Robust Learning-Augmented Caching: An Experimental Study

Jakub Chłędowski [*1] **Adam Polak** [*2] **Bartosz Szabucki** [*1] **Konrad Żołna** [*1 3]

Abstract

Effective caching is crucial for the performance of modern-day computing systems. A key optimization problem arising in caching – which item to evict to make room for a new item – cannot be optimally solved without knowing the future. There are many classical approximation algorithms for this problem, but more recently researchers started to successfully apply machine learning to decide what to evict by discovering implicit input patterns and predicting the future. While machine learning typically does not provide any worst-case guarantees, the new field of learning-augmented algorithms proposes solutions that leverage classical online caching algorithms to make the machine-learned predictors robust. We are the first to comprehensively evaluate these learning-augmented algorithms on real-world caching datasets and state-of-the-art machine-learned predictors. We show that a straightforward method – blindly following either a predictor or a classical robust algorithm, and switching whenever one becomes worse than the other – has only a low overhead over a well-performing predictor, while competing with classical methods when the coupled predictor fails, thus providing a cheap worst-case insurance.

1. Introduction

Caching is an important part of almost any modern-day computing system because it can vastly speed up memory access. A major optimization problem arising with regard to caching is: *Which item to evict from the cache in order to make room for a new item when the cache is full?* The optimization goal is to maximize the number of *cache hits*, i.e., situations when the requested item is still present in the cache. If we choose a wrong item to evict and it is requested again soon after, a *cache miss* occurs, and the item has to be reloaded to the cache from the main memory[1], which is usually orders of magnitude slower than reading it directly from the cache.

Classical caching algorithms. In the *offline* scenario, i.e., when we know in advance the sequence of requested items, the problem is easy to solve optimally. Indeed, Belady (1966) proved that the number of cache misses is minimized by a greedy eviction policy – always evict the item which will reappear the furthest in the future (or which will never reappear, if there is such item).

In the more realistic *online* scenario, we do not know the future requests. For the cache size of k items, the classical MARKER algorithm (Fiat et al., 1991) is $\mathcal{O}(\log k)$-competitive, i.e. it incurs at most $\mathcal{O}(\log k \cdot \text{OPT})$ cache misses on inputs which the optimal offline algorithm serves with OPT misses. There is also a matching lower bound (Fiat et al., 1991), showing that no algorithm can do better, up to a constant factor. On the other hand, real-world applications employ simple heuristics, such as the gold standard Least Recently Used (LRU), which happens to perform better in practice.

Recent machine learning approaches try to discover implicit access patterns specific to individual applications, which are likely to occur in future request sequences and use that knowledge to make better eviction decisions (Jain & Lin, 2016; Shi et al., 2019; Liu et al., 2020; Yan & Li, 2020).

Learning-augmented caching algorithms. In general, algorithms based on machine learning models tend to work well on typical inputs but can perform arbitrarily badly when, e.g. training data is scarce, or input patterns change unexpectedly over time. Lykouris & Vassilvitskii (2018) proposed a workaround to that issue. Their online caching algorithm, PredictiveMarker, takes as an additional input for each requested item a prediction (e.g. generated by an ML model) when this item will be requested again. The

[*]Equal contribution [1]Jagiellonian University, Kraków, Poland [2]EPFL, Lausanne, Switzerland [3]DeepMind, London, United Kingdom. Correspondence to: Jakub Chłędowski <jakub.chledowski@gmail.com>, Adam Polak <adam.polak@epfl.ch>, Bartosz Szabucki <bartosz.szabucki@gmail.com>, Konrad Żołna <konrad.zolna@gmail.com>.

Proceedings of the 38^{th} International Conference on Machine Learning, PMLR 139, 2021.

[1]Even if a requested item is unlikely to be ever reused, it has to be put into the cache.

algorithm is *consistent*, i.e., it incurs an almost optimal number of cache misses when given nearly perfect predictions, and *robust*, i.e., it is $O(\log k)$-competitive (just like the optimal MARKER algorithm) even when the predictions are completely wrong.[2] Formally, PredictiveMarker incurs at most

$$\mathcal{O}\left(\mathrm{OPT}\cdot\min\left(\sqrt{\frac{\eta_{\mathrm{reuse}}}{\mathrm{OPT}}}+2,\log k\right)\right)$$

cache misses, where η_{reuse} denotes the total $L1$ error of the predictor. Rohatgi (2020) and Wei (2020) came up with more caching algorithms working in this setup, with improved dependencies on the prediction error η_{reuse}.

Antoniadis et al. (2020) proposed a different setup for learning-augmented caching algorithms. In their setup, a predictor has to keep guessing what the optimal (knowing the future) policy would do. In principle, any caching algorithm can serve as a predictor itself. The prediction error η_{cache} is measured as the size of the symmetric difference between cache configurations of the optimal algorithm and the predictor, summed over all time steps. Actually, the setup works for any *metrical task system* (MTS), a general class of online problems that includes caching. Antoniadis et al. (2020) provide two consistent and robust algorithms (one deterministic and one randomized) for general MTS, and specifically for caching, they propose an algorithm, dubbed Trust&Doubt, with a better dependence on prediction error.

Motivation. A comprehensive comparison of the above learning-augmented caching algorithms is a difficult task. Admittedly, worst-case competitive ratios of algorithms within the same setup can be compared, but such theoretical comparison between the two setups is implausible. It follows from the fact that the final competitive ratios depend on coupled predictors, with different error measures that can not be translated between the setups.

