $c(M) := \sum_{e \in E} c_e$. This problem is exactly captured by the following primal and dual linear programming formulations.

$$\begin{aligned} \min \quad & \sum_{e \in E} c_e x_e && \text{(MWPM-P)} \\ & \sum_{e \in N(i)} x_e = 1 && \forall i \in V \\ & x_e \geq 0 && \forall e \in E \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{i \in V} y_i && \text{(MWPM-D)} \\ & y_i + y_j \leq c_e && \forall e = ij \in E \end{aligned}$$

Recently, Dinitz et al. [15] studied initializing the Hungarian algorithm with a prediction $\hat{y}$ of the optimal dual solution y^*. They propose an algorithm which operates in two steps. First, the predicted dual solution $\hat{y}$ may not be feasible, so they give an $O(n + m)$ time algorithm which recovers feasibility (which we refer to as Make-Feasible). Second, the now-feasible dual solution is used in a primal-dual algorithm such as the Hungarian algorithm (which we refer to as Primal-Dual) and they show that the running time depends on the ℓ_1 error in the predicted solution. In addition to this they show that learning a good initial dual solution is computationally efficient with low sample complexity. More formally, they proved the following theorems.

Theorem 3.1 (Dinitz et al. [15])**.** *Let (G, c) be an instance of minimum cost bipartite matching and $\hat{y}$ be a prediction of an optimal dual solution y^*. There exists an algorithm which returns an optimal solution and runs in time $O(m\sqrt{n} \cdot \|y^* - \hat{y}\|_1)$.*

Theorem 3.2 (Dinitz et al. [15])**.** *Let $\mathcal{D}$ be an unknown distribution over instances (G, c) on n vertices and let $y^*(G, c)$ be an optimal dual solution for the given instance. Given S independent*

[1] $\tilde{O}(\cdot)$ suppresses logarithmic factors

samples from $\mathcal{D}$, there is a polynomial time algorithm that outputs a solution $\hat{y}$ such that

$$\mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\|y^*(G,c)-\hat{y}\|_1\right] \le \min_y \mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\|y^*(G,c)-y\|_1\right]+\epsilon$$

with probability $1-\delta$ where $S=\text{poly}(n,\frac{1}{\epsilon},\frac{1}{\delta})$.

3.2 Using k Predicted Dual Solutions Efficiently

Given k predicted dual solutions $\hat{y}^1,\hat{y}^2,\ldots,\hat{y}^k$, we would like to efficiently determine which solution has the minimum error for the given problem instance. Note that the predicted solutions may still be infeasible and that we do not know the target optimal dual solution y^*. We propose the following simple algorithm which takes as input k predicted solutions and whose running time depends only on the ℓ_1 error of the *best* predicted solution. First, we make each predicted solution feasible, just as before. Next, we select the (now-feasible) dual solution with highest dual objective value and proceed running the primal-dual algorithm with only that solution. See Algorithm 1 for pseudo-code.

We have the following result concerning Algorithm 1. To interpret this result, note that the cost for increasing the number of predictions is $O(k(n+m))$, which will be dominated by the $m\sqrt{n}$ term we pay for running the Hungarian algorithm unless k is extremely large (certainly larger than $\sqrt{n}$) or there is a prediction with 0 error (which is highly unlikely). Hence we can reap the benefit of a large number of predictions "for free".

Algorithm 1 Minimum cost matching with k predicted dual solutions

```
procedure k-PredictedPrimal-Dual(G, c, ŷ^1, ŷ^2, ..., ŷ^k)
    for ℓ ∈ [k] do
        y^ℓ ← MakeFeasible(G, c, ŷ^ℓ)
    end for
    ℓ' ← arg max_{ℓ∈[k]} Σ_{i∈V} y_i^ℓ
    M ← Primal-Dual(G, c, y^{ℓ'})
    Return M
end procedure
```

Theorem 3.3. *Let (G,c) be a minimum cost bipartite matching instance and let $\hat{y}^1,\hat{y}^2,\ldots,\hat{y}^k$ be predicted dual solutions. Algorithm 1 returns an optimal solution and runs in time $O(k(n+m)+m\sqrt{n}\cdot\min_{\ell\in[k]}\|y^*-\hat{y}^\ell\|_1)$.*

We defer the proof to the supplementary material. But correctness is essentially direct from [15], and the running time requires just a simple modification of the analysis of [15].

