Competitive Caching with Machine Learned Advice

THODORIS LYKOURIS, Microsoft Research New York City
SERGEI VASSILVITSKII, Google Research New York City

Traditional online algorithms encapsulate decision making under uncertainty, and give ways to hedge against all possible future events, while guaranteeing a nearly optimal solution, as compared to an offline optimum. On the other hand, machine learning algorithms are in the business of extrapolating patterns found in the data to predict the future, and usually come with strong guarantees on the expected generalization error.

In this work, we develop a framework for augmenting online algorithms with a machine learned predictor to achieve competitive ratios that provably improve upon unconditional worst-case lower bounds when the predictor has low error. Our approach treats the predictor as a complete black box and is not dependent on its inner workings or the exact distribution of its errors.

We apply this framework to the traditional caching problem—creating an eviction strategy for a cache of size k. We demonstrate that naively following the oracle's recommendations may lead to very poor performance, even when the average error is quite low. Instead, we show how to modify the Marker algorithm to take into account the predictions and prove that this combined approach achieves a competitive ratio that both (i) decreases as the predictor's error decreases and (ii) is always capped by $O(\log k)$, which can be achieved without any assistance from the predictor. We complement our results with an empirical evaluation of our algorithm on real-world datasets and show that it performs well empirically even when using simple off-the-shelf predictions.

CCS Concepts: • **Theory of computation** → **Caching and paging algorithms**; **Adversary models**;

Additional Key Words and Phrases: Online algorithms, machine learned predictions, paging, beyond worst-case analysis

ACM Reference format:
Thodoris Lykouris and Sergei Vassilvitskii. 2021. Competitive Caching with Machine Learned Advice. *J. ACM* 68, 4, Article 24 (July 2021), 25 pages.
https://doi.org/10.1145/3447579

1 INTRODUCTION

Despite the success and prevalence of machine learned systems across many application domains, there are still a lot of hurdles that one needs to overcome to deploy an ML system in practice [43]. As these systems are rarely perfect, a key challenge is dealing with errors that inevitably arise.

An extended abstract of this article appeared in the *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)* [31].
This article was initiated during the first author's internship at Google NYC while he was a Ph.D. student at Cornell University; the work was partly supported by NSF under award CCF-1563714 and by a Google Ph.D. Fellowship.
Authors' addresses: T. Lykouris, Microsoft Research New York City; email: thlykour@microsoft.com; S. Vassilvitskii, Google Research New York City; email: sergeiv@google.com.

0004-5411/2021/07-ART24
https://doi.org/10.1145/3447579

There are many reasons why learned systems may exhibit errors when deployed. First, most of them are trained to be good *on average*, minimizing some expected loss. In doing so, the system may invest its efforts in reducing the error on the majority of inputs, at the expense of increased error on a handful of outliers. Another problem is that generalization error guarantees only apply when the train and test examples are drawn from the same distribution. If this assumption is violated, e.g., due to distribution drift or adversarial examples [46], the machine learned predictions may be very far from the truth. In all cases, any system backed by machine learning needs to be robust enough to handle occasional errors.

While machine learning is in the business of predicting the unknown, online algorithms provide guidance on how to act without *any* knowledge of future inputs. These powerful methods show how to hedge decisions so that, regardless of what the future holds, the online algorithm performs nearly as well as the optimal offline algorithm. However these guarantees come at a cost: since they protect against the worst case, online algorithms may be overly cautious, which translates to high competitive ratios even for seemingly simple problems.

In this work, we ask:

What if the online algorithm is equipped with a machine learned predictor? How can one use this predictor to combine the predictive power of machine learning with the robustness of online algorithms?

We focus on a prototypical example of this area: the online paging, or *caching* problem. In this setting, a series of requests arrives one at a time to a server equipped with a small amount of memory. Upon processing a request, the server places the answer in the memory (in case an identical request comes in the future). Since the local memory has limited size, the server must decide which of the current elements to evict. It is well known that if the local memory or *cache* has size k, then any deterministic algorithm incurs competitive ratio $\Omega(k)$. However, an $O(k)$ bound can be also achieved by almost any reasonable strategy, thus this metric fails to distinguish between algorithms that perform well in practice and those that perform poorly. The competitive ratio of the best randomized algorithm is $\Theta(\log k)$ which, despite its elegant analysis , is much higher than what is observed on real inputs.

In contrast, we show how to use machine learned predictions to achieve a competitive ratio of $2 + O(\min(\sqrt{\eta/\textsc{Opt}}, \log k)$, when using a predictor—with total prediction error of η, where $\textsc{Opt}$ is the value of the offline optimal solution see Section 3.2 for a precise statement of results . Thus, when the predictions are accurate (small η), our approach circumvents the worst-case lower bounds. On the other hand, even when the oracle is inaccurate (large η), the performance degrades gracefully to almost match the worst-case bound.

1.1 Our Contribution

The conceptual contribution of this article lies in formalizing the interplay between machine learning and competitive analysis by introducing a general framework (Section 2), and a set of desiderata for online algorithms that use machine learned predictions.

We look for approaches that:

- Make *minimal* assumptions on the machine learned predictor: Specifically, since most machine learning guarantees are on the expected performance, our results are parametric as a function of the error of the predictor, η, and not the distribution of the error.
- Are *robust*: A better oracle (one with lower η) results in a smaller competitive ratio
- Are worst-case *competitive*: No matter the performance of the oracle on the particular instance, the algorithm behaves comparably to the best online algorithm for the problem.

We instantiate the general framework to the online caching problem, specifying the form of the predictions and presenting an algorithm that uses these predictions effectively (Section 3.2). Along the way, we show that algorithmic innovation is necessary: Simply following the recommendations of the predictor may lead to poor performance, even when the average error is small (Section 3.1). Instead, we adapt the Marker algorithm [22] to carefully incorporate the feedback of the predictor. The resulting approach, which we call the *Predictive Marker,* has guarantees that capture the best of both worlds: The algorithm performs better as the error of the predictor decreases, but performs nearly as well as the best online algorithm in the worst case. Our analysis generalizes to multiple loss functions (such as absolute loss and squared loss). This freedom in the loss function with the black-box access to the oracle allows our results to be strengthened with future progress in machine learning and reduces the task of designing better algorithms to the task of finding better predictors.

Our approach enjoys many practically desired qualities as we discuss in Section 4 among other extensions. Most notably among those qualities, our framework enables us to robustify commonly used practical heuristics such as the **Least Recently Used (LRU)** algorithm. Despite their practical performance, these heuristics have poor provable guarantees as they are prone to particular worst-case instances. Expressing the guidance of LRU as a predictor in Predictive Marker, we retain its practical performance while also arming it with strong worst-case guarantees (Section 4.2).

We complement our theoretical findings with empirical results (Section 5). We test the performance of our algorithm on public data using off-the-shelf machine learning models. We compare the performance to the LRU algorithm, which serves as the gold standard, the original Marker algorithm, as well as directly using the predictor. In all cases, the Predictive Marker algorithm outperforms known approaches.

Before moving to the technical content, we provide a simple example that highlights the main concepts of this work.

1.2 Example: Faster Binary Search

Consider the classical binary search problem. Given a sorted array A on n elements and a query element q, the goal is to either find the index of q in the array or state that it is not in the set. The textbook method is binary search: compare the value of q to that of the middle element of A and recurse on the correct half of the array. After $O(\log n)$ probes, the method either finds q or returns.

Instead of applying binary search, one can train a classifier, h, to predict the position of q in the array. (Although this may appear to be overly complex, Kraska et al. [28] empirically demonstrate the advantages of such a method.) How to use such a classifier? A simple approach is to first probe the location at $h(q)$; if q is not found there, we immediately know whether it is smaller or larger. Suppose q is larger than the element in $A[h(q)]$ and the array is sorted in increasing order. We probe elements at $h(q)+2, h(q)+4, h(q)+8$, and so on, until we find an element larger than q (or we hit the end of the array). Then we simply apply binary search on the interval that's guaranteed to contain q.

What is the cost of such an approach? Let $t(q)$ be the true position of q in the array (or the position of the largest element smaller than q if it is not in the set). The absolute loss of the classifier on q is then $\epsilon_q = |h(q) - t(q)|$. On the other hand, the cost of running the above algorithm starting at $h(q)$ is at most $2(\log|h(q) - t(q)|) = 2\log\epsilon_q$.

If the queries q come from a distribution, then the expected cost of the algorithm is:

$$2\mathbb{E}_q\Big[\log\left(|h(q) - t(q)|\right)\Big] \leq 2\log\mathbb{E}_q\Big[|h(q) - t(q)|\Big] = 2\log\mathbb{E}_q[\epsilon_q],$$

where the inequality follows by Jensen's inequality. This gives a tradeoff between the performance of the algorithm and the absolute loss of the predictor. Moreover, since ϵ_q is trivially bounded by n, this shows that even relatively weak classifiers (those with average error of $\sqrt{n}$) can lead to an improvement in asymptotic performance.

1.3 Related Work

Our work builds upon the foundational work on competitive analysis and online algorithms; for a great introduction, see the book by Borodin and El-Yaniv [14]. Specifically, we look at the standard caching problem, see, for example, [39]. While many variants of caching have been studied over the years, our main starting point will be the Marker algorithm by Fiat et al. [22].

As we mentioned earlier, competitive analysis fails to distinguish between algorithms that perform well in practice, and those that perform well only in theory. Several fixes have been proposed to address these concerns, ranging from resource augmentation, where the online algorithm has a larger cache than the offline optimum [44], to models of real-world inputs that restrict the inputs analyzed by the algorithm, for example, insisting on locality of reference [5], or the more general Working Set model [19].

The idea of making assumptions on the nature of the input to prove better bounds is common in the literature. The most popular of these is that the data arrive in a random order. This is a critical assumption in the secretary problem, and, more generally, in other streaming algorithms; see, for instance, the survey by McGregor [33]. While the assumption leads to algorithms with useful insight into the structure of the problem, it rarely holds true and is often hard to verify.

Another common assumption on the structure of the input gives rise to *Smoothed Analysis*, introduced in a pioneering work by Spielman and Teng [45], explaining the practical efficiency of the Simplex method. This approach assumes that any worst-case instance is perturbed slightly before being passed to the algorithm; the idea is that this perturbation may be due to measurement error or some other noise inherent in the data. The goal then is to show that the worst-case inputs are brittle and do not survive the addition of random noise. Since its introduction, this method has been used to explain the unusual effectiveness of many practical algorithms such as ICP [11], Lloyd's method [10], and local search [20], in the face of exponential worst-case bounds.