To the best of our knowledge, so far there is no experimental evaluation of these algorithms using real-world datasets nor predictors. Experiments were included only in works by Lykouris & Vassilvitskii (2018) and Antoniadis et al. (2020). However, these are small-size proof-of-concept experiments on datasets adapted from other problems (not related to caching) and using simple ad-hoc predictors instead of fully-fledged machine learning models. The following question remains wide open.

Are learning-augmented caching algorithms practical?

Our study. In this paper, we set out to answer this question experimentally. We use benchmark dataset from the 2nd Cache Replacement Championship (CRC, 2017), also used by Shi et al. (2019) and Liu et al. (2020). As a predictor, we use state-of-the-art machine-learning-based caching algorithm Parrot (Liu et al., 2020). Conveniently, Parrot also predicts when items reappear in the request sequence, on top of predicting which item the optimal policy discards. It can thus be used as a predictor for learning-augmented algorithms in both setups, by Lykouris & Vassilvitskii (2018) and by Antoniadis et al. (2020), respectively.

First, we test how the algorithms perform with a fully-fledged predictor so that we can see if the overhead they incur is an acceptable cost to pay in exchange for the worst-case guarantees they provide. Second, we also run a scenario with a predictor under-performing due to the scarcity of available training data. That lets us evaluate if the theoretical robustness guarantees play a role in practice.

Our secondary contribution is a ready-to-use benchmark, which facilitates an easier future comparison with our results. It is based on a dataset which was previously used by Liu et al. (2020). However, since the method they use to generate input instances tends to have high variance, by providing the ready-to-use inputs we make it possible to compare directly with the numbers we report, without having to rerun all the experiments.

2. Background and Algorithms

In this section, we aim to concisely describe the background needed for the understanding of the following sections. We will start by explaining the two prediction setups for learning-augmented caching. Next, we will continue with a short overview of the learning-augmented algorithms used in each setup. At the end, we will describe Parrot (Liu et al., 2020), the neural network predictor that we use to generate predictions for the algorithms.

2.1. Prediction setups for caching

Lykouris & Vassilvitskii (2018) proposed the first prediction setup for online caching. In their setup, after each request, the predictor has to forecast when the requested item will be requested again – the value called *reuse distance*. This is a very natural choice since the reuse distance is the only statistic that Belady's optimal offline algorithm looks at. They define the prediction error η_{reuse} to be the $L1$ distance, i.e., the sum over all requests of the absolute difference between the actual and predicted reuse distances.

Antoniadis et al. (2020) noticed a limitation of this setup – it does not generalize to other online problems. Already for the weighted caching problem (where the cost of loading each item can be different) even perfect reuse distance predictions are not sufficient to beat the best classical prediction-less algorithm. To address this limitation,

[2] The exact meaning of consistency and robustness seems to be somewhat inconsistent throughout the literature.

Table 1. **Classical and learning-augmented caching algorithms.** Constants in competitive ratios are omitted for brevity.

Algorithm	Prediction	Competitive ratio	Combiner	Reference
OPT	n/a	1	n/a	Belady (1966)
LRU	n/a	k	n/a	folklore
MARKER	n/a	$\log k$	n/a	Fiat et al. (1991)
PredictiveMarker	reuse distance	$\min(\log k, \sqrt{\eta_{\text{reuse}}/\text{OPT}})$	n/a	Lykouris & Vassilvitskii (2018)
LMarker	reuse distance	$\min(\log k, \log(\eta_{\text{reuse}}/\text{OPT}))$	n/a	Rohatgi (2020)
LNonMarker$^{\text{D}}$	reuse distance	$\min(\log k, {}^{\log k}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	deterministic	Rohatgi (2020)
LNonMarker$^{\text{R}}$	reuse distance	$\min(\log k, {}^{\log k}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	randomized	Rohatgi (2020)
BlindOracle$^{\text{D}}$	reuse distance	$\min(\log k, {}^{1}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	deterministic	Wei (2020)
BlindOracle$^{\text{R}}$	reuse distance	$\min(\log k, {}^{1}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	randomized	Wei (2020)
RobustFtP$^{\text{D}}$	optimal policy	$\min(\log k, \eta_{\text{cache}}/\text{OPT})$	deterministic	Antoniadis et al. (2020)
RobustFtP$^{\text{R}}$	optimal policy	$\min(\log k, \eta_{\text{cache}}/\text{OPT})$	randomized	Antoniadis et al. (2020)
Trust&Doubt	optimal policy	$\min(\log k, \log(\eta_{\text{cache}}/\text{OPT}))$	n/a	Antoniadis et al. (2020)

they proposed an alternative setup that works for *metrical task systems* – a general class of online problems, which includes caching. In their setup, after each request, the predictor has to guess what an optimal offline algorithm would do. The prediction error η_{cache} is the size of the symmetric difference between the caches maintained by the optimal algorithm and the predictor, summed over time.

A direct comparison of the two setups is problematic for at least two reasons. First, even though predictions for the first setup can be translated (by following Belady's rule) to predictions for the second setup (but not the other way round), the respective errors cannot be related to each other, as shown by the two instructive examples in Antoniadis et al. (2020, Sect. 1.3). Second, a priori, it is not clear which of the two types of predictors is easier to train well. On the one hand, predicting reuse distances can be framed as a standard supervised learning task, while predicting optimal policy seems to require more advanced approaches such as imitation learning or reinforcement learning. On the other hand, one can imagine an input distribution such that it is hard to accurately predict reuse distances while it is still easy to always find an item with a reuse distance likely so large that it is safe to evict.

2.2. Augmented algorithms

We evaluate the six learning-augmented caching algorithms proposed up to date. PredictiveMarker (Lykouris & Vassilvitskii, 2018), LMarker and LNonMarker (Rohatgi, 2020), and BlindOracle (Wei, 2020) are all augmented with reuse distance predictions, while RobustFtP and Trust&Doubt (Antoniadis et al., 2020) utilize optimal policy predictions. See Table 1 for a summary of these algorithms.