3.3 Learning k Predicted Dual Solutions

Next we extend Theorem 3.2 to the setting where we output k predictions. Let $\mathcal{D}$ be a distribution over problem instances (G,c) on n vertices. We show that we can find the best set of k predictions. More formally, we prove the following theorem.

Theorem 3.4. *Let $\mathcal{D}$ be an unknown distribution over instances (G,c) on n vertices and let $y^*(G,c)$ be an optimal dual solution for the given instance. Given S independent samples from $\mathcal{D}$, there is a polynomial time algorithm that outputs a k solutions $\hat{y}^1,\hat{y}^2,\ldots,\hat{y}^k$ such that*

$$\mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\min_{\ell\in[k]}\|y^*(G,c)-\hat{y}^\ell\|_1\right] \le O(1)\cdot\min_{y^1,y^2,\ldots,y^k}\mathbb{E}_{(G,c)\sim\mathcal{D}}\left[\min_{\ell\in[k]}\|y^*(G,c)-y^\ell\|_1\right]+\epsilon$$

with probability $1-\delta$ where $S=\text{poly}(n,k,\frac{1}{\epsilon},\frac{1}{\delta})$.

The proof of this theorem can be found in the supplementary material, but it is straightforward. The sample complexity is due to combining Theorem 2.1 with Theorem 3.2 (or more precisely, with the pseudo-dimension bound which implies Theorem 3.2). The $O(1)$-approximation factor and polynomial running time is from the observation that the ERM problem in this setting is just k-median.

4 Online Load Balancing with Predicted Machine Weights

We now apply our framework to online load balancing with restricted assignments. In particular, we consider proportional weights, which have been considered in prior work [22–24]. Informally, we

show in Section 4.2 that if β is the cost of the *best* of the k predictions, then even without knowing a priori which prediction is best, we get cost of $O(\beta \log k)$. Then in Section 4.3 we show that it does not take many samples to actually learn the best k predictions.

4.1 Problem Definition and Proportional Weights

In online load balancing with restricted assignments there is a sequence of n jobs which must be assigned to m machines in an online fashion. Upon seeing job j, the online algorithm observes its size $p_j > 0$ and a neighborhood $N(j) \subseteq [m]$ of *feasible* machines. The algorithm must then choose some feasible machine $i \in N(j)$ to irrevocably assign the job to before seeing any more jobs in the sequence. We also consider fractional assignments, i.e. vectors belonging to the set $X = \{x \in \mathbb{R}_+^{m \times n} \mid \forall j \in [n], \sum_i x_{ij} = 1, \text{ and } x_{ij} = 0 \iff i \notin N(j)\}$.

Prior work studied the application of proportional weights[3, 22–24]. Intuitively, a prediction in this setting is a weighting of *machines*, which then implies an online assignment, which is shown to be near-optimal. Slightly more formally, suppose that we are given weights w_i for each machine i. Then we say that each job j is *fractionally* assigned to machine i to a fractional amount of $\frac{w_i}{\sum_{i' \in N(j)} w_{i'}}$. Notice that given weights, this also gives an online assignment. It is known that there exist weights for any instance where the fractional solution has a near optimal makespan, even though there are only m "degree of freedom" in the weights compared to mn in an assignment. That is, for all machines i, $\sum_{j \in [n]} p_j \cdot \frac{w_i}{\sum_{i' \in N(j)} w_{i'}}$ is at most a $(1+\epsilon)$ factor larger than the optimal makespan for any constant $\epsilon > 0$ [3, 22].