The prior work that is closest in spirit to ours looks for algorithms that optimistically assume that the input has a certain structure, but also have worst-case guarantees when that fails to be the case. One such assumption is that the data are coming from a stochastic distribution and was studied in the context of online matching [35] and bandit learning [16]; both of these works provide improved guarantees if the input is stochastic but retain the worst-case guarantees otherwise. Subsequent work has provided a graceful decay in performance when the input is mostly stochastic (analogous to our robustness property) both in the context online matching [21] and bandit learning [30]. In a related note, Ailon et al. [4] consider "self-improving" algorithms that effectively learn the input distribution, and adapt to be nearly optimal in that domain. Contrasting to these works, our approach utilizes a different structure in the data: the fact that the sequence can be predicted.

Our work is not the first to use predictions to enhance guarantees in online decision-making. The ability to predict something about the input has also used been used in online learning by Rakhlin and Sridharan [41] where losses of next round are predicted and the guarantees scale with how erroneous these precitions are. Our focus is on competitive analysis approaches where requests affect the state of the system; as a result, a single misprediction can have long-lasting effect on the system. With respect to using predictions in competitive analysis, another approach was suggested by Mahdian et al. [32], who assume the existence of an optimistic, highly competitive, algorithm, and then provide a meta algorithm with a competitive ratio that interpolates between that of the worst-case algorithm and that of the optimistic one. This work is most similar to our approach, but it ignores two key challenges that we face: (i) identifying predictions that can lead to (near) offline optimality, and (ii) developing algorithms that use these predictions effectively in a robust way. The work of Mahdian et al. [32] starts directly from the point where such an "optimistic" algorithm is available, and combines it with a "good in the worst-case" algorithm in a black-box manner. This has similarities to the approaches we discuss in Section 4.3 and Remark 4.4, but does not answer how to develop the optimistic algorithm. As we show in this article, developing such algorithms may be non-trivial even when the predictions are relatively good.

In other words, we do not assume anything about the data, or the availability of good algorithms that work in restricted settings. Rather, we use the predictor to implicitly classify instances into "easy" and "hard" depending on their predictability. The "easy" instances are those on which the latest machine learning technology, be it perceptrons, decision trees, SVMs, Deep Neural Networks, GANs, LSTMs, or whatever else may come in the future, has few errors. In these instances, our goal is to take advantage of the predictions, and obtain low competitive ratios. (Importantly, our approach is completely agnostic to the inner workings of the predictor and treats it as a black box.) The "hard" instances, are those where the prediction quality is poor, and we have to rely more on classical competitive analysis to obtain good results.

A previous line of work has also considered the benefit of enhancing online algorithms with oracle advice (see [15] for a recent survey). This setting assumes access to an infallible oracle and studies the amount of information that is needed to achieve desired competitive ratio guarantees. Our work differs in two major regards. First, we do not assume that the oracle is perfect, as that is rarely the case in machine learning scenarios. Second, we study the tradeoff between oracle error and the competitive ratio, rather than focusing on the number of perfect predictions necessary.

Another avenue of research close to our setting asks what happens if the algorithm cannot view the whole input, but must rely on a sample of the input to make its choices. Introduced in the seminal work of Cole and Roughgarden [18], this notion of *Learning from Samples*, can be viewed as first designing a good prediction function, h, and then using it in the algorithms. Indeed, some of the follow-up work [12, 38] proves tight bounds on precisely how many samples are necessary to achieve good approximation guarantees. In contrast, we assume that the online algorithm is given access to a machine learned predictor, but does not know any details of its inner workings—we know neither the average performance of the predictor, nor the distribution of the errors.

Very recently, two articles explored domains similar to ours. Medina and Vassilvitskii [34] showed how to use a machine learned oracle to optimize revenue in repeated posted price auctions. Their work has a mix of offline calculations and online predictions and focuses on the specific problem of revenue optimization. Kraska et al. [28] demonstrated empirically that introducing machine learned components to classical algorithms (in their case, index lookups) can result in significant speed and storage gains. Unlike this work, their results are experimental, and they do not provide tradeoffs on the performance of their approach vis-à-vis the error of the machine learned predictor.

Finally, since the publication of the original article, learning-augmented algorithms has emerged as a rich area. Subsequently to our work, researchers have studied how to incorporate machine learned predictions in other settings such as ski rental [24, 40], scheduling [8, 29, 40], bin packing [8], bloom filters [36, 47], queueing [37], streaming algorithms [25], weighted paging [27], and page migration [26]. While many of these focus on improving competitive ratios, some of them explore other performance metrics, such as space complexity [25, 36, 47]. With respect to the unweighted caching problem, we consider, subsequent work has also provided refined guarantees under our prediction model [42, 48] or alternate prediction models [9].

2 ONLINE ALGORITHMS WITH MACHINE LEARNED ADVICE

In this section, we introduce a general framework for combining online algorithms with machine learning predictions, which we term **Online with Machine Learned Adviceframework*(OMLA)***. Before introducing the framework, we review some basic notions from machine learning and online algorithms.

2.1 Preliminaries

Machine Learning Basics. We are given a feature space $\mathcal{X}$, describing the salient characteristics of each item and a set of labels $\mathcal{Y}$. An example is a pair (x, y), where $x \in \mathcal{X}$ describes the specific features of the example, and $y \in \mathcal{Y}$ gives the corresponding label. In the binary search example, x can be thought as the query element q searched and y as its true position $t(x)$.

A hypothesis is a mapping $h : \mathcal{X} \to \mathcal{Y}$ and can be probabilistic in which case the output on $x \in \mathcal{X}$ is some probabilistically chosen $y \in \mathcal{Y}$. In binary search, $h(x)$ corresponds to the predicted position of the query.

To measure the performance of a hypothesis, we first define a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^{\geq 0}$. When the labels lie in a metric space, we define absolute loss $\ell_1(y, \hat{y}) = |y - \hat{y}|$, squared loss $\ell_2(y, \hat{y}) = (y - \hat{y})^2$, and, more generally, classification loss $\ell_c(y, \hat{y}) = \mathbf{1}_{y \neq \hat{y}}$.

Competitive analysis. To obtain worst-case guarantees for an online algorithm (that must make decisions as each element arrives), we compare its performance to that of an offline optimum (that has the benefit of hindsight). Let σ be the input sequence of elements for a particular online decision-making problem, $cost_A(\sigma)$ be the cost incurred by an online algorithm $\mathcal{A}$ on this input, and $\textsc{Opt}(\sigma)$ be the cost incurred by the optimal offline algorithm. Then algorithm $\mathcal{A}$ has competitive ratio $\textsc{cr}$ if for all sequences σ,

$$cost_{\mathcal{A}}(\sigma) \leq \textsc{cr} \cdot \textsc{Opt}(\sigma).$$

If the algorithm $\mathcal{A}$ is randomized, then $cost_{\mathcal{A}}(\sigma)$ corresponds to the expected cost of the algorithm in input σ where the expectation is taken over the randomness of the algorithm.

The Caching Problem. The caching (or online paging) problem considers a system with two levels of memory: a slow memory of size m and a fast memory of size k. A caching algorithm is faced with a sequence of requests for elements. If the requested element is in the fast memory, a *cache hit* occurs and the algorithm can satisfy the request at no cost. If the requested element is not in the fast memory, a *cache miss* occurs, the algorithm fetches the element from the slow memory, and places it in the fast memory before satisfying the request. If the fast memory is full, then one of the elements must be evicted. The eviction strategy forms the core of the problem. The goal is to find an eviction policy that results in the fewest number of cache misses.

It is well known that the optimal offline algorithm at time t evicts the element from the cache that will arrive the furthest in the future; this is typically referred to in the literature as Bélády's optimal replacement paging algorithm [13]. On the other hand, without the benefit of foresight, any deterministic caching algorithm achieves a competitive ratio of $\Omega(k)$, and any randomized caching algorithm achieves a competitive ratio of $\Omega(\log k)$ [39].

2.2 OMLA Definition

To define our framework in generality, we consider a general problem setting associated with a general prediction model and then explain how both can be instantiated in the context of caching.

In traditional online algorithms, there is an universe $\mathcal{Z}$ and elements $z_i \in \mathcal{Z}$ arrive one at a time for rounds $i = 1, 2, \ldots$. The problem Π specifies the optimization problem at hand, along with the required constraints and any necessary parameters. For example, in the problem of caching studied in this article, $\Pi_{\text{CACHING}} = \textsc{Caching}(n,k)$, is parametrized by the number of requests n and the cache size k.

Augmenting Online Algorithms with Machine Learned Predictors. In our framework, we assume that the requested elements are augmented with a feature space $\mathcal{X}$ (discussed below). We refer to the resulting feature-augmented elements as *items* and denote the item of the ith request by σ_i. An input $\sigma \in \Pi$ corresponds to a sequence of items: $\sigma = (\sigma_1, \sigma_2, \ldots)$. For the problem of caching $\Pi_{\text{CACHING}} = \textsc{Caching}(n,k)$, the item sequence σ has length n.

Each item is associated with a particular element by $z(\sigma_i) \in \mathcal{Z}$ as well as a feature $x(\sigma_i) \in \mathcal{X}$. The features capture any information that may be available to the machine learning algorithm to help provide meaningful predictions. In caching, these may include information about the sequence prior to the request, time patterns associated to the particular request, or any other information. We note that even for caching, items are more general than their associated element: two items with the same element are *not* necessarily the same, as their corresponding features may differ.

Prediction Model. The prediction model $\mathcal{H}$ prescribes a label space $\mathcal{Y}$; the ith item has label $y(\sigma_i) \in \mathcal{Y}$. This label space can be viewed as the information needed to solve the task (approximately) optimally. As we discuss in the end of Section 2.2, deciding on a particular label space is far from trivial and it often involves tradeoffs between learnability and accuracy.

Given a prediction model $\mathcal{H}$ determining a label space $\mathcal{Y}$, a machine learned predictor $h \in \mathcal{H}$ maps features $x \in \mathcal{X}$ to predicted labels $h(x) \in \mathcal{Y}$. In particular, for item σ_i, the predictor h returns a predicted label $h(x(\sigma_i))$. To ease notation, we denote this by $h(\sigma_i)$. Here we assume that this mapping from features to labels is deterministic; our results extend to randomized mappings by applications of Jensen's inequality (see Section 3.5).

Loss Functions and Error of Predictors. To evaluate the performance of a predictor on a particular input, we consider a loss function ℓ. Similar to the prediction model, selecting a loss function involves tradeoffs between the learnability of the predictor and the resulting performance guarantees; we elaborate on these tradeoffs in the end of Section 2.2. For a given loss function ℓ, problem Π, and prediction model $\mathcal{H}$, the performance of the predictor $h \in \mathcal{H}$ on input $\sigma \in \Pi$ is evaluated by its error $\eta_\ell(h,\sigma)$. In full generality, this error can depend on the whole input in complicated ways.