Combiners. One way to achieve robustness is to combine a non-robust learning-augmented algorithm with a robust classical algorithm, e.g. MARKER, using a *combiner*. A combiner is a procedure that takes two algorithms and uses them in a black-box way to perform on par with the better of the two algorithms on each input (up to a constant factor).

A straightforward combiner can simulate the two algorithms, keep track of their respective costs up to date, and switch between them whenever one heavily outperforms the other. The idea dates back to Fiat et al. (1994), and it was adapted to the learning-augmented setting by Lykouris & Vassilvitskii (2018). We will call the above combiner *deterministic*. In contrast, Wei (2020) and Antoniadis et al. (2020) use an idea of Blum & Burch (2000) to provide an alternative *randomized* combiner. The more intricate combining algorithm allows to bring down the multiplicative overhead to $1 + \varepsilon$ at the cost of an extra additive constant depending on ε.

Some learning-augmented algorithms are built using combiners, while others achieve robustness out-of-the-box. In principle, each of the algorithms implementing a combiner can have (at least) four variants. It can be paired with either MARKER or LRU[3], and, independently of that choice, it can use either the deterministic or the randomized combiner.

Algorithms with reuse distance. PredictiveMarker, LMarker, and LNonMarker all work by keeping track of *eviction chains*. An eviction chain is a sequence of suboptimal evictions where each eviction is forced by the cache-miss that can be blamed on the previous eviction in the sequence. The three algorithms differ with respect to (1) for how long they trust predictions in each eviction chain, and (2) what they do instead when they eventually lose the

[3]Formally, combining with LRU does not yield robustness, since LRU is not $\mathcal{O}(\log k)$-competitive. However, it has at least a provably bounded competitive ratio, and its good practical performance is well understood. Hence, in practice, it makes sense to use LRU as a fallback option for a potentially arbitrarily inaccurate machine-learned algorithm.

Table 2. **Characteristics of our datasets.** The first row shows the total sizes of all used datasets. These sizes are later split into train/valid/test sets with 80%/10%/10% splits. Further rows display the cache hit rates of pure (non-learning-augmented) algorithms, illustrating varying difficulties of the datasets.

	astar	bwaves	bzip	cactusadm	gems	lbm	leslie3d	libq	mcf	milc	omnetpp	sphinx3	xalanc
Size	1,154,048	570,368	167,680	221,952	723,456	782,080	716,032	579,840	2,965,504	556,800	555,520	328,704	69,120
Cache hits													
OPT	37.4%	4.9%	80.8%	33.7%	12.7%	24.8%	30.9%	5.3%	44.6%	1.4%	42.4%	74.8%	56.9%
RANDOM	8.3%	0.2%	56.5%	4.5%	3.9%	2.2%	9.5%	0.0%	20.5%	0.0%	17.6%	52.8%	36.8%
LRU	4.0%	0.0%	63.8%	0.0%	2.9%	0.0%	9.5%	0.0%	27.1%	0.0%	20.4%	12.7%	45.4%
MARKER	4.7%	0.0%	62.7%	1.3%	4.1%	0.0%	9.3%	0.0%	24.9%	0.0%	20.3%	42.2%	43.5%
PARROT-REUSE	29.0%	0.1%	57.9%	21.8%	0.4%	0.5%	4.9%	5.3%	32.5%	1.1%	11.9%	67.6%	37.3%
PARROT-CACHE	32.2%	0.3%	68.4%	32.9%	3.1%	0.0%	11.4%	0.0%	43.9%	0.0%	21.9%	70.6%	49.7%

trust. We refer to the original papers for detailed discussions of these strategies, designed to allow better and better for worst-case competitive ratios, expressed as a function of the normalized error $\eta_{\text{reuse}}/\text{OPT}$. PredictiveMarker and LMarker are provably robust out-of-the-box. In contrast, LNonMarker achieves robustness thanks to the classical MARKER algorithm and a black-box combiner.

The next learning-augmented caching algorithm, BlindOracle, simply applies Belady's rule to the prediction – i.e. it evicts the item with the predicted next arrival time furthest in the future – and adds the guarantee of a robust algorithm (e.g. MARKER) using either of the aforementioned combiners. We note that, despite being the simplest out of the four, BlindOracle also has the best competitive ratio, though the proof is much more involved than the algorithm itself.

Algorithms with policy predictions. RobustFtP stands for *Robust Follow-the-Prediction*. It is based on essentially the same algorithmic idea as BlindOracle – i.e. apply a combiner to (1) robust MARKER and (2) consistent following the predictions – but in the alternative prediction setup and with a quite different theoretical analysis.

Trust&Doubt utilizes predictions in a more intricate way in order to achieve better dependence on the prediction error η_{cache}. Similar to PredictiveMarker and LMarker, it is robust out-of-the-box and hence does not require a combiner.

2.3. Predictor

We couple the learning-augmented algorithms with the state-of-the-art predictors. Specifically, Liu et al. (2020) proposed Parrot, an imitation learning algorithm that tries to mimic Belady's oracle policy with a deep neural network.

At each step, the model is input the last $H = 30$ requested items (including the current one) and the $k = 16$ items that are at the time in the cache (in the relevant set). All these items are encoded to fixed-size embedding, and the requested items are additionally processed by LSTM (Hochreiter & Schmidhuber, 1997), which is continuously fed with them one by one. That leads to obtaining $H + k$ embeddings, which are further processed with attention (specifically, Transformer (Vaswani et al., 2017) and BiDAF (Seo et al., 2017)), which finally outputs one vector per cache item. These elements are passed through a linear layer with softmax to form a prediction.