Let w^* be a set of near optimal weights for a given instance. Lattanzi et al. [22] showed the following theorem:

Theorem 4.1. *Given predicted weights $\hat{w}$, there is an online fractional algorithm which has makespan $O(\log(\eta(\hat{w}, w^*)\mathrm{OPT})$, where $\eta(\hat{w}, w^*) := \max_{i \in [m]} \max(\frac{\hat{w}_i}{w_i^*}, \frac{w_i^*}{\hat{w}_i})$ to be the error in the prediction.*

Moreover, this fractional assignment can be converted online to an integral assignment while losing only an $O(\log \log m)$ factor in the makespan [22, 24].

4.2 Combining Fractional Solutions Online

Given k different predicted weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$, we want to give an algorithm which is competitive against the *minimum* error among the predicted weights, i.e. we want the competitiveness to depend upon $\eta_{\min} := \min_{\ell \in [k]} \eta(\hat{w}^\ell, w^*)$.

The challenge is that we do not know up front which $\ell \in [k]$ will yield the smallest error, but instead learn this in hindsight. We consider the following approach to address this. For each $\ell \in [k]$, let x^ℓ, be the resulting fractional assignment from applying the fractional online algorithm due to [22] with weights $\hat{w}^\ell$. This fractional assignment is revealed one job at a time.

We give an algorithm which is $O(\log k)$-competitive against any collection of k fractional assignments which are revealed online. Moreover, our result applies to the unrelated machines setting, in which each job has a collection of machine-dependent sizes $\{p_{ij}\}_{i \in [m]}$. The algorithm is based on the doubling trick and is similar to results due to [8] which apply to metrical task systems. Let $\beta := \min_{\ell \in [k]} \max_i \sum_j p_{ij} x_{ij}^\ell$ be the best fractional makespan in hindsight. As in previous work, our algorithm is assumed to know β, an assumption that can be removed [22]. At a high level, our algorithm maintains a set $A \subseteq [k]$ of solutions which are good with respect to the current value of β, averaging among these. See Algorithm 2 for a detailed description. We have the following theorem.

Theorem 4.2. *Let $x^1, x^2, \ldots, x^k$ be fractional assignments which are revealed online. If Algorithm 2 is run with $\beta := \min_{\ell \in [k]} \max_i \sum_j p_{ij} x_{ij}^\ell$, then it yields a solution of cost $O(\log k) \cdot \beta$ and never reaches the fail state (line 7 in Algorithm 2).*

Let $\beta_\ell = \max_i \sum_j p_{ij} x_{ij}^\ell$ and OPT be the optimal makespan. Theorem 4.1 shows that $\beta_\ell \leq O(\log \eta_\ell)\mathrm{OPT}$. The following corollary is then immediate:

Corollary 4.3. *Let $w^1, w^2, \ldots, w^k$ be the predicted weights with errors $\eta^1, \eta^2, \ldots, \eta^k$. Then Algorithm 2 returns a fractional assignment with makespan at most $\mathrm{OPT} \cdot O(\log k) \cdot \min_{\ell \in [k]} \log(\eta^\ell)$.*

Algorithm 2 Algorithm for combining fractional solutions online for load balancing.

procedure COMBINE-LOADBALANCING(β)
  $A \leftarrow [k]$ ▷ Initially all solutions are good
  for each job j **do**
    Receive the assignments $x^1, x^2, \ldots, x^k$
    $A(j,\beta) \leftarrow \{\ell \in A \mid \forall i \in [m], x^\ell_{ij} > 0 \implies p_{ij} x^\ell_{ij} \leq \beta\}$
    if $A = \emptyset$ or $A(j,\beta) = \emptyset$ **then**
      Return "Fail"
    end if
    $\forall i \in [m], x_{ij} \leftarrow \frac{1}{|A(j,\beta)|} \sum_{\ell \in S(j,\beta)} x^\ell_{ij}$
    $A \leftarrow \{\ell \in S \mid \max_{i \in [m]} \sum_{j' \leq j} p_{ij'} x^\ell_{ij'} > \beta\}$ ▷ Bad solutions w.r.t. β
    $A \leftarrow A \setminus B$
  end for
end procedure

We defer the proof of Theorem 4.2 to the Supplementary material.