For the caching problem, the prediction model we consider predicts the subsequent time that a requested element will get requested again. In this case, a natural loss function such as absolute or squared loss decomposes the error across items. In later sections, we focus on such loss functions throughout this article and therefore can express the error as:

$$\eta_\ell(h,\sigma) = \sum_i \ell(y(\sigma_i), h(\sigma_i)).$$

Instantiated with the absolute loss function, the error of the predictor is $\eta_{\ell_1}(h,\sigma) = \sum_i |y(\sigma_i) - h(\sigma_i)|$. We will use $\eta_1(h,\sigma)$ as a shorthand for this absolute loss.

We note that this decomposition across items may not be possible. For example, edit distance does not decompose across items but needs to be evaluated with respect to the whole instance. The general framework we define extends to such non-decomposable loss functions but the above restriction lets us better describe our results and draws more direct connection with classical machine learning notions.

We now summarize the general concepts of our framework in the following definition:

Definition 1. The Online with Machine Learned Advice (OMLA) framework is defined with respect to (a) a problem Π, (b) a prediction model $\mathcal{H}$ determining a feature space $\mathcal{X}$ and a label space $\mathcal{Y}$, and (c) a loss function ℓ. An instance consists of:

- An input $\sigma \in \Pi$ consisting of items σ_i arriving online, each with features $x(\sigma_i) \in \mathcal{X}$ and label $y(\sigma_i) \in \mathcal{Y}$;
- A predictor $h : \mathcal{X} \rightarrow \mathcal{Y}$ that predicts a label $h(\sigma_i)$ for each $x(\sigma_i) \in \mathcal{X}$;
- The error of predictor h at sequence σ w.r.t. loss ℓ, $\eta_\ell(h, \sigma)$.

Our goal is to create online algorithms that, when augmented with a predictor h, can use its advice to achieve an improved competitive ratio. To evaluate how well an algorithm $\mathcal{A}$ performs with respect to this task, we extend the definition of competitive ratio to be a function of the predictor's error. We first define the set of predictors that are sufficiently accurate.

Definition 2. For a fixed optimization problem Π, let $\textsc{Opt}_\Pi(\sigma)$ denote the value of the optimal solution on the input σ. Consider a prediction model $\mathcal{H}$. A predictor $h \in \mathcal{H}$ is *ϵ-accurate* with respect to a loss function ℓ for Π if for any $\sigma \in \Pi$:

$$\eta_{\ell,\mathcal{H},\Pi}(h, \sigma) \leq \epsilon \cdot \textsc{Opt}_\Pi(\sigma).$$

We will use $\mathcal{H}_{\ell,\mathcal{H},\Pi}(\epsilon)$ to denote the class of ϵ-accurate predictors.

At first glance, it may appear unnatural to tie the error of the prediction to the value of the optimal solution. However, our goal is to have a definition that is invariant to simple padding arguments. For instance, consider a sequence $\sigma' = \sigma\sigma$, which concatenates two copies of an input σ.[1] It is clear that the prediction error of any predictor doubles, but this is not due to the predictor suddenly being worse. One could instead normalize the prediction error by the length of the sequence, but in many problems, including caching, one can artificially increase the length of the sequence without impacting the value of the optimum solution, or the impact of predictions. Normalizing by the value of the optimum addresses both of these problems.

Call an algorithm $\mathcal{A}$ *ϵ-assisted* if it has access to an ϵ-accurate predictor. The competitive ratio of an ϵ-assisted algorithm is itself a function of ϵ and may also depend on parameters specified by Π such as the cache size k or the number of elements n.

Definition 3. For a fixed optimization problem Π and a prediction model $\mathcal{H}$, let $\textsc{input}\mathrm{CR}_{\mathcal{A},\mathcal{H},\Pi}(h, \sigma)$ be the competitive ratio of algorithm $\mathcal{A}$ that uses a predictor $h \in \mathcal{H}$ when applied on an input $\sigma \in \Pi$. The *competitive ratio* of an ϵ-assisted algorithm $\mathcal{A}$ for problem Π with respect to loss function ℓ and prediction model $\mathcal{H}$ is:

$$\textsc{cr}_{\mathcal{A},\ell,\mathcal{H},\Pi}(\epsilon) = \max_{\sigma \in \Pi,\, h \in \mathcal{H}_{\ell,\mathcal{H},\Pi}(\epsilon)} \textsc{input}CR_{\mathcal{A},\mathcal{H},\Pi}(h, \sigma)$$

We now define the desiderata that we wish our algorithm to satisfy. We would like our algorithm to perform as well as the offline optimum when the predictor is perfect, degrade gracefully with the error of the predictor, and perform as well as the best online algorithm regardless of the error of the predictor. We define these properties formally for the performance of an algorithm $\mathcal{A}$ a particular loss function ℓ, prediction model $\mathcal{H}$, and problem Π.

Definition 4. $\mathcal{A}$ is *β-consistent* if $\textsc{cr}_{\mathcal{A},\ell,\mathcal{H},\Pi}(0) = \beta$.

Definition 5. $\mathcal{A}$ is *α-robust* for a function $\alpha(\cdot)$, if $\textsc{cr}_{\mathcal{A},\ell,\mathcal{H},\Pi}(\epsilon) = O(\alpha(\epsilon))$.

Definition 6. $\mathcal{A}$ is *γ-competitive* if $\textsc{cr}_{\mathcal{A},\ell,\mathcal{H},\Pi}(\epsilon) \leq \gamma$ for all values of ϵ.

Our goal is to find algorithms that simultaneously optimize the aforementioned three properties. First, they are ideally 1-consistent: recovering the optimal solution when the predictor is perfect.

[1]In order for both instances to be equally sized and therefore be inputs of the same problem Π, we can think of padding the end of the first instance with the same dummy request.

This is not necessarily feasible for multiple reasons. From a computational side, the underlying problem may be NP-hard. Moreover, achieving any notion of robustness may inevitably be at odds with exact consistency. As a result, we are satisfied with β-consistency for some small constant β. Second, they are $\alpha(\cdot)$-robust for a slow growing function α: seamlessly handling errors in the predictor. This function depends on the exact prediction model and the way that the loss is defined with respect to it. As discussed below, different prediction models and loss functions may well lead to different robustness guarantees while also achieve different levels of learnability. Finally, they are worst-case competitive: they perform as well as the best online algorithms even when the predictor's error is high. As hinted before, all competitive ratios can be functions of the problem dimensions inherent in Π; for example, in caching, the worst-case performance γ needs to depend on the cache size k. Ideally, the consistency and robustness quantities β and $\alpha(\epsilon)$ (for small $\epsilon > 0$) do not display such dependence on these problem dimensions.

Discussion on the OMLA Framework. For the caching problems predictions and loss functions as decomposable per element, but one can also define predictions with respect to different parts of the instance. For example, subsequent works used strong lookahead for weighted paging [27] and learned weights for scheduling [29]—both of these prediction models are not per-element. Similarly, loss functions can be computed with respect to the complete instance. Per-item predictions, however, have a stronger connection to classical machine learning terminology.

Next, thus far, we have disregarded the question of where the predictor comes from and how learnable it is. This is an important question and has been elegantly discussed in multiple contexts such as revenue maximization [18]. In general, the decision on both the prediction model $\mathcal{H}$ and the loss function ℓ needs to take into account the learnability question and have a better understanding of the exact tradeoffs is a major open direction of our work. Subsequent work sheds further light on the learnability question in the context of our framework [6].

Finally, although we define our framework with respect to competitive analysis, predictions can be useful to augment online algorithm design with respect to other metrics such as space complexity [25, 36, 47] and our framework can be easily extended to capture such performance gains.

2.3 Caching with ML Advice

In order to instantiate the framework to the caching problem, we need to specify the items of the input sequence σ, the prediction model $\mathcal{H}$ (and thereby the label space $\mathcal{Y}$), as well as the loss function ℓ. Each item corresponds to one request σ_i; the latter is associated with an element $z(\sigma_i) \in \mathcal{Z}$ and features $x(\sigma_i) \in \mathcal{X}$ that encapsulate any information available to the machine learning algorithm. The element space $\mathcal{Z}$ consists of the m elements of the caching problem defined in Section 2.1. The exact choice of the feature space $\mathcal{X}$ is orthogonal to our setting, though of course richer features typically lead to smaller errors. The input sequence $\sigma = (\sigma_1, \sigma_2, \ldots)$ of the requested items is assumed to be fixed in advance and is oblivious to the realized randomness of the algorithm but unknown to the algorithm.

The main design choice of the prediction model is the question of what to predict which is captured in our framework by the choice of the label space. For caching problems, a natural candidate is predicting the next time a particular element is going to appear. It is well known [13] that when such predictions are perfect, the online algorithm can recover the best offline optimum. Formally, the label space $\mathcal{Y}$ we consider is a set of positions in the sequence, $\mathcal{Y} = \mathbb{N}^+$. Given a sequence σ, the label of the ith element is $y(\sigma_i) = \min_{t>i}\{t : x(\sigma_t) = x(\sigma_i)\}$. If the element is never seen again, we set $y(\sigma_i) = n + 1$. Note that $y(\sigma_i)$ is completely determined by the sequence σ. We use $h(\sigma_i)$ to denote the outcome of the prediction on an element with features $x(\sigma_i)$. Note that the feature is not only a function of the element identity $z(\sigma_i)$; when an element reappears, its features may be drastically different.

In what follows, we fix the problem $\Pi = \textsc{Caching}(n, k)$ to a caching problem with n requests and cache size k and the prediction model $\mathcal{H}$ to be about the next appearance of a requested element. We consider a variety of loss functions (discussed in detail in Section 3.3) that capture, for example, absolute and squared loss functions. To ease notation, we therefore drop any notational dependence on the prediction model $\mathcal{H}$ and the problem Π as both are fixed throughout the rest of the article, but keep the dependence on the loss function ℓ.

3 MAIN RESULT: PREDICTIVE MARKER

In this section, we describe the main result: an algorithm that satisfies the three desiderata of the previous section. Before describing our algorithm, we show that combining the predictions with ideas from competitive analysis is to a large extent essential; blindly evicting the element that is predicted the furthest in the future by the predictor (or simple modifications of this idea) can result in poor performance both with respect to robustness and competitiveness.

3.1 Blindly Following the Predictor Is Not Sufficient

Evicting Element Predicted the Furthest in the Future. An immediate way to use the predictor is to treat its output as truth and optimize based on the whole predicted sequence. This corresponds to the Bélády rule that evicts the element predicted to appear the furthest in the future. We refer to this algorithm as algorithm $\mathcal{B}$ (as it follows the Bélády rule). Since this rule achieves offline optimality, this approach is consistent, i.e., if the predictor is perfect, this algorithm is ex-post optimal. Unfortunately, this approach does not have similarly nice performance with respect to the other two desiderata. With respect to robustness, the degradation with the average error of the predictor is far from the best possible, while a completely unreliable predictor leads to unbounded competitive ratios, far from the ones of the best online algorithm.