The model uses two different prediction heads, which are optimized simultaneously during training. The first one predicts which element would be evicted from the cache by Belady's oracle and defines the PARROT-CACHE predictor, which is exactly what the RobustFtP and Trust&Doubt algorithms need.

The second head estimates, for each element in the cache, the number of steps before the element is requested again. We use that estimate for the last requested item to construct the second predictor, which we call PARROT-REUSE. Such predictions are compatible with the PredictiveMarker, LMarker, LNonMarker, and BlindOracle algorithms.

3. Datasets

Our datasets come from the 2nd Cache Replacement Championship (CRC, 2017) and consist of real-world memory access traces from the SPEC CPU2006 benchmark (Henning, 2006).

There are two ways to obtain the traces to begin with. One can either download the traces released in the Cache Replacement Championship or collect custom traces using a dynamic binary instrumentation tool DynamoRIO (Bruening et al., 2003). Liu et al. (2020) used the second method to evaluate Parrot and shared their procedure, but they were unable to release their exact traces. We follow this procedure to create datasets from the publicly available traces (CRC, 2017). In particular, we subsample the traces by choosing 64 out of 2048 *sets*[4] and filtering the accesses[5] to those sets. The first 80% of this sequence is used for training, followed

[4] A *set-associative* cache is split into n sets, each holding k items, called *lines*. Each line's address uniquely predetermines the set it can be cached in. In that sense, each trace constitutes n independent instances of the caching problem. (Here $k = 16$.)

[5] Following Shi et al. (2019) and Liu et al. (2020) we evaluate our approach on the last level of a three-level cache hierarchy.

by 10% used for validation, and the last 10% for testing. We refer to Table 2 for details of the datasets.

We noticed that even slight differences in the data collection and postprocessing might create datasets of significantly different characteristics. To account for that, we have reevaluated Parrot on our inputs, and made them publicly available at https://github.com/chledowski/Robust-Learning-Augmented-Caching-An-Experimental-Study-Datasets.

4. Experimental Setup

For both Parrot and the learning-augmented algorithms, we use the code made available by Liu et al. (2020) and Antoniadis et al. (2020), respectively. We unified both codebases and connected them into a single pipeline. The source code is publicly available at https://github.com/chledowski/ml_caching_with_guarantees.

Due to constrained computing resources, we limited the number of training steps to 20 000. The second change is to ablate DAgger (Ross et al., 2011), as it did not yield any improvements in our case. No other Parrot's hyperparameter was changed.

As a result, our models are trained for 20 000 steps on each dataset, with a batch size of 32. The best model is chosen to be the one with the highest validation cache hit rate among the evaluated checkpoints (done every 5000 steps).

To measure the practicality of learning-augmented caching algorithms, we will weaken the Parrot model by training only on prefixes (e.g. 1%) of the original datasets. We will analyze how the learning-augmented algorithms perform under such a change.

As we mention in Section 2.2, each of the algorithms implementing a combiner (LNonMarker, BlindOracle, and RobustFtP) have four variants, using the deterministic or the randomized combiner, with MARKER or with LRU. We noticed that MARKER and LRU perform on par with each other on most of the datasets that we use. Hence, for simplicity, we analyze only the two variants with MARKER (deterministic and randomized). For completeness, we also ran experiments with LRU, but as expected the choice turns out to be irrelevant.

5. Results

In this section, we analyze our experiments on the practicality of the learning-augmented algorithms. We begin with full training sets to assess algorithms' overhead over fully-fledged predictors. Then we move to predictors trained on only 1% of data to check whether the algorithms are robust in practice. We end with a closer look at two datasets, which we further subsample to better illustrate the behavior of the algorithms coupled with predictors of varying accuracy.

To make comparison across datasets easier, we normalize scores to both LRU and Belady's OPT at the same time. Specifically, for an algorithm ALG with empirical competitive ratio[6] cr_{ALG}, we report the *LRU-normalized empirical competitive ratio*, i.e.,

$$\frac{\mathrm{cr}_{ALG} - 1}{\mathrm{cr}_{LRU} - 1}.$$

The lower the value, the better, meaning that the algorithm's performance is closer to Belady's optimal oracle. For the sake of completeness, the raw unnormalized scores are included in the supplementary material.

5.1. State-of-the-art predictor

In order to be applicable in practice, a learning-augmented caching algorithm should have a low overhead. In other words, when the underlying predictor is accurate, the final performance should be as close to the predictor's performance as possible. In our first experiment, we aim to test which of the learning-augmented caching algorithms fulfill this key requirement.

To this end, for each dataset considered, we train the Parrot model on the full training set, which leads to obtaining two state-of-the-art predictors – PARROT-REUSE and PARROT-CACHE. Then, we couple each learning-augmented caching algorithm with one of the predictors, depending on the nature of the algorithm. The results are presented in Figure 1.

The bars correspond to the augmented algorithms, while horizontal lines reflect the performance of non-augmented methods. PARROT-CACHE is almost always better than the classical LRU baseline and even approaches optimal behavior for a few datasets. PARROT-REUSE performs significantly worse than PARROT-CACHE and even lags behind LRU in a few cases. It makes the use of algorithms coupled with PARROT-CACHE preferable, at least with the currently best available predictors.

When a predictor is better than MARKER, all learning-augmented algorithms leverage its accurate predictions to improve over the classical baseline.