4.3 Learning k Predicted Weight Vectors

We now turn to the question of showing how to learn k different predicted weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$. Recall that there is an unknown distribution $\mathcal{D}$ over sets of n jobs from which we receive independent samples $J_1, J_2, \ldots, J_S$. Our goal is to show that we can efficiently learn (in terms of sample complexity) k predicted weight vectors to minimize the expected minimum error. Let $w^*(J)$ be the correct weight vector for instance J and let $\eta(w, w') = \max_{i \in [m]} \max(\frac{w_i}{w'_i}, \frac{w'_i}{w_i})$ be the error between a pair of weight vectors. We have the following result.

Theorem 4.4. *Let $\mathcal{D}$ be an unknown distribution over restricted assignment instances on n jobs and let $w^*(J)$ be a set of good weights for instance J. Given S independent samples from $\mathcal{D}$, there is a polynomial time algorithm that outputs k weight vectors $\hat{w}^1, \hat{w}^2, \ldots, \hat{w}^k$ such that $\mathbb{E}_{J \sim \mathcal{D}} \left[\min_{\ell \in [k]} \log(\eta(\hat{w}^\ell, w^*(J))\right] \leq O(1) \cdot \min_{w^1, w^2, \ldots, w^k} \mathbb{E}\left[\min_{\ell \in [k]} \log(\eta(w^\ell, w^*(J))\right] + \epsilon$ with probability $1 - \delta$, where $S = \mathrm{poly}(m, k, \frac{1}{\epsilon}, \frac{1}{\delta})$*

The proof of Theorem 4.4 is deferred to the Supplementary material, but we note that to get a polynomial time algorithm we carry out an interesting reduction to k-median clustering. Namely, we show that the function $d(w, w') := \log(\eta(w, w'))$ satisfies the triangle inequality and thus forms a metric space.

5 Scheduling with Predicted Permutations

In this problem there are n jobs, indexed by $1, 2, \ldots, n$, to be scheduled on a single machine. We assume that they are all available at time 0. Job j has size p_j and needs to get processed for p_j time units to complete. If all job sizes are known a priori, Shortest Job First (or equivalently Shortest Remaining Time First), which processes jobs in non-decreasing order of their size, is known to be optimal for minimizing total completion time. We assume that the true value of p_j is unknown and is revealed only when the job completes, i.e. the *non-clairvoyant* setting. In the non-clairvoyant setting, it is known that Round-Robin (which processes all alive jobs equally) is 2-competitive and that this is the best competitive ratio one can hope for [31].

We study this basic scheduling problem assuming certain predictions are available for use. Following the recent work by Lindermayr and Megow [26], we will assume that we are given k orderings/sequences as prediction, $\{\sigma_\ell\}_{\ell \in [k]}$. Each σ_ℓ is a permutation of $J := [n]$. Intuitively, it suggests an ordering in which jobs should be processed. This prediction is inspired by the aforementioned Shortest Job First (SJF) as an optimum schedule can be described as an ordering of jobs, specifically increasing order of job sizes.

For each σ_ℓ, its error is measured as $\eta(J, \sigma_\ell) := \textsc{cost}(J, \sigma_\ell) - \mathrm{OPT}(J)$, where $\textsc{cost}(J, \sigma_\ell)$ denotes the objective of the schedule where jobs are processed in the order of σ_ℓ and $\mathrm{OPT}(J)$ denotes the optimum objective. We may drop J from notation when it is clear from the context.

As observed in [26], the error can be expressed as $\eta(J, \sigma_\ell) = \sum_{i<j \in J} I^\ell_{i,j} \cdot |p_i - p_j|$, where $I^\ell_{i,j}$ is an indicator variable that has value 1 if and only if σ_ℓ predicts the pairwise ordering of i and j incorrectly. That is, if $p_i < p_j$, then the optimal schedule would process i before j; here $I^\ell_{i,j} = 1$ iff $i \succ_{\sigma_\ell} j$.

As discussed in [26], this error measure satisfies two desired properties, monotonicity and Lipschitzness, which were formalized in [20].