Proposition 3.1. *Consider the caching problem Π with n requests and cache size k, the prediction model $\mathcal{H}$ that predicts the next arrival of a requested element and the absolute loss function ℓ_1. The competitive ratio of ϵ-assisted algorithm $\mathcal{B}$ is $\textsc{cr}_{\mathcal{B},\ell_1}(\epsilon) = \Omega(\epsilon)$.*

The implication is that when the error of the predictor is much worse than the offline optimum, the competitive ratio becomes unbounded. With respect to robustness, the rate of decay is far from optimal, as we will see in Section 3.3.

Proof of Proposition 3.1. We will show that, for every ϵ, there exist a sequence σ and a predictor h such that the absolute error $\eta_1(h, \sigma) \leq \epsilon \cdot \textsc{Opt}$ while the competitive ratio of algorithm $\mathcal{B}$ is $\frac{\epsilon-1}{2}$. For ease of presentation, assume that $\epsilon > 3$. Consider a cache of size $k = 2$ and three elements a, b, c; the initial configuration of cache is a, c. The sequence consists of repetitions of the following sequence with ϵ requests per repetition. The actual sequence is $a\,\underbrace{bcbc\ldots bc}_{\epsilon-1}\,a\,\underbrace{bcbc\ldots bc}_{\epsilon-1}\ldots$ (a appears once and then bc appears $(\epsilon-1)/2$ times).

In any repetition, the predictor accurately predicts the arrival time of all elements apart from two: (i) when element a arrives, it predicts that it will arrive again two steps after the current time (instead of in the first step of the next repetition) and (ii) when b arrives for the last time in one repetition, it predicts it to arrive again in the fourth position of the next repetition (instead of the second). As a result, the absolute error of the predictor is ϵ ($\epsilon - 2$ error in the a-misprediction and 2 error in the b-misprediction). The optimal solution has two evictions per repetition (one to bring a in the cache and one to directly evict it afterwards). Instead, the algorithm never evicts a as it is predicted to arrive much earlier than all other elements, and therefore has $\epsilon - 1$ cache misses. This means that the competitive ratio of this algorithm is $\Omega(\eta_1(h,\sigma)/\text{Opt}(\sigma))$ which completes the proof. □

Evicting Elements with Proven Wrong Predictions. The problem in the above algorithm is that algorithm $\mathcal{B}$ keeps too much faith in predictions that have been already proven to be wrong (as the corresponding elements are predicted to arrive in the past). It is tempting to "fix" the issue by evicting any element whose predicted time has passed, and evict again the element predicted the furthest in the future if no such element exists. We call this algorithm $\mathcal{W}$ as it takes care of wrong predictions. Formally, let $h(j,t)$ denote the last prediction about z_j at or prior to time t. At time t algorithm $\mathcal{W}$ evicts an arbitrary element from the set $S_t = \{j : h(j,t) < t\}$ if $S_t \neq \emptyset$ and $\arg\max_{z_i \in \text{Cache(t)}} h(i,t)$ otherwise. We show that algorithm $\mathcal{W}$ has similarly bad performance guarantees.

Proposition 3.2. *Consider the caching problem* Π *with n requests and cache size k, the prediction model* $\mathcal{H}$ *that predicts the next arrival of a requested element and the absolute loss function* ℓ_1. *The competitive ratio of* ϵ*-assisted algorithm* $\mathcal{W}$ *is* $\text{cr}_{\mathcal{W},\ell_1}(\epsilon) = \Omega(\epsilon)$.

Proof. Consider a cache of size $k = 3$ and four elements a, b, c, d; the initial configuration of cache is a, b, c and then d arrives. The actual sequence consists of repetitions of the following sequence with $(\epsilon/2) + 1$ requests per repetition (for ease of presentation, assume that $\epsilon > 6$). The actual sequence σ is $d\,\underbrace{abcabc\ldots abc}_{\epsilon/2}\,d\,\underbrace{abcabc\ldots abc}_{\epsilon/2}\ldots$.

In any repetition, the predictor h accurately predicts the arrival time of element d but always makes mistake in elements a, b, c by predicting them to arrive two time steps earlier. As a result, the absolute error of the predictor is ϵ (error of 2 for any of the appearances of a, b, c). The optimal solution has two evictions per repetition (one to bring element d and one to evict it afterwards). Instead the algorithm always evicts elements a, b, c because they are predicted earlier than their actual arrival and are therefore evicted as "wrong" predictions. This means that the competitive ratio of this algorithm is also $\Omega(\eta_1(h,\sigma)/\text{Opt}(\sigma))$ which completes the proof. □

The latter issue can be again fixed by further modifications of the algorithm but these simple examples demonstrate that, unless taken into account, mispredictions can cause significant inefficiency in the performance of the algorithms.

Beyond Blindly Trusting the Predictor. The common problem in both examples is that there is an element that should be removed but the algorithm is tricked into keeping it in the cache. To deal with this in practice, most popular heuristics such as **LRU (Least Recently Used)** and **FIFO (First In First Out)** avoid evicting recent elements when some elements have been dormant for a long time. This tends to utilize nice locality properties leading to strong empirical performance (especially for LRU). However, such heuristics impose a strict eviction policy which leads to weak performance guarantees. Moreover, incorporating additional information provided by the predictor becomes complicated.

Competitive analysis has also built on the idea of evicting dormant elements via developing algorithms with stronger theoretical guarantees such as Marker. In the next section, we show how we can incorporate predictions in the Marker algorithm to enhance its performance when the predictions are good while retaining the worst-case guarantees. Interestingly, via our framework, we can provide improved guarantees for the aforementioned heuristics such as LRU, improving their worst-case guarantees while retaining their practical performance (see Section 4.2).

3.2 Predictive Marker Algorithm

We now present our main technical contribution, a prediction-based adaptation of the Marker algorithm [22]. This ϵ-assisted algorithm gets a competitive ratio of $2 \cdot \min(1 + \sqrt{5\epsilon}, 2H_k)$ where $H_k = 1 + 1/2 + \cdots + 1/k$ denotes the k-th Harmonic number.

Classic Marker algorithm. We begin by recalling the Marker algorithm and the analysis of its performance. The algorithm runs in phases. At the beginning of each phase, all elements are unmarked. When an element arrives and is already in the cache, the element is marked. If it is not in the cache, a *random unmarked* element is evicted, the newly arrived element is placed in the cache and is marked. Once all elements are marked and a new cache miss occurs, the phase ends and we unmark all of the elements.

For the purposes of analysis, an element is called *clean* in phase r if it appears during phase r, but does not appear during phase $r - 1$. In contrast, elements that also appeared in the previous phase are called *stale*. The marker algorithm has a competitive ratio of $2H_k - 1$ and the analysis is tight [3]. We use a slightly simpler analysis that achieves competitive ratio of $2H_k$ below.

The crux of the upper bound lies in two claims. The first relates the performance of the optimal offline algorithm to the total number of clean elements Q across all phases.

Claim 1 ([22]). *Let Q be the number of clean elements. Then the optimal algorithm suffers at least $Q/2$ cache misses.*

The second comes from bounding the performance of the algorithm as a function of the number of clean elements.

Claim 2 ([22]). *Let Q be the number of clean elements. Then the expected number of cache misses of the marker algorithm is $Q \cdot H_k$.*

Predictive Marker. The algorithm of [22] is part of a larger family of *marking* algorithms; informally, these algorithms never evict marked elements when there are unmarked elements present. Any algorithm in this family has a worst-case competitive ratio of k. Therefore, pairing predictions with a marking algorithm would avoid the pathological examples we saw previously.

A natural approach is to use predictions for tie-breaking, specifically evicting the element whose predicted next appearance time is furthest in the future. When the predictor is perfect (and has zero error), the stale elements never result in cache misses, and therefore, by Claim 1, the algorithm has a competitive ratio of 2. On the other hand, by using the Marker algorithm and not blindly trusting the oracle, we can guarantee a worst-case competitive ratio of k.

We extend this direction to further reduce the worst-case competitive ratio to $O(H_k)$. To achieve this, we combine the prediction-based tie-breaking rule with the random tie-breaking rule. Suppose an element e is evicted during the phase. We construct a blame graph to understand the reason why e is evicted; this may happen for two distinct reasons. First, e may have been evicted when a clean element c arrived; in this case, we create a new node c which can be thought as the start of a distinct chain of nodes. Alternatively, it may have been evicted because a stale element s arrived (s was previously evicted in the same phase); in this case, we add a directed edge from e to s. Note that the graph is always a set of chains (paths). The total length of the chains represents the total number of evictions incurred by the algorithm during the phase, whereas the number of distinct chains represents the number of clean elements. We call the lead element in every chain *representative* and denote it by $\omega(r, c)$, where r is the index of the phase and c the index of the chain in the phase.

Our modification is simple—when a stale element arrives, it evicts a new element in a prediction-based manner if the chain containing it has length less than H_k. Otherwise, it evicts a random unmarked element. Looking ahead to the analysis, this switch to uniform evictions results in at most

H_k additional elements added to any chain during the course of the phase. This guarantees that the competitive ratio is at most $O(H_k)$ in expectation; we make the argument formal in Theorem 3.3.

The key to the analysis is the fact that the chains are disjoint, thus the interactions between evictions can be decomposed cleanly. We give a formal description of the algorithm in Algorithm 1. For simplicity, we drop dependence on σ from the notation.