However, only the two simplest algorithms – BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$ – have overheads over predictors low enough to be considered practical. Note that these two are based on the same rule, just applied in two different settings. The rule is to blindly follow either a predictor or MARKER, and switch whenever one heavily outperforms the other (see Section 2.2). The versions of these

[6]The *empirical competitive ratio* of the algorithm ALG is defined as $\mathrm{cr}_{ALG} = \mathrm{cost}_{ALG}/\mathrm{cost}_{OPT}$, where cost_{ALG} denotes the number of cache misses that algorithm ALG incurs on a dataset.

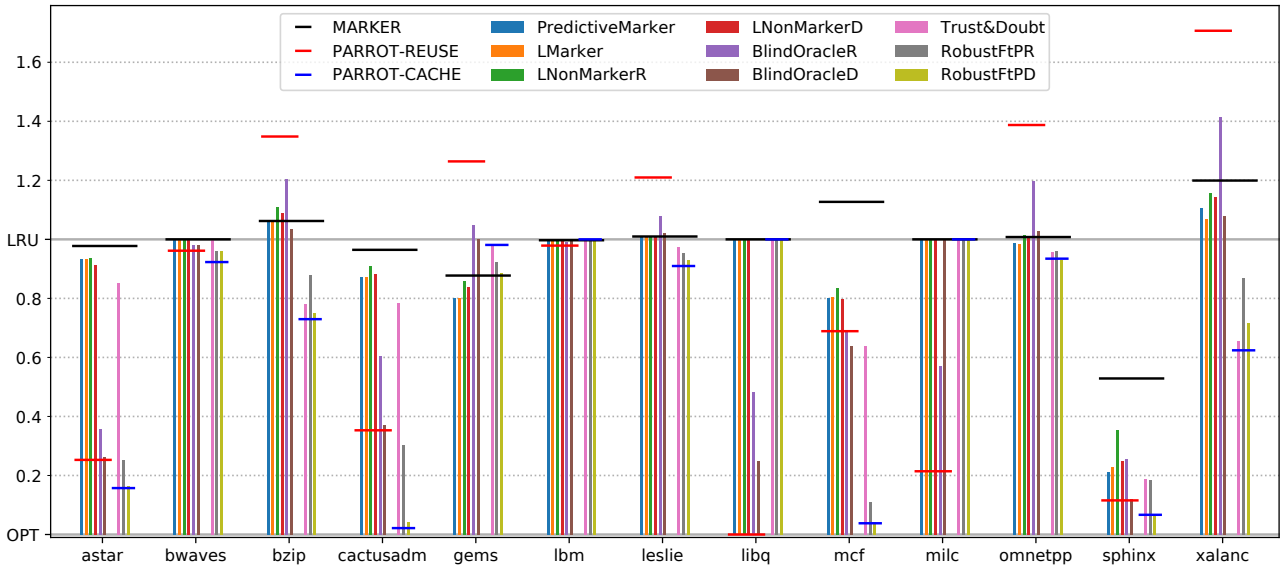

Figure 1. **Normalized cost (the lower the better) of learning-augmented algorithms compared to coupled predictors and classical algorithms.** The bars reflect the performance of the augmented algorithms, while horizontal lines correspond to non-augmented methods. The scores reflect LRU-normalized empirical competitive ratio (defined in Section 5). The predictor PARROT-CACHE (blue line) outperforms PARROT-REUSE (red line) and even approaches optimal policy for a few datasets. MARKER (black line) is comparable to LRU. The best augmented methods are RobustFtP$^{\text{D}}$ (yellow bar) and BlindOracle$^{\text{D}}$ (brown bar) which have very low overhead while stay robust when the coupled predictor performs worse than MARKER. RobustFtP$^{\text{D}}$ is, however, coupled with better predictors and hence significantly outperforms BlindOracle$^{\text{D}}$. The remaining augmented algorithms are overly conservative and hence fail to leverage good predictions.