Our main result is the following.

Theorem 5.1. *Consider a constant $\epsilon > 0$. Suppose that for any $S \subseteq J$ with $|S| = \Theta(\frac{1}{\epsilon^3}(\log\log n + \log k + \log(1/\epsilon)))$, we have $\mathrm{OPT}(S) \leq c\epsilon \cdot \mathrm{OPT}(J)$ for some small absolute constant c. Then, there exists a randomized algorithm that yields a schedule whose expected total completion time is at most $(1+\epsilon)\mathrm{OPT} + (1+\epsilon)\frac{1}{\epsilon^5}\eta(J, \sigma_\ell)$ for all $\ell \in [k]$.*

As a corollary, by running our algorithm with $1-\epsilon$ processing speed and simultaneously running Round-Robin with the remaining ϵ of the speed, the cost increases by a factor of at most $\frac{1}{1-\epsilon}$ while the resulting hybrid algorithm is $2/\epsilon$-competitive.[2]

5.1 Algorithm

To make our presentation more transparent we will first round job sizes. Formally, we choose ρ uniformly at random from $[0, 1)$. Then, round up each job j's size to the closest number of the form $(1+\epsilon)^{\rho+t}$ for some integer t. Then, we scale down all job sizes by $(1+\epsilon)^\rho$ factor. We will present our algorithm and analysis assuming that every job has a size equal to a power of $(1+\epsilon)$. In the supplementary we show how to remove this assumption without increasing our algorithm's objective by more than $1+\epsilon$ factor in expectation.

We first present the following algorithm that achieves Theorem 5.1 with $|S| = \Theta(\frac{1}{\epsilon^3}(\log n + \log k))$. The improved bound claimed in the theorem needs minor tweaks of the algorithm and analysis and they are deferred to the supplementary material.

Our algorithm runs in rounds. Let J_r be the jobs that complete in round $r \geq 1$. For any subset S of rounds, $J_S := \cup_{r \in S} J_r$. For example, $J_{\leq r} := J_1 \cup \ldots \cup J_r$. Let $n_r := |J_{\geq r}| = n - |J_{<r}|$ denote the number of alive jobs at the beginning of round r.

Fix the beginning of round r. The algorithm processes the job in the following way for this round. If $n_r \leq \frac{1}{\epsilon^3}(\log n + \log k)$, we run Round-Robin to complete all the remaining jobs, $J_{\geq r}$. This is the last round and it is denoted as round $L+1$. Otherwise, we do the following Steps 1-4:

Step 1. Estimating ϵ-percentile. Roughly speaking, the goal is to estimate the ϵ-percentile of job sizes among the remaining jobs. For a job $j \in J_{\geq r}$, define its rank among $J_{\geq r}$ as the number of jobs no smaller than j in $J_{\geq r}$ breaking ties in an arbitrary yet fixed way. Ideally, we would like to estimate the size of job of rank ϵn_r, but do so only approximately.

The algorithm will find $\tilde{q}_r$ that is the size of a job whose rank lies in $[\epsilon(1-\epsilon)n_r, \epsilon(1+\epsilon)n_r]$. To handle the case that there are many jobs of the same size $\tilde{q}_r$, we estimate y_r the number of jobs no bigger than $\tilde{q}_r$; let $\tilde{y}_r$ denote our estimate of y_r. We will show how we can do these estimations without spending much time by sampling some jobs and partially processing them in Round-Robin manner (the proof of the following lemma can be found in the supplementary material.)

Lemma 5.2. *W.h.p. the algorithm can construct estimates $\tilde{q}_r$ and $\tilde{y}_r$ in time at most $O(\tilde{q}_r \frac{1}{\epsilon^2}\log n)$ such that there is a job of size $\tilde{q}_r$ whose rank lies in $[\epsilon(1-\epsilon)n_r, \epsilon(1+\epsilon)n_r]$ and $|\tilde{y}_r - y_r| \leq \epsilon^2 n_r$.*

Step 2. Determining Good and Bad Sequences. Let σ^r_ℓ denote σ_ℓ with all jobs completed in the previous rounds removed and with the relative ordering of the remaining jobs fixed. Let $\sigma^r_{\ell,\epsilon}$ denote the first $\tilde{y}_r$ jobs in the ordering. We say a job j is big if $p_j > \tilde{q}_r$; middle if $p_j = \tilde{q}_r$; small otherwise. Using sampling and partial processing we will approximately distinguish good and bad sequences.