ALGORITHM 1: Predictive Marker

```
Require: Cache $C$ of size $k$ initially empty ($C \leftarrow \emptyset$).
 1: Initialize phase counter $r \leftarrow 1$, unmark all elements ($\mathcal{M} \leftarrow \emptyset$), and set round $i \leftarrow 1$.
 2: Initialize clean element counter $q_r \leftarrow 0$ and tracking set $\mathcal{S} \leftarrow \emptyset$.
 3: Element $z_i$ arrives, and the predictor gives a prediction $h_i$. Save prediction $p(z_i) \leftarrow h_i$.
 4: if $z_i$results in cache hit or the cache is not full ($z_i \in C$ or $|C| < k$) then
 5:     Add to cache $C \leftarrow C \cup \{z_i\}$ without evicting any element and go to Step 26
 6: end if
 7: if the cache is full and all cache elements are marked ($|\mathcal{M}| = k$) then
 8:     Increase phase ($r \leftarrow r + 1$), initialize clean counter ($q_r \leftarrow 0$), save current cache($C \rightarrow \mathcal{S}$) as
        the set of elements that are possibly stale in the new phase, and unmark elements ($\mathcal{M} \leftarrow \emptyset$).
 9: end if
10: if $z_i$ is a clean element ($z_i \notin \mathcal{S}$) then
11:     Increase number of clean elements: $q_r \leftarrow q_r + 1$.
12:     Initialize size of new clean chain: $n(r, q_r) \leftarrow 1$.
13:     Select to evict unmarked element with highest predicted time: $e = arg\,max_{z \in C - \mathcal{M}} p(z)$.
14: end if
15: if $z_i$ is a stale element ($z_i \in \mathcal{S}$) then
16:     It is the representative of some clean chain. Let $c$ be this clean chain: $z_i = \omega(r, c)$.
17:     Increase length of the clean chain $n(r, c) \leftarrow n(r, c) + 1$.
18:     if $n(r, c) \leq H_k$ then
19:         Select to evict unmarked element with highest predicted time: $e = arg\,max_{z \in C - \mathcal{M}} p(z)$.
20:     else
21:         Select to evict a random unmarked element $e \in C - \mathcal{M}$.
22:     end if
23:     Update cache by evicting $e$: $C \leftarrow C \cup \{z_i\} - \{e\}$.
24:     Set $e$ as representative for the chain: $\omega(r, c) \leftarrow e$.
25: end if
26: Mark incoming element ($\mathcal{M} \leftarrow \mathcal{M} \cup \{z_i\}$), increase round ($i \leftarrow i + 1$), and go to Step 3.
```

3.3 Analysis

In order to analyze the performance of the proposed algorithm, we begin with a technical definition that captures how slowly a loss function ℓ can grow.

Definition 7. Let $\mathcal{A}_T$ be the set of all the sequences $A_T = a_1, a_2, \ldots, a_T$, of increasing integers of length T, that is $a_1 < a_2 < \cdots < a_T$, and $\mathcal{B}_T$ be the set of all sequences $B_T = b_1, b_2, \ldots, b_T$ of non-increasing reals of length T, $b_1 \geq b_2 \geq \cdots \geq b_T$. For a fixed loss function ℓ, we define its *spread* $S_\ell : \mathbb{N}^+ \rightarrow \mathbb{R}^+$ as:

$$S_\ell(m) = \min\{T : \forall A_T \in \mathcal{A}_T, B_T \in \mathcal{B}_T : \ell(A_T, B_T) \geq m\}$$

The spread captures the length of a subsequence that can be predicted in completely reverse order as a function of the error of the predictor with respect to loss function ℓ. We note that the sequence B_T is assumed to be over reals instead of integers as it corresponds to the outcome of the machine learned predictor and we do not want to unnecessarily restrict the output of this predictor.

The following lemma instantiates the spread for loss metrics we consider and is proved in the Appendix A.

LEMMA 3.1. *For absolute loss, $\ell_1(A, B) = \sum_i |a_i - b_i|$, the spread of ℓ_1 is $S_{\ell_1}(m) \leq \sqrt{5m}$. For squared loss, $\ell_2(A, B) = \sum_i (a_i - b_i)^2$, the spread of ℓ_2 is $S_{\ell_2}(m) \leq \sqrt[3]{14m}$.*

We now prove the main theorem of the paper.

THEOREM 3.3. *Consider the caching problem Π with n requests and cache size k, the prediction model $\mathcal{H}$ that predicts the next arrival of a requested element and any loss function ℓ with spread bounded by S_ℓ for some function S_ℓ that is concave in its argument. Then, the competitive ratio of ϵ-assisted Predictive Marker PM is bounded by:*

$$cr_{PM,\ell}(\epsilon) \leq 2 \cdot \min\left(1 + 2S_\ell(\epsilon), 2H_k\right).$$

To prove this theorem, we first introduce an analogue of Claim 2, which decomposes the total cost into that incurred by each of the chains individually.

To aid in our analysis, we consider the following marking algorithm, which we call **SM (Special Marking)**. Initially, we simply evict an arbitrary unmarked element. At some point, the adversary designates an arbitrary element not in the cache as *special*. For the rest of the phase, upon a cache miss, if the arriving element is special, the algorithm evicts a *random* unmarked element and designates the evicted element as special. If the arriving element is not special, the algorithm proceeds as before, evicting an arbitrary unmarked element.

LEMMA 3.2. *Using algorithm SM, in expectation at most H_k special elements cause cache misses per phase.*

PROOF. Since we use a marking algorithm, the set of elements that are in the cache at the end of each phase is determined by the element sequence $(z_1, z_2, \ldots)$ and is independent of the particular eviction rule among unmarked elements. Fix a phase that begins at time τ. Let E be the set of k distinct elements that arrive in this phase. Note that the arrival of the $k + 1$st distinct elements starts a new phase.

Consider the time $\tau^\star$ that an element is designated special and assume that, at this time, there are $i^\star$ special elements. At this point, we define $A \subseteq E$ to be the subset of the initial elements that are unmarked and in the cache; we refer to this set as the *candidate special set* as they are the only ones that can subsequently get designated as special; the set's initial cardinality is $i^\star$. This set is shrinking over time as elements are getting marked or evicted from the cache. Order the elements by the time of their first requst in this phase.

We now bound the probability of the event $\mathcal{E}_i$ that an element becomes special when it is the ith last element in A (based on the ordering by first arrival). By principle of deferred decisions, we consider the first time that, upon request of a special element, it evicts one of the last i elements in the active set. If this never happens, then the event $\mathcal{E}_i$ never occurs. Otherwise, observe that we select the element to evict uniformly at random, and there exists at least one element in the cache that never appears before the end of the phase. Second, if at any point an element j among the $i - 1$ elements in the active set becomes special, the i-th element can no longer become special as, at the time that j is requested, i is already marked. The above imply that the probability of the event $\mathcal{E}_i$ is at most:

$$Pr[\mathcal{E}_i] \leq \frac{1}{i+1}. \tag{1}$$

Therefore, given Equation (1), we can bound the expected number of misses caused by special elements as:

$$1 + \sum_{i=1}^{k-1} \frac{1}{i+1} = H_k,$$

where the first term is due to the first special element and the second term is due to events $\mathcal{E}_1$ through $\mathcal{E}_{k-1}$. □

We now provide the lemma that lies in the heart of our robustness property.

Lemma 3.3. *For any loss metric ℓ, any phase r, the expected length of any chain is at most $1 + \mathcal{S}_\ell(\eta_{r,c})$ where $\eta_{r,c}$ is the cumulative error of the predictor on the elements in the chain and $\mathcal{S}_\ell$ is the spread of the loss metric.*

Proof. The clean element that initiates the clean chain evicts one of the unmarked elements upon arrival. Since it does so based on the Bélády rule, it evicts the element s_1 that is predicted to reappear the latest in the future. If the predictor is perfect, this element will never appear in this phase. If, on the other hand, s_1 comes back (is a stale element), let s_2 be the element it evicts, which is predicted to arrive the furthest among the current unmarked elements.

Suppose there are m such evictions: $s_1, s_2, \ldots, s_m$. The elements were predicted to arrive in reverse order of their evictions. This is because elements s_j for $j > i$ were unmarked and in the cache when element s_i got evicted; therefore, s_i was predicted to arrive later. However, the actual arrival order is the reverse. If $\eta_{r,c}$ is the total error of these elements, setting the actual arriving times as the sequence A_T and the predicted ones as the sequence B_T in the definition of spread (Definition 7), it means that $m \leq \mathcal{S}_\ell(\eta_{r,c})$. □

Combining the above two lemmas, we can obtain a bound on the expected length of any chain.

Lemma 3.4. *For any loss metric ℓ, any phase r, the expected length of any chain is at most $\min(1 + 2\mathcal{S}_\ell(\eta_{r,c}), 2\log k)$ where $\eta_{r,c}$ is the cumulative error of the predictor on the elements in the chain and $\mathcal{S}_\ell$ is the spread of the loss metric.*

Proof. The proof follows from combining the two above lemmas. By Lemma 3.2, if the chain switches to random evictions, it incurs another H_k cache misses in expectation after the switch point (and its length increases by the same amount), capping in expectation the total length by $2H_k \leq 2\log k$. If the chain does not switch to random evictions, it has Bélády evictions and, by Lemma 3.3, it incurs at most $\mathcal{S}_\ell(\eta_{r,c})$ misses from stale elements. To ensure that the $2\log k$ term dominates the bound when $S_\ell(\eta_{r,c}) \geq \log k$, we multiply $S_\ell(\eta_{r,c})$ by a factor of 2 in the first term. □

Proof of Theorem 3.3. Consider an input $\sigma \in \Pi$ determining the request sequence. Let Q be the number of clean elements (and therefore also chains). Any cache miss corresponds to a particular eviction in one clean chain; there are no cache misses not charged to a chain by construction. By Lemma 3.4, we can bound the evictions from the clean chain c of the rth phase in expectation by $\min(1 + 2 \cdot S_\ell(\eta_{r,c}), 2\log k)$. Since both S_ℓ and the minimum operator are concave functions, the way to maximize the length of chains is to apportion the total error, η, equally across all of the chains. Thus, for a given error η and number Q of clean chains, the competitive ratio is maximized when the error in each chain is $\eta_{r,c} = \eta/Q$ each. The total number of stale elements is therefore in expectation at most: $Q \cdot \min(2 \cdot S_\ell(\eta/Q), 2H_k)$. By Claim 1, it holds that $Q/2 \leq \text{Opt}(\sigma)$, implying the result since $\text{Opt}(\sigma) \leq Q$. □

We now specialize the results for the absolute and squared losses.

Corollary 1. *The competitive ratio of ϵ-assisted Predictive Marker with respect to the absolute loss metric ℓ_1 is bounded by* $\textsc{cr}_{\mathcal{PM},\ell_1}(\epsilon) \le \min(2 + 2 \cdot \sqrt{5\epsilon}, 4H_k)$.

Corollary 2. *The competitive ratio of ϵ-assisted Predictive Marker with respect to the absolute loss metric ℓ_2 is bounded by* $\textsc{cr}_{\mathcal{PM},\ell_1}(\epsilon) \le \min(2 + 2 \cdot \sqrt[3]{14\epsilon}, 4H_k)$.

3.4 Tightness of Analysis

Robustness Rate of Predictive Marker. We show that our analysis is tight: any marking algorithm that uses the predictor in a deterministic way cannot achieve an improved guarantee with respect to robustness.

Theorem 3.4. *Any deterministic ϵ-assisted marking algorithm $\mathcal{A}$, that only uses the predictor in tie-breaking among unmarked elements in a deterministic fashion, has a competitive ratio of* $\textsc{cr}_{\mathcal{A},\ell}(\epsilon) = \Omega(\min(\mathcal{S}_\ell(\epsilon), k))$.