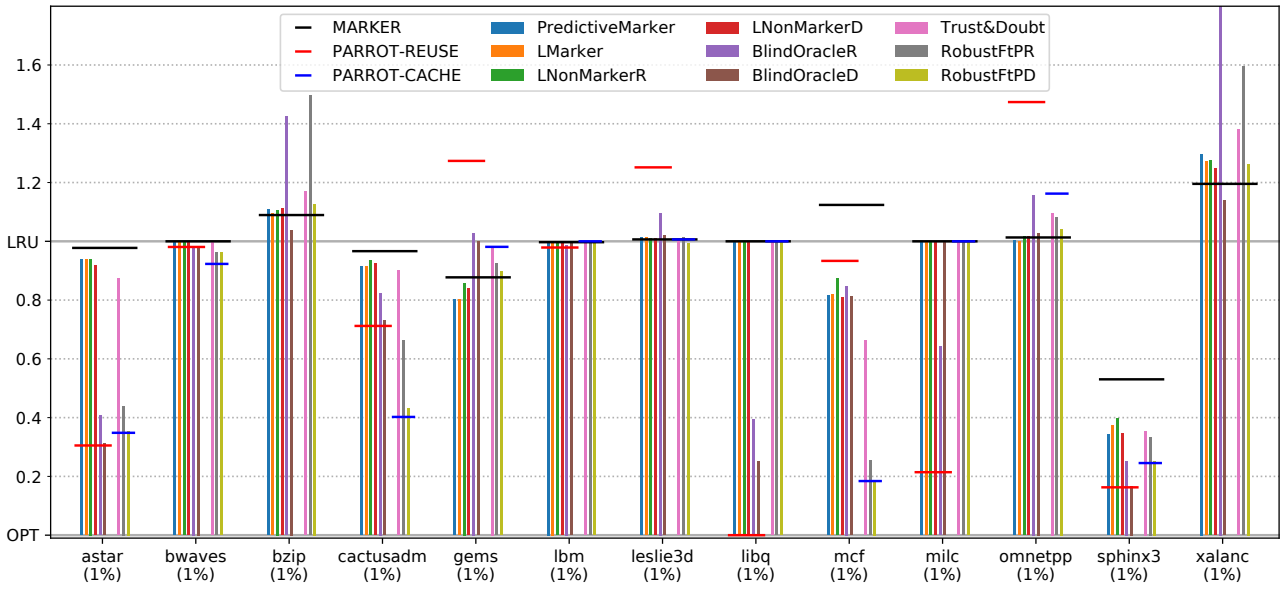

Figure 2. **Normalized cost (the lower the better). Underperforming predictors trained on 1% of data.** The predictors (red and blue lines) significantly outperform MARKER (black line) for only 4 datasets due to limited training data. Augmented algorithms (bars) are robust – they approach MARKER (black line), even when coupled predictors are wildly inaccurate. The robustness of RobustFtP$^{\text{R}}$ (violet bar) and BlindOracle$^{\text{R}}$ (gray bar) seems the weakest (especially for bzip and xalanc datasets). At the same time, RobustFtP$^{\text{D}}$ (yellow bar) and BlindOracle$^{\text{D}}$ (brown bar) prove to be the best, as they are able to leverage accurate predictions, while the remaining augmented algorithms are overly conservative. Predictors' scores for bzip and xalanc are not shown as they are above 2.0. Most augmented algorithms, however, still perform comparably to MARKER for these two datasets.

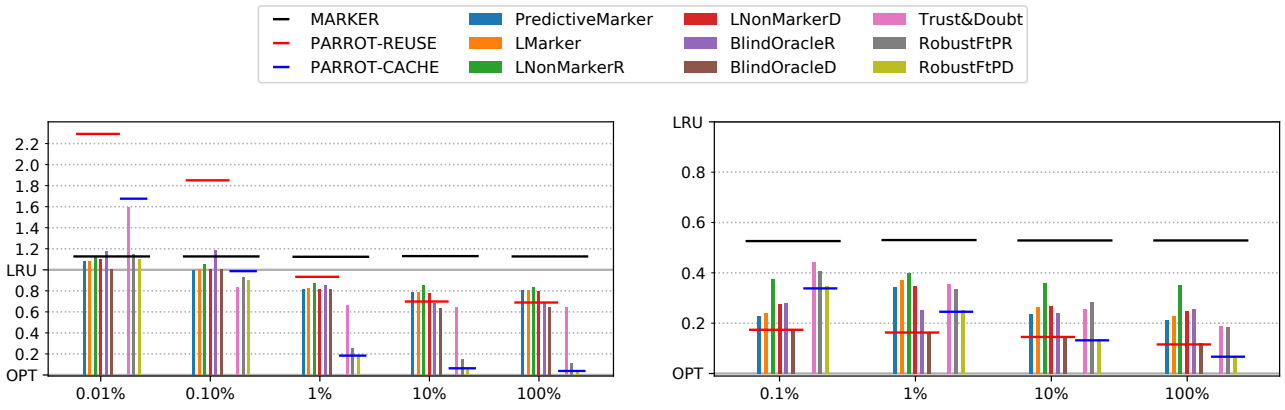

Figure 3. **Normalized cost (the lower the better). A closer look at mcf subsamples.** Performance of the PARROT-REUSE and PARROT-CACHE predictors improve along with the growing size of the trained dataset, while the two best augmented algorithms in the respective setups are able to follow the minimum of the results of the combined predictor and classical algorithms.

Figure 4. **Normalized cost (the lower the better). A closer look at sphinx3 subsamples.** The PARROT-REUSE is more robust to the scarcity of data when trained on subsamples of sphinx3. The predictors improve along with the growing size of the trained dataset. The augmented algorithms perform significantly better than MARKER, but only RobustFtP$^{\mathrm{D}}$ and BlindOracle$^{\mathrm{D}}$ approach the predictor.

methods with randomized combinations (i.e., BlindOracle$^{\mathrm{R}}$ and RobustFtP$^{\mathrm{R}}$) seem to overly rely on MARKER, and their performance is clearly worse.

All remaining methods offer only a slight improvement over MARKER, even if the predictor excels. These complex strategies on how to use predictions, developed to prove worst-case bounds, do not work in practice for the analyzed cases.

In a nutshell, PARROT-CACHE significantly outperforms all algorithms and RobustFtP$^{\mathrm{D}}$ uses it with only a small overhead. We will check how these methods work when coupled with underperforming predictors in the next subsection.

5.2. Underperforming predictor

As mentioned before, we impair the training procedure to obtain underperforming predictors. Specifically, we heavily subsample the training sets, leaving the first 1% of requests available to the models. The results are presented in Figure 2.

Interestingly, even when trained on only 1% of the data, PARROT-REUSE and PARROT-CACHE are still sometimes able to find policies better than MARKER. In these cases, the best learning-augmented caching algorithms perform better than MARKER, with a small overhead, similar to the results in the previous subsection.

In most cases, however, the predictors overfit to severely limited training data, and the resulting caching strategy is no better than MARKER. Both PARROT-CACHE and PARROT-REUSE lag behind the classical method for a few tasks (bzip, gems, leslie3d, omnetpp and xalanc). In these cases, the augmented algorithms – with the exception of BlindOracle$^{\mathrm{R}}$ and RobustFtP$^{\mathrm{R}}$ for two datasets – perform comparably with MARKER, which empirically proves their robustness. Notably, the robustness of the two simplest methods (BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$), shown in the previous subsection to be much better at leveraging good predictions, is comparable to the rest of the algorithms.