[2] This hybrid algorithm is essentially the preferential time sharing [20, 26, 32]. Formally, we run our algorithm ignoring RR's processing and also run RR ignoring our algorithm; this can be done by a simple simulation. Thus, we construct two schedules concurrently and each job completes at the time when it does in either schedule. This type of algorithms was first used in [32].

Informally σ_ℓ^r is good if $\sigma_{\ell,\epsilon}^r$ has few big jobs and bad if it does many big jobs. The proof of the following lemma can be found in the supplementary material.

Lemma 5.3. *For all $\ell \in [k]$, we can label sequence σ_ℓ^r either good or bad in time at most $O(\tilde{q}_r \frac{1}{\epsilon^2}(\log n + \log k))$ that satisfies the following with high probability: If it is good, $\sigma_{\ell,\epsilon}^r$ has at most $3\epsilon^2 n_r$ big jobs; otherwise $\sigma_{\ell,\epsilon}^r$ has at least $\epsilon^2 n_r$ big jobs.*

Step 3. Job Processing. If all sequences are bad, then we process all jobs, each up to $\tilde{q}_r$ units in an arbitrary order. Otherwise, we process the first $\tilde{y}_r$ jobs in an arbitrary good sequence, in an arbitrary order, each up to $\tilde{q}_r$ units.

Step 4. Updating Sequences. The jobs completed in this round drop from the sequences but the remaining jobs' relative ordering remains fixed in each (sub-)sequence. For simplicity, we assume that partially processed jobs were never processed—this is without loss of generality as this assumption only increases our schedule's objective.

5.2 Analysis of the Algorithm's Performance

We defer the analysis of the above algorithm (the proof of Theorem 5.1) to the supplementary material, as it is quite technical and complex. At a very high level, though, we use the fact that the error in each prediction can be decomposed into pair-wise inversions, and moreover we can partition the inversions into the rounds of the algorithm in which they appear. Then we look at each round, and split into two cases. First, if all sequences are bad then every prediction has large error, so we can simply use Round Robin (which is 2-competitive against OPT) and the cost can be charged to the error of any prediction. Second, if there is a good sequence, then in any good sequence the number of big jobs is small (so we do not spend much time processing them), and we therefore complete almost all of the non-big jobs. Here, we crucially use the fact that we can process the first ϵ fraction of jobs in a sequence in an arbitrary order remaining competitive against the sequence. Finally, we show that all of the additional assumptions and costs (e.g., rounding processing times and the cost due to sampling) only change our performance by a $1+\epsilon$ factor. Getting all of these details right requires much care.

5.3 Learning k Predicted Permutations

Now we show that learning the best k permutations has polynomial sample complexity.

Theorem 5.4. *Let $\mathcal{D}$ be an unknown distribution of instances on n jobs. Given S independent samples from $\mathcal{D}$, there is an algorithm that outputs k permutations $\hat{\sigma}_1, \hat{\sigma}_2, \ldots, \hat{\sigma}_k$ such that $\mathbb{E}_{J\sim\mathcal{D}}\left[\min_{\ell\in[k]} \eta(J, \hat{\sigma}_\ell)\right] \leq \min_{\sigma_1,\sigma_2,\ldots,\sigma_k} \mathbb{E}_{J\sim\mathcal{D}}\left[\min_{\ell\in[k]} \eta(J, \sigma_\ell)\right] + \epsilon$ with probability $1-\delta$, where $S = \mathrm{poly}(n, k, \frac{1}{\epsilon}, \frac{1}{\delta})$.*