Proof. Consider a cache of size k with $k+1$ elements and any ϵ such that $S_\ell(\epsilon) < k$. We will construct an instance that exhibits the above lower bound. Since $\mathcal{A}$ uses marking, we can decompose its analysis into phases. Let σ be the request sequence, and assume that we do not have any repetition of an element inside the phase; as a result, the ith element of phase r corresponds to element $\sigma_{(r-1)k+i}$.

Suppose the predictor is always accurate on elements 2 through $k - S_\ell(\epsilon) + 1$ in each phase.

For the last $S_\ell(\epsilon) - 1$ elements of phase r as well as the first element of the of the next phase, the elements are predicted to come again at the beginning of the subsequent phase, at time $t = rk + 1$. Since the algorithm is deterministic, we order the elements so that their evictions are in reverse order of their arriving time. By the definition of spread, the error of the predictor in these elements is exactly ϵ and the algorithm incurs a cache miss in each of them. On the other hand, the offline optimum has only 1 miss per phase, which concludes the proof. □

On the Rate of Robustness in Caching. Theorem 3.4 establishes that the analysis of Predictive Marker is tight with respect to the rate of robustness, and suggests that algorithms that use the predictor in a deterministic manner may suffer from similar rates. However, a natural question that comes up is whether a better rate can be achieved using the predictor in a randomized way. We conjecture that a rate of $\log(1 + \sqrt{\epsilon})$ with respect to the absolute loss is possible, similar to the exponential improvement randomized schemes obtain over the deterministic guarantees of k with respect to worst-case competitiveness. In subsequent work, Rohatgi [42] made significant progress towards identifying the correct rate by proving refined upper and lower bounds.

3.5 Randomized Predictors

We now remove the assumption that the predictor h is deterministic and extend the definition of ϵ-accurate predictors (Definition 2) to hold in expectation. The randomness may either come in how the inputs are generated or in the predictions of h.

Definition 8. For a fixed optimization problem Π, let $\textsc{Opt}_\Pi(\sigma)$ denote the value of the optimal solution on input σ. Assume that the predictor is probabilistic and therefore the error of the predictor at σ is a random variable $\eta_\ell(h, \sigma)$. Taking expectation over the randomness of the predictor, we say that a predictor h is ϵ-accurate in expectation for Π if:

$$\mathbb{E}[\eta_\ell(h, \sigma)] \le \epsilon \cdot \textsc{Opt}_\Pi(\sigma).$$

Similarly an algorithm is ϵ-assisted if it has access to an ϵ-accurate predictor in expectation.

Analogously to the previous part, we can show:

Theorem 3.5. *Consider any loss function ℓ with spread bounded by S_ℓ for some function S_ℓ that is concave in its argument. Then, the competitive ratio of ϵ-assisted in expectation Predictive Marker PM is bounded by:*

$$\textsc{cr}_{PM,\ell}(\epsilon) \le 2 \cdot \min\left(1 + 2S_\ell\left(\epsilon\right), 2H_k\right).$$

Proof. For ease of notation assume that the outcomes of the predictors are finite. For each of these potential realizations, we can bound the performance of the algorithm by Theorem 3.3. The proof then follows by applying an additional Jensen's inequality on all the possible realizations due to the concavity of the spread and the min operator. □

4 DISCUSSION AND EXTENSIONS

Thus far, we have shown how to use an ϵ-accurate predictor to get a caching algorithm with an $O(\sqrt{\epsilon})$ competitive ratio for the absolute loss metric. We now provide a deeper discussion of the main results. In Section 4.1, we give a finer tradeoff of competitiveness and robustness. We then discuss some traits that limit the impact of the errors of the predictors in Section 4.2. Subsequently, we show that common heuristic approaches, such as LRU, can be expressed as predictors in our framework. This allows us to combine their predictive power with robust guarantees when they fail. Finally, in Section 4.3, we provide a black-box way to combine robust and competitive approaches.

4.1 Robustness vs Competitiveness Tradeoffs

One of the free parameters in Algorithm 1 is the length of the chain when the algorithm switches from following the predictor to random evictions. If the switch occurs after chains grow to γH_k in length, this provides a trade-off between competitiveness and robustness.

Theorem 4.1. *Suppose that, for $\gamma > 0$, the algorithm uses γH_k as switching point (line 18 in Algorithm 1); denote this algorithm by $\mathcal{PM}(\gamma)$. Let a loss function ℓ with spread bounded by S_ℓ for some function S_ℓ that is concave in its argument. Then, the competitive ratio of ϵ-assisted $\mathcal{PM}(\gamma)$ is bounded by:*

$$\textsc{cr}_{\mathcal{PM}(\gamma),\ell}(\epsilon) \le 2 \cdot \min\left(1 + \frac{1+\gamma}{\gamma} S_\ell\left(\epsilon\right), \gamma H_k, k\right).$$

Proof. The proof follows the proof of Theorem 3.3 but slightly changes the Lemma 3.2 to account for the new switching point. In particular, with respect to the second term, the expected length of each clean chain is at most H_k after the switching point, and, at most γH_k before the switching point by construction.

With respect to the robustness term, the length of each clean chain before the switch is bounded by the spread of the metric on this subsequence. Since the total length is in expectation at most $(1+\gamma)/\gamma$ higher, we need to adjust the first term accordingly.

Finally, the length of its clean chain is at most k regardless of the tie-breaking since we are using marking which provides the last term. □

Let us reflect on the above guarantee. When $\gamma \to 0$, then the algorithm is more conservative (switching to random evictions earlier); this reduces the worst-case competitive ratio but at the cost of abandoning the predictor unless it is extremely accurate. On the other hand, setting γ very high makes the algorithm trust the predictor more, reducing the competitive ratio when the predictor is accurate at the expense of a worst guarantee when the predictor is unreliable.

4.2 Practical Traits of Predictive Marker

Locality. The guarantee in Theorem 3.3 bounds the competitive ratio as a function of the quality of the prediction. One potential concern is that if the predictions have of a small number of very large errors, then the applicability of Predictive Marker may be quite limited.

Here we show that this is not the case. Due to the phase-based nature of the analysis, the algorithm essentially "resets" at the end of every phase, and therefore the errors incurred in one phase do not carry over to the next. Moreover, the competitive ratio in every phase is bounded by $O(H_k)$.

Formally, for any sequence σ, we can define phases that consist of exactly k distinct elements. Let $\textsc{cl}(r,\sigma)$ be the number of clean elements in phase r of sequence σ, and let $\eta_{\ell,r}(h,\sigma)$ denote the error of predictor h restricted only to elements occurring in phase r.

Theorem 4.2. *Consider a loss function ℓ with spread S_ℓ. If S_ℓ is concave, the competitive ratio of Predictive Marker PM at sequence σ when assisted by a predictor h is at most:*

$$\mathit{CR}_{\mathcal{PM},\ell} \leq \frac{\sum_r \mathit{CL}(r,\sigma)\cdot \min\left(1+2S_\ell(\eta_{\ell,r}(h,\sigma)), 2H_k\right)}{\sum_r \mathit{CL}(r,\sigma)}$$

Proof. The proof follows directly from Lemma 3.4 and applying Jensen's inequality only within the chains of the phase (instead of also across phases as we did in Theorem 3.3). □

This theorem illustrates a nice property of our algorithm. If the predictor h is really bad for a period of time (i.e., its errors are localized), then the clean chains of the corresponding phases will contribute to the second term (the logarithmic worst-case guarantee) but the other phases will provide enhanced performance utilizing the predictor's advice. In this way, the algorithm adapts to the quality of the predictions, and bad errors do not propagate beyond the end of a phase. This quality is very useful in caching where most patterns are generally well predicted but there may be some unforeseen sequences.

Robustifying LRU. Another practical property of our algorithm is that it can seamlessly incorporate heuristics that are known to perform well in practice. In particular, the popular Least Recently Used (LRU) algorithm can be expressed within the Predictive Marker framework. Consider the following predictor, h: when an element σ_i arrives at time i, the LRU predictor predicts next arrival time $h(\sigma_i) = -i$.

Note that, by doing so, at any point of time, among the elements that are in the cache, the element that is predicted the furthest in the future is exactly the one that has appeared the least recently. Also note that any marked element needs to have arrived later than any unmarked element. As a result, if we never switched to random evictions (or had k in the RHS of line 18 in Algorithm 1), the Predictive Marker algorithm assisted with the LRU predictor is exactly the LRU algorithm.

The nice thing that comes from this observation is that we can robustify the analysis of LRU. LRU, and its variants like LRU(2), tend to have very good empirical performance as using the recency of requests is a good predictor about how future requests will arise. However, the worst-case guarantee of LRU is unfortunately $\Theta(k)$ since it is a deterministic algorithm. By expressing LRU as a predictor in the Predictive Marker framework and using a switching point of H_k for each clean chain, we exploit most of this predictive power while also guaranteeing a logarithmic worst-case bound on it.

4.3 Combining Robustness and Competitiveness in a Black-Box Manner

In the previous section, we showed how we can slightly modify a classical competitive algorithm to ensure that it satisfies nice consistency and robustness properties when given access to a good predictor, while retaining the worst-case competitiveness guarantees otherwise. In this part, we show that, in fact, achieving the requirements individually is enough. In particular, we show a black-box way to combine an algorithm that is robust and one that is worst-case competitive. This reduction leads to a slightly worse bound, but shows that proving the robustness property (i.e., a graceful degradation with the error of the predictor) is theoretically sufficient to augment an existing worst-case competitive algorithm.

Theorem 4.3. *For the caching problem, let A be an α-robust algorithm and B a γ-competitive algorithm. We can then create a black-box algorithm ALG that is both 9α-robust and 9γ-competitive.*

Proof. We proceed by simulating A and B in parallel on the dataset, and maintaining the cache state and the number of misses incurred by each. Our algorithm switches between following the strategy of A and the strategy of B. Let $c_t(A)$ and $c_t(B)$ denote the cost (number of misses) of A and B up to time t. Without loss of generality, let *ALG* begin by following strategy of A; it will do so until a time t where $c_t(A) = 2 \cdot c_t(B)$. At this point *ALG* switches to following the eviction strategy of B, doing so until the simulated cost of B is double that of A: a time t' with $c_{t'}(B) = 2 \cdot c_{t'}(A)$. At this point, it switches back to following eviction strategy of A, and so on. When *ALG* switches from A to B, the elements that A has in cache may not be the same as those that B has in the cache. In this case, it needs to reconcile the two. However, this can be done lazily (at the cost of an extra cache miss for every element that needs to be reconciled). To prove the bound on the performance of the algorithm, we next show that $c_t(ALG) \leq 9 \cdot \min(c_t(A), c_t(B))$ for all t. We decompose the cost incurred by *ALG* into that due to following the different algorithms, which we denote by $f_t(ALG)$, and that due to reconciling caches, $r_t(ALG)$.