In short, together with conclusions from the previous subsection, the two methods closely track the better of their two components – the predictor or MARKER. The remaining methods add too much overhead when coupled with well-performing predictors, while it does not result in better robustness.

5.3. A closer look into specific datasets

To illustrate changes in the performance of the compared algorithms, we further subsample the largest dataset in our suite – mcf – in order to obtain series of training sets of varying sizes from 0.01% to 100% of the original size.

We train PARROT-REUSE and PARROT-CACHE on each of them, and then couple augmented algorithms. The results are present in Figure 3.

As expected, the performance of the PARROT-CACHE and PARROT-REUSE models improves along with the growth of the available data. We can observe that the performance of all the augmented algorithms continues to improve along with the performance of the coupled predictor. However, when the predictor underperforms MARKER, the performances of the augmented algorithms remain on par with the classical algorithm, with a notable exception of the

Trust&Doubt, which is clearly the worst.

However, as the PARROT-CACHE and PARROT-REUSE predictors start to outperform LRU and MARKER, the performance of the combining algorithms improves along with them. The most notable result here is how closely the RobustFtP$^{\mathrm{D}}$ follows improvements in the performance of PARROT-CACHE, while overhead for remaining algorithms remains significantly larger.

At the first glance it might be surprising that sometimes, e.g. mcf (1%), augmented algorithms outperform both MARKER and their coupled predictors. However, an augmented algorithm can take advantage of the fact that each of its ingredients may perform better on a different part of the dataset.

Another dataset that we investigate in more detail is sphinx3. As it is smaller than mcf, we can only train models on the range from 0.1% to 100%. Interestingly here on small subsamples (0.1% and 1%) the PARROT-REUSE performs better than PARROT-CACHE, suggesting it might be more robust to scarcity of training data. After careful examination the same trend can be observed comparing Figures 1 and 2.

Since even the weakest predictors easily outperform MARKER on this dataset, the learning-augmented algorithms perform similar to what we see already in Section 5.1.

5.4. Towards performance explanation

To better understand the differences in performance between the algorithms, we ran additional experiments (on a subset of datasets) in which we measured how much the algorithms followed the predictors. That notion is straightforward for BlindOracle and RobustFtP, as they, in each time step, either do exactly what the predictor advises, or follow MARKER, which in turn is completely independent from the predictor. However, the notion becomes more subtle for other algorithms, which, e.g., apply predictions only to a varying subset of items. To overcome that difficulty, we used a measure of *prediction usage*, which is algorithm independent. Specifically, for each algorithm we computed the Jaccard similarity between the caches maintained by the algorithm and the predictor (if followed blindly), averaged over time.

Our general conclusion is that algorithms' performance is correlated with how much they choose to follow the underlying predictor. The correlation is positive on datasets where the respective predictor performs better than MARKER, and negative otherwise, see Figure 5. Most algorithms correctly decide for each dataset whether it is on average better to follow the predictor or not. It is, however, the level of commitment to that decision that differs: With fully-fledged predictors, deterministic BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$ followed them >97% of the time, randomized BlindOracle$^{\mathrm{R}}$ and RobustFtP$^{\mathrm{R}}$ – around 90% of the time, and remaining algorithms – at most 85%, often much less. For most algo-

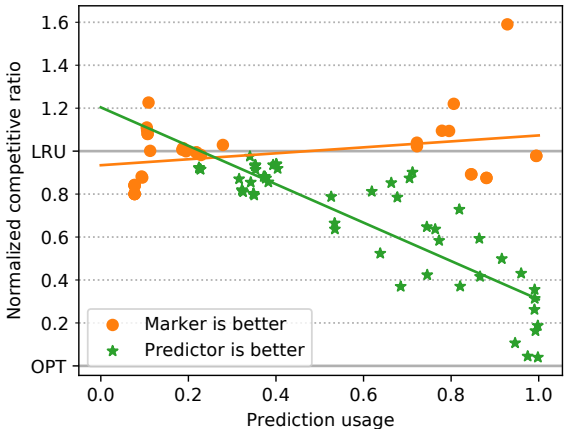

Figure 5. **Prediction usage**. Each data point represents a pair (dataset, learning-augmented algorithm). Orange dots correspond to pairs such that MARKER performed better on that dataset than the predictor used by that algorithm (if followed blindly). Green stars correspond to pairs where the predictor was better. See Section 5.4 for the definition of *prediction usage*. Intuitively, a good learning-augmented algorithm should have a large prediction usage for green stars and a small one for orange dots.

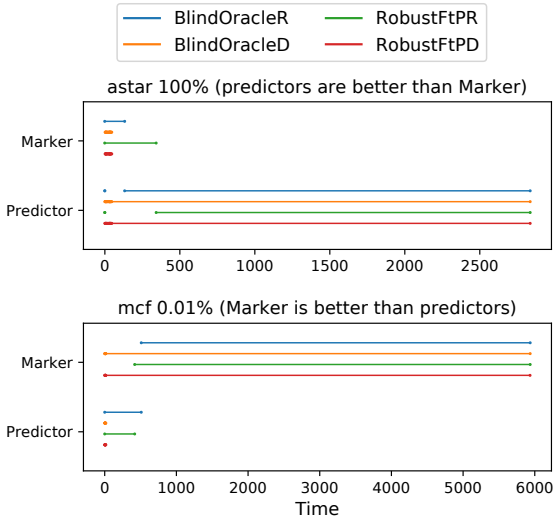

Figure 6. **Switching behavior of the combiners**. For each time step, we draw a point in the upper part of the subplot if the combiner follows MARKER, and in the lower part if it follows the predictor. For astar 100% dataset, the deterministic combiner, after a few switches, quickly infers that the predictor is better than MARKER and follows it until the end. On the other hand, the randomized combiner needs more time to make a decision. As a result, even though it follows the predictor most of the time, the initial hesitancy jeopardizes its overall performance. For mcf 0.01% dataset, where the predictors are outperformed by MARKER, the deterministic combiner again outperforms randomized variant, as it quickly identifies what to follow.

rithms, how much the algorithm follows the predictor seems to depend on the predictor's performance – the better the predictor, the more it is followed. Only for Trust&Doubt these two numbers seem uncorrelated, which may explain its lower robustness.