Proof. The algorithm is basic ERM, and the polynomial sample complexity follows from Theorem 2.1 and Theorem 20 in Lindermayr and Megow [26]. □

6 Conclusion

Despite the explosive recent work in algorithms with predictions, almost all of this work has assumed only a single prediction. In this paper we study algorithms with *multiple* machine-learned predictions, rather than just one. We study three different problems that have been well-studied in the single prediction setting but not with multiple predictions: faster algorithms for min-cost bipartite matching using learned duals, online load balancing with learned machine weights, and non-clairvoyant scheduling with order predictions. For all of the problems we design algorithms that can utilize multiple predictions, and show sample complexity bounds for learning the best set of k predictions.

Surprisingly, we have shown that in some cases, using multiple predictions is essentially "free." For instance, in the case of min-cost perfect matching examining $k = O(\sqrt{n})$ predictions takes the same amount of time as one round of the Hungarian algorithm, but the number of rounds is determined by the quality of the *best* prediction. In contrast, for load balancing, using k predictions always incurs an $O(\log k)$ cost, so using a constant number of predictions may be best. More generally, studying this trade-off between the cost and the benefit of multiple predictions for other problems remains an interesting and challenging open problem.

References

[1] Anders Aamand, Piotr Indyk, and Ali Vakilian. (learned) frequency estimation algorithms under zipfian distribution. *arXiv preprint arXiv:1908.05198*, 2019.

[2] Priyank Agrawal, Eric Balkanski, Vasilis Gkatzelis, Tingting Ou, and Xizhi Tan. Learning-augmented mechanism design: Leveraging predictions for facility location, 2022. URL https://arxiv.org/abs/2204.01120.

[3] Shipra Agrawal, Morteza Zadimoghaddam, and Vahab S. Mirrokni. Proportional allocation: Simple, distributed, and diverse matching with high entropy. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 99–108. PMLR, 2018. URL http://proceedings.mlr.press/v80/agrawal18b.html.

[4] Sara Ahmadian, Hossein Esfandiari, Vahab Mirrokni, and Binghui Peng. *Robust Load Balancing with Machine Learned Advice*, pages 20–34. 2022. doi: 10.1137/1.9781611977073.2. URL https://epubs.siam.org/doi/abs/10.1137/1.9781611977073.2.

[5] Keerti Anand, Rong Ge, and Debmalya Panigrahi. Customizing ML predictions for online algorithms. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 303–313. PMLR, 2020. URL http://proceedings.mlr.press/v119/anand20a.html.

[6] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. Online algorithms with multiple predictions. In *Proceedings of the 39th International Conference on Machine Learning, ICML 2022*, 2022.

[7] Antonios Antoniadis, Themis Gouleakis, Pieter Kleer, and Pavel Kolev. Secretary and online matching problems with machine learned advice. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/5a378f8490c8d6af8647a753812f6e31-Abstract.html.

[8] Yossi Azar, Andrei Z. Broder, and Mark S. Manasse. On-line choice of on-line algorithms. In Vijaya Ramachandran, editor, *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas, USA*, pages 432–440. ACM/SIAM, 1993. URL http://dl.acm.org/citation.cfm?id=313559.313847.

[9] Yossi Azar, Stefano Leonardi, and Noam Touitou. Flow time scheduling with uncertain processing time. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1070–1080. ACM, 2021. doi: 10.1145/3406325.3451023. URL https://doi.org/10.1145/3406325.3451023.

[10] Yossi Azar, Stefano Leonardi, and Noam Touitou. Distortion-oblivious algorithms for minimizing flow time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 252–274. SIAM, 2022. doi: 10.1137/1.9781611977073.13. URL https://doi.org/10.1137/1.9781611977073.13.

[11] Maria-Florina Balcan, Dan F. DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, and Ellen Vitercik. How much data is sufficient to learn high-performing algorithms? *CoRR*, abs/1908.02894, 2019. URL http://arxiv.org/abs/1908.02894.

[12] Maria-Florina Balcan, Tuomas Sandholm, and Ellen Vitercik. Generalization in portfolio-based algorithm selection. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 12225–12232. AAAI Press, 2021. URL https://ojs.aaai.org/index.php/AAAI/article/view/17451.