We prove a bound on the following cost f_t by induction on the number of switches. Without loss of generality, suppose that at time t, *ALG* switched from A to B, and at time t' it switches from B back to A. By induction, suppose that $f_t(ALG) \leq 3\min(c_t(A), c_t(B)) = 3c_t(B)$, where the equality follows since *ALG* switched from A to B at time t. In both cases, assume that caches are instantly reconciled. Then:

$$\begin{aligned}
f_{t'}(ALG) &= f_t(ALG) + (c_{t'}(B) - c_t(B)) \\
&= f_t(ALG) + 2c_{t'}(A) - {}^1\!/\!{}_2 c_t(A) \\
&\leq 3c_t(B) + 2(c_{t'}(A) - c_t(A)) + {}^3\!/\!{}_2 \cdot c_t(A) \\
&= 3c_t(A) + 2(c_{t'}(A) - c_t(A)) \\
&\leq 3c_{t'}(A) \\
&= 3\min(c_{t'}(A), c_{t'}(B))
\end{aligned}$$

What is left is to bound the following cost for the time since the last switch. Let s denote the time of the last switch and, assume without loss of generality that it was done from A to B. Let s' denote the last time step. By the previous set of inequalities (changing the second equation to inequality) and the fact that the algorithm never switched back to A after s, it holds that $f_{s'}(ALG) \leq 3c_{s'}(A) \leq 6\min(c_{s'}(A), c_{s'}(B))$.

To bound the reconciliation cost, assume the switch at time t is from A to B. We charge the reconciliation of each element in $B \setminus A$ to the cache miss when the element was last evicted by A. Therefore, the overall reconciliation cost is bounded by $r_t(ALG) \leq c_t(A) + c_t(B) \leq 3\min(c_t(A), c_t(B))$. □

Observe that the above construction can extend beyond caching and applies to any setting where we can bound the cost that the algorithm needs to incur to reconcile the states of the robust and the worst-case competitive algorithm. In particular, this occurs in the more general k-server problem.

Table 1. Number of Sequences; Sequence Length; Min and Max Number of Elements for Each Dataset

Dataset	Num Sequences	Sequence Length	Unique Elements
BK	100	2,101	67– 800
Citi	24	25,000	593 – 719

Remark 4.4. The above construction is similar to that of Fiat et al. [23] who showed how to combine multiple competitive algorithms. In subsequent work, Antoniadis et al. [9] relied on a similar construction to provide results for metrical task systems under a different prediction model.

5 EXPERIMENTS

In this section, we evaluate our approach on real-world datasets, empirically demonstrate its dependence on the errors in the oracle, and compare it to standard baselines.

Datasets and Metrics. We consider two datasets taken from different domains to demonstrate the wide applicability of our approach.

- BK is data extracted from BrightKite, a now defunct social network. We consider sequences of checkins, and extract the top 100 users with the longest non-trivial checkin sequences—those where the optimum policy would have at least 50 misses. This dataset is publicly available at [1] and [17]. Each of the user sequences represents an instance of the caching problem.
- `Citi` is data extracted from CitiBike, a popular bike-sharing platform operating in New York City. We consider CitiBike trip histories, and extract stations corresponding to starting points of each trip. We create 12 sequences, one for each month of 2017 for the New York City dataset. We consider only the first 25,000 events in each file. This data is publicly available at [2].

We give some additional statistics about each datasets in Table 1.

Our main metric for evaluation will be the *competitive ratio* of the algorithm, defined as the number of misses incurred by the particular strategy divided by the optimum number of misses.

Predictions. We run experiments with both synthetic predictions to showcase the sensitivity of our methods to learning errors, and with predictions using an of-the-shelf classifier, published previously [7].

- *Synthetic Predictions.* For each element, we first compute the true next arrival time, $y(t)$, setting it to $n + 1$ if it does not appear in the future. To simulate the performance of an ML system, we set $h(t) = y(t) + \epsilon$, where ϵ is drawn i.i.d. from a lognormal distribution with mean parameter 0 and standard deviation σ. We chose the lognormal distribution of errors to showcase the effect of rare but large failures of the learning algorithm. Finally, observe that, since we only compare the relative predicted times for each method, adding a bias term to the predictor would not change the results.
- *PLECO Predictions.* In their work, Anderson et al. [7] developed a simple framework to model repeat consumption, and published the parameters of their PLECO (Power Law with Exponential Cut Off) model for the BrightKite dataset. While their work focused on predicting the relative probabilities of each element (re)appearing in the subsequent time step, we modify it to predict the next time an element will appear. Specifically, we set $h(t) = t + 1/p(t)$, where $p(t)$ represents the probability that element that appeared at time t will re-appear at time $t + 1$.

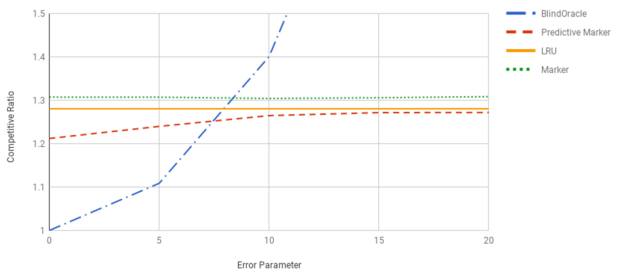

Fig. 1. Ratio of average number of evictions as compared to optimum for varying levels of oracle error.

Table 2. Competitive Ratio using PLECO Model

Algorithm	Competitive Ratio on BK	Competitive Ratio on Citi
Blind Oracle	2.049	2.023
LRU	1.280	1.859
Marker	1.310	1.869
Predictive Marker	1.266	1.810

Algorithms. We consider multiple algorithms for evaluation.

- *LRU* is the *Least Recently Used* policy that is wildly successful in practice.
- *Marker* is the classical Marker algorithm due to Fiat et al. [22].
- *Predictive Marker* is the algorithm we develop in this work. We set the switching cost to k, and therefore never switch to random evictions.
- *Blind Oracle* is the algorithm $\mathcal{B}$ described in Section 3.1, which evicts the element predicted to appear furthest in the future.

5.1 Results

We set $k = 10$ and summarize the synthetic results on the BK dataset in Figure 1. Observe that the performance of Predictive Marker is consistently better than LRU and standard Marker, and degrades slowly as the average error increases, as captured by the theoretical analysis. Second, we empirically verify that blindly following the oracle works well when the error is very low, but quickly becomes incredibly costly.

The results using the PLECO predictor are shown in Table 2, where we keep $k = 10$ for the BK dataset and set $k = 100$ for `Citi`; we note that the ranking of the methods is not sensitive to the cache size, k. We can again see that the Predictive Marker algorithm outperforms all others, and is 2.5% better than the next best method, LRU. While the gains appear modest, we note they are statistically significant at $p < 0.001$. Moreover, the off-the-shelf PLECO model was not tuned or optimized for predicting the *next* appearance of each element.

In that regard, the large difference in performance between using the predictor directly (Blind Oracle) and using it in combination with Marker (Predictive Marker) speaks to the power of the algorithmic method. By considering only the straightforward use of the predictor in the Blind Oracle setting, one may deem the ML approach not powerful enough for this application; what we show is that a more judicious use of the same model can result in tangible and statistically significant gains.

6 CONCLUSION

In this work, we introduce the study of online algorithms with the aid of machine learned predictors. This combines the empirical success of machine learning with the rigorous guarantees of online algorithms. We model the setting for the classical caching problem and give an oracle-based algorithm whose competitive ratio is directly tied to the accuracy of the machine learned oracle.

Our work opens up two avenues for future work. On the theoretical side, it would be interesting to see similar predictor-based algorithms for other online settings such as the k-server problem; this has already led to a fruitful line of current research as we discussed in Section 1.3. On the practical side, our caching algorithm shows how we can use machine learning in a safe way, avoiding problems caused by rare wildly inaccurate predictions. At the same time, our experimental results show that even with simple predictors, our algorithm provides an improvement compared to LRU. In essence, we have reduced the worst-case performance of the caching problem to that of finding a good (on average) predictor. This opens up the door for practical algorithms that need not be tailored towards the worst-case or specific distributional assumptions, but still yield provably good performance.

APPENDIX

A PROOF OF LEMMA 3.1

In this section, we provide the proof of the lemma connecting spread to absolute and squared loss. Before doing so, we provide a useful auxiliary lemma.

LEMMA A.1. *For odd $T = 2n + 1$, one pair (A_T, B_T) minimizing either absolute or squared loss subject to the constraints of the spread definition is $A_{2n+1} = (0 \ldots 2n)$ and $B_T = (n \ldots n)$.*

PROOF. First, we show that there exists a B_T minimizing the loss with $b_i = b_j$ for all i, j. Assume otherwise; then there exist two subsequent i, j with $b'_i > b'_j$. Since $a_i < a_j + 1$ by the assumption on spread, $\min_{x \in b_i, b_j}\{\ell(a_i, b) + \ell(a_j, b)\} \leq \ell(a_i, b_i) + \ell(a_j, b_j)$. Applying this recursively, we conclude that such a B_T exists.

Second, we show that there exist an A_T that consists of elements $a_{i+1} = a_i + 1$. Since the elements of B_T are all equal to b, the sequence $\sum_{i=0}^{2n} \ell(a_i, b)$ is minimized for both absolute and squared loss when $a_i = b + i - n$.

Last, the exact value of b does not make a difference and therefore we can set it to be $b = n$ concluding the lemma. □

LEMMA 3.1. restated: *For absolute loss, $\ell_1(A, B) = \sum_i |a_i - b_i|$, the spread of ℓ_1 is $S_{\ell_1}(m) \leq \sqrt{5m}$.*

For squared loss, $\ell_2(A, B) = \sum (a_i - b_i)^2$, the spread of ℓ_2 is $S_{\ell_2}(m) \leq \sqrt[3]{14m}$.

PROOF. It will be easier to restrict ourselves to odd $T = 2n + 1$ and also assume that $T \geq 3$. This will give an upper bound on the spread (which is tight up to small constant factors). By Lemma A.1, a pair of sequence minimizing absolute/squared loss is $A_T = (0, \ldots, 2n)$ and $B_T = (n, \ldots, n)$. We now provide bounds on the spread based on this sequence, that is we find a $T = 2n + 1$ that satisfies the inequality $\ell(A_T, B_T) \leq m$.

Absolute Loss. The absolute loss of the above sequence is:

$$\ell(A_T, B_T) = 2 \cdot \sum_{j=1}^{n} j = 2 \cdot \frac{n(n+1)}{2} = n(n+1) = \frac{T-1}{2} \cdot \frac{T+1}{2} = \frac{T^2 - 1}{4}.$$

A T that makes $\ell(A_T, B_T) \geq m$ is $T = \sqrt{4m + 1}$. Therefore, for absolute loss $S_\ell(m) \leq \sqrt{5m}$, since $m \geq 1$.