Next, we analyzed the switching behavior of the combiners, see Figure 6. The deterministic combiner, after few brief switches, quickly infers what to follow for each dataset. The randomized combiner also eventually follows the better of the two underlying algorithms, but needs much more time to figure out that it has to switch, which jeopardizes its overall performance. That explains why randomized BlindOracle$^{\mathrm{R}}$ and RobustFtP$^{\mathrm{R}}$ tend to perform worse than their deterministic counterparts, BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$.

6. Related Work

Learning-augmented algorithms. The idea of augmenting an algorithm with hints or predictions coming from a potentially untrusted oracle is not new. The recent variant, clearly inspired by the now omnipresent machine-learned predictors for various tasks, seems to spark from Lykouris & Vassilvitskii (2018) and Purohit et al. (2018). The idea has been applied to many problems, also beyond the online algorithms, e.g. to Bloom filters (Kraska et al., 2018). For an overview of the field, see the recent survey by Mitzenmacher & Vassilvitskii (2021).

Robust machine learning. Robustness of machine learning methods is sometimes defined as robustness to *adversarial examples* – approximately estimated worst-case inputs laying controllably far from training distribution (Carlini & Wagner, 2017; Weng et al., 2019; Szegedy et al., 2014). More broadly, however, robustness in machine learning can be seen as an ability to generalize, i.e., to perform well on unseen examples (Bishop, 2007), where a distribution shift between training and testing examples poses a challenge (Moreno-Torres et al., 2012; Geirhos et al., 2020).

We experiment with heterogeneous sequential datasets (Henning, 2006; CRC, 2017). Their characteristics change over time, as they are generated by real-world programs. We leverage this property to test generalization ability – and hence robustness – of state-of-the-art machine learning predictors (Liu et al., 2020). We vary amount of data available during training to analyze pessimistic cases and use learning-augmented algorithms to incorporate robustness to caching policies based on neural network predictions.

7. Conclusions

We fill a critical gap in the learning-augmented literature. We evaluated the learning-augmented caching algorithms using the state-of-the-art predictors on real-world datasets. In a nutshell, we conclude that learning-augmented algorithms can have only a low overhead over a well-performing predictor, while competing with classical methods when the coupled predictor fails, thus providing a cheap worst-case insurance.

Our experiments show that when the training data is scarce, the performance of the state-of-the-art Parrot model tends to degrade quickly, depending on the dataset. Hence, it justifies looking for a way to benefit from the robustness of the classical online algorithms. As learning augmented algorithms do exactly that, we test their performance in practice.

Two algorithms – BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$ – turn out to be the best. They provide a very low overhead over good predictions but still compete with the robust classical methods even when the predictors fail. The remaining four tested algorithms are robust, but they seem to be overly conservative and do not fully utilize good predictions.

We show that the theoretical asymptotic competitive ratio is not a good proxy for the practical performance of the learning-augmented algorithms. In the reuse distance setup, it correctly points to the leader but incorrectly distinguishes between the remaining algorithms. In the policy setup, the theoretically inferior algorithm turns out the best in practice. Moreover, according to the theoretical analysis, the randomized combiner should perform better than the deterministic one, while in our experiments we observe the opposite.

On most datasets, the predictor for the optimal policy setup outperforms the predictor for the reuse distance setup. Hence, learning-augmented algorithms in the latter setup can hardly compete with the RobustFtP$^{\mathrm{D}}$, designed for the former. That conclusion is valid with respect to current state-of-the-art predictors, and future improvements to reuse distance predictors may invalidate it. However, in the view of our results, a direct empirical comparison between the two alternative setups becomes less relevant. Indeed, theoretical developments in both setups independently led to the same algorithmic idea behind BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$. The idea excels in practice and can presumably be applied to any type of predictor. As the examples of BlindOracle$^{\mathrm{R}}$, RobustFtP$^{\mathrm{R}}$, and Trust&Doubt show, optimizing the methods further towards an objective specific to a particular setup does not necessarily lead to improved performance in practice.

Acknowledgements

Adam Polak was supported by the Swiss National Science Foundation (SNF) within the project *Lattice Algorithms and Integer Programming (185030)*. Konrad Żołna was supported by the National Science Center, Poland (2017/27/N/ST6/00828, 2018/28/T/ST6/00211).

References

The 2nd Cache Replacement Championship, 2017. URL https://crc2.ece.tamu.edu/.

Antoniadis, A., Coester, C., Eliás, M., Polak, A., and Simon, B. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, 2020.

Belady, L. A. A study of replacement algorithms for virtual-storage computer. *IBM Systems Journal*, 1966. doi: 10.1147/sj.52.0078.

Bishop, C. M. *Pattern recognition and machine learning, 5th Edition*. Springer, 2007.

Blum, A. and Burch, C. On-line learning and the metrical task system problem. *Machine Learning*, 2000. doi: 10.1023/A:1007621832648.

Bruening, D., Garnett, T., and Amarasinghe, S. An infrastructure for adaptive dynamic optimization