[13] Étienne Bamas, Andreas Maggiori, and Ola Svensson. The primal-dual method for learning augmented algorithms. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/e834cb114d33f729dbc9c7fb0c6bb607-Abstract.html.

[14] Justin Y. Chen, Sandeep Silwal, Ali Vakilian, and Fred Zhang. Faster fundamental graph algorithms via learned predictions. *CoRR*, abs/2204.12055, 2022. doi: 10.48550/arXiv.2204.12055. URL https://doi.org/10.48550/arXiv.2204.12055.

[15] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Faster matchings via learned duals. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 10393–10406, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/5616060fb8ae85d93f334e7267307664-Abstract.html.

[16] Elbert Du, Franklyn Wang, and Michael Mitzenmacher. Putting the "learning" into learning-augmented algorithms for frequency estimation. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2860–2869. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/du21d.html.

[17] Paul Dütting, Silvio Lattanzi, Renato Paes Leme, and Sergei Vassilvitskii. Secretaries with advice. *CoRR*, abs/2011.06726, 2020. URL https://arxiv.org/abs/2011.06726.

[18] Vasilis Gkatzelis, Kostas Kollias, Alkmini Sgouritsa, and Xizhi Tan. Improved price of anarchy via predictions, 2022. URL https://arxiv.org/abs/2205.04252.

[19] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *7th International Conference on Learning Representations*, 2019.

[20] Sungjin Im, Ravi Kumar, Mahshid Montazer Qaem, and Manish Purohit. Non-clairvoyant scheduling with predictions. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 285–294. ACM, 2021. doi: 10.1145/3409964.3461790. URL https://doi.org/10.1145/3409964.3461790.

[21] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. Online algorithms for weighted paging with predictions. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 69:1–69:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi: 10.4230/LIPIcs.ICALP.2020.69. URL https://doi.org/10.4230/LIPIcs.ICALP.2020.69.

[22] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. Online scheduling via learned weights. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1859–1877. SIAM, 2020. doi: 10.1137/1.9781611975994.114. URL https://doi.org/10.1137/1.9781611975994.114.

[23] Thomas Lavastida, Benjamin Moseley, R. Ravi, and Chenyang Xu. Learnable and instance-robust predictions for online matching, flows and load balancing. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*, volume 204 of *LIPIcs*, pages 59:1–59:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi: 10.4230/LIPIcs.ESA.2021.59. URL https://doi.org/10.4230/LIPIcs.ESA.2021.59.

[24] Shi Li and Jiayi Xian. Online unrelated machine load balancing with predictions revisited. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on*

Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, pages 6523–6532. PMLR, 2021. URL http://proceedings.mlr.press/v139/li21w.html.

[25] Alexander Lindermayr and Nicole Megow. Algorithms with predictions. https://algorithms-with-predictions.github.io/, 2022.

[26] Alexander Lindermayr and Nicole Megow. Non-clairvoyant scheduling with predictions revisited. *CoRR*, abs/2202.10199, 2022. URL https://arxiv.org/abs/2202.10199.

[27] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 3302–3311, 2018.

[28] Andrés Muñoz Medina and Sergei Vassilvitskii. Revenue optimization with approximate bid predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1856–1864, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[29] Michael Mitzenmacher and Sergei Vassilvitskii. *Algorithms with Predictions*, page 646–662. Cambridge University Press, 2021. doi: 10.1017/9781108637435.037.

[30] Jamie H Morgenstern and Tim Roughgarden. On the pseudo-dimension of nearly optimal auctions. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 136–144. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5766-on-the-pseudo-dimension-of-nearly-optimal-auctions.pdf.

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

[32] Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ml predictions. In *Advances in Neural Information Processing Systems*, pages 9661–9670, 2018.

[33] Dhruv Rohatgi. Near-optimal bounds for online caching with machine learned advice. In *Symposium on Discrete Algorithms (SODA)*, 2020.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [N/A]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [N/A]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]