Squared Loss. The squared loss of the above sequence is:

$$\ell(A_T, B_T) = 2 \cdot \sum_{j=1}^{n} j^2 = 2 \cdot \frac{n(n+1)(2n+1)}{6} = \frac{(T^2-1)\cdot T}{12} = \frac{T^3 - T}{12} \geq \frac{8T^3}{9 \cdot 12} = \frac{2T^3}{27}$$

where the inequality holds because $T \geq 3$.

A T that makes $\ell(A_T, B_T) \geq m$ is $T = \sqrt[3]{14m}$. Therefore, for squared loss, $S_\ell(m) \leq \sqrt[3]{14m}$. □

ACKNOWLEDGMENTS

The authors would like to thank Andrés Muñoz-Medina and Éva Tardos for valuable discussions on the presentation of the article, Shuchi Chawla and Seffi Naor for useful feedback regarding Section 3.4, Ola Svensson for suggesting the locality extension (Theorem 4.2) as well as an anonymous reviewer for pointing towards the direction of Theorem 4.3.

REFERENCES

[1] [n.d.]. Brightkite data. http://snap.stanford.edu/data/loc-brightkite.html.

[2] [n.d.]. Citibike System Data. http://https://www.citibikenyc.com/system-data.

[3] Dimitris Achlioptas, Marek Chrobak, and John Noga. 2000. Competitive analysis of randomized paging algorithms. *Theoret. Comput. Sci.* 234, 1-2 (2000), 203–218. DOI: https://doi.org/10.1016/S0304-3975(98)00116-9

[4] Nir Ailon, Bernard Chazelle, Kenneth L. Clarkson, Ding Liu, Wolfgang Mulzer, and C. Seshadhri. 2011. Self-improving algorithms. *SIAM J. Comput.* 40, 2 (2011), 350–375. DOI: https://doi.org/10.1137/090766437

[5] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. 2002. On paging with locality of reference. In *Proceedings of the T34th Annual ACM Symposium on Theory of Computing (STOC '02)*. ACM, New York,, 258–267. DOI: https://doi.org/10.1145/509907.509949

[6] Keerti Anand, Rong Ge, and Debmalya Panigrahi. 2020. Customizing ML predictions for online algorithms. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

[7] Ashton Anderson, Ravi Kumar, Andrew Tomkins, and Sergei Vassilvitskii. 2014. The dynamics of repeat consumption. In *Proceedings of the 23rd International Conference on World Wide Web (WWW '14)*. ACM, New York,, 419–430. DOI: https://doi.org/10.1145/2566486.2568018

[8] Spyros Angelopoulos, Christoph Dürr, Shendan Jin, Shahin Kamali, and Marc Renault. 2019. Online computation with untrusted advice. *arXiv preprint arXiv:1905.05655* (2019).

[9] Antonios Antoniadis, Christian Coester, Marek Elias, Adam Polak, and Simon Betrand. 2020. Online metric algorithms with untrusted predictions. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

[10] David Arthur, Bodo Manthey, and Heiko Röglin. 2011. Smoothed analysis of the k-means method. *J. ACM* 58, 5 (2011), 19:1–19:31. DOI: https://doi.org/10.1145/2027216.2027217

[11] David Arthur and Sergei Vassilvitskii. 2006. Worst-case and smoothed analysis of the ICP algorithm, with an application to the k-means method. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2006), 21-24 October 2006, Berkeley, California, USA, Proceedings*. 153–164. DOI: https://doi.org/10.1109/FOCS.2006.79

[12] Eric Balkanski, Aviad Rubinstein, and Yaron Singer. 2017. The limitations of optimization from samples. In *Proceedings of the Smposium on the Theory of Computing (STOC)*. http://arxiv.org/abs/1512.06238

[13] L. A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Syst. J.* 5, 2 (June 1966), 78–101. DOI: https://doi.org/10.1147/sj.52.0078

[14] Allan Borodin and Ran El-Yaniv. 1998. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, NY, USA.

[15] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. 2016. Online algorithms with advice: A survey. *SIGACT News* 47, 3 (Aug. 2016), 93–129. DOI: https://doi.org/10.1145/2993749.2993766

[16] Sébastien Bubeck and Aleksandrs Slivkins. 2012. The best of both worlds: Stochastic and adversarial bandits. In *COLT 2012 - The 25th Annual Conference on Learning Theory, June 25-27, 2012, Edinburgh, Scotland*. 42.1–42.23. http://www.jmlr.org/proceedings/papers/v23/bubeck12b/bubeck12b.pdf.

[17] Eunjoon Cho, Seth A. Myers, and Jure Leskovec. 2011. 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 (KDD '11)*. ACM, New York, NY, USA, 1082–1090. DOI: https://doi.org/10.1145/2020408.2020579

[18] Richard Cole and Tim Roughgarden. 2014. The sample complexity of revenue maximization. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014.* 243–252. DOI: https://doi.org/10.1145/2591796.2591867

[19] Peter J. Denning. 1968. The working set model for program behavior. *Commun. ACM* 11, 5 (May 1968), 323–333. DOI: https://doi.org/10.1145/363095.363141

[20] Matthias Englert, Heiko Röglin, and Berthold Vöcking. 2016. Smoothed analysis of the 2-opt algorithm for the general TSP. *ACM Trans. Algorithms* 13, 1 (2016), 10:1–10:15. DOI: https://doi.org/10.1145/2972953

[21] Hossein Esfandiari, Nitish Korula, and Vahab Mirrokni. 2015. Online allocation with traffic spikes: Mixing adversarial and stochastic models. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation.* ACM, 169–186.

[22] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. 1991. Competitive paging algorithms. *J. Algorithms* 12, 4 (Dec. 1991), 685–699. DOI: https://doi.org/10.1016/0196-6774(91)90041-V

[23] Amos Fiat, Yuval Rabani, and Yiftach Ravid. 1994. Competitive k-server algorithms. *J. Comput. System Sci.* 48, 3 (1994), 410–428. DOI: https://doi.org/10.1016/S0022-0000(05)80060-1

[24] Sreenivas Gollapudi and Debmalya Panigrahi. 2019. Online algorithms for rent-or-buy with expert advice. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, California, USA, 2319–2327. http://proceedings.mlr.press/v97/gollapudi19a.html.

[25] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. 2019. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations (ICLR).*

[26] Piotr Indyk, Frederik Mallmann-Trenn, Slobodan MitroviÄĞ, and Ronitt Rubinfeld. 2020. Online Page Migration with ML Advice. arxiv:cs.DS/2006.05028

[27] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. 2020. Online algorithms for weighted paging with predictions. In *Proceedings of the 47th International Colloquium on Automata, Languages and Programming (ICALP).*

[28] Tim Kraska, Alex Beutel, Ed H. Chi, Jeff Dean, and Neoklis Polyzotis. 2017. The case for learned index structures. https://arxiv.org/abs/1712.01208.

[29] Silvio Lattanzi, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. 2020. Online scheduling via learned weights. In *Proceedings of the Twenty-Third Annual Symposium on Discrete Algorithms (SODA).*

[30] Thodoris Lykouris, Vahab Mirrokni, and Renato Paes Leme. 2018. Stochastic bandits robust to adversarial corruptions. In *Proceedings of the 50th ACM Annual Symposium on Theory of Computing (STOC).*

[31] Thodoris Lykouris and Sergei Vassilvtiskii. 2018. Competitive caching with machine learned advice. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research).* PMLR, 3296–3305.

[32] Mohammad Mahdian, Hamid Nazerzadeh, and Amin Saberi. 2012. Online optimization with uncertain information. *ACM Trans. Algorithms* 8, 1 (2012), 2:1–2:29. DOI: https://doi.org/10.1145/2071379.2071381

[33] Andrew McGregor. 2014. Graph stream algorithms: A survey. *SIGMOD Rec.* 43, 1 (May 2014), 9–20. DOI: https://doi.org/10.1145/2627692.2627694

[34] Andres Muñoz Medina and Sergei Vassilvitskii. 2017. Revenue optimization with approximate bid predictions. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA.* 1856–1864.

[35] Vahab S. Mirrokni, Shayan Oveis Gharan, and Morteza Zadimoghaddam. 2012. Simultaneous approximations for adversarial and stochastic online budgeted allocation. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012.* 1690–1701. http://portal.acm.org/citation.cfm?id=2095250&CFID=63838676&CFTOKEN=79617016.

[36] Michael Mitzenmacher. 2018. A model for learned bloom filters and optimizing by sandwiching. In *Advances in Neural Information Processing Systems (NeurIPS).*

[37] Michael Mitzenmacher. 2020. Queues with small advice. *CoRR* abs/2006.15463 (2020). arxiv:2006.15463https://arxiv.org/abs/2006.15463.

[38] Jamie Morgenstern and Tim Roughgarden. 2016. Learning simple auctions. In *Proceedings of the 29th Conference on Learning Theory, COLT 2016, New York, USA, June 23-26, 2016.* 1298–1318. http://jmlr.org/proceedings/papers/v49/morgenstern16.html.

[39] Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms.* Cambridge University Press, New York, NY, USA.

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

[41] Alexander Rakhlin and Karthik Sridharan. 2013. Online learning with predictable sequences. In *Proceedings of the 26th Annual Conference on Learning Theory (COLT).*

[42] Dhruv Rohatgi. 2020. Near-optimal bounds for online caching with machine learned advice. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, Shuchi Chawla (Ed.). SIAM, 1834–1845. DOI: https://doi.org/10.1137/1.9781611975994.112

[43] D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS'15)*. MIT Press, Cambridge, MA, USA, 2503–2511. http://dl.acm.org/citation.cfm?id=2969442.2969519.

[44] Daniel D. Sleator and Robert E. Tarjan. 1985. Amortized efficiency of list update and paging rules. *Commun. ACM* 28, 2 (Feb. 1985), 202–208. DOI: https://doi.org/10.1145/2786.2793

[45] Daniel A. Spielman and Shang-Hua Teng. 2004. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM* 51, 3 (2004), 385–463. DOI: https://doi.org/10.1145/990308.990310

[46] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. In *International Conference on Learning Representations.* http://arxiv.org/abs/1312.6199.

[47] Kapil Vaidya, Eric Knorr, Tim Kraska, and Michael Mitzenmacher. 2020. Partitioned learned bloom filter. *CoRR* abs/2006.03176 (2020). arxiv:2006.03176 https://arxiv.org/abs/2006.03176.

[48] Alexander Wei. 2020. Better and simpler learning-augmented online caching. In *International Conference on Approximation Algorithms for Combinatorial Optimization Problems (APPROX).*

Received August 2019; revised August 2020; accepted January 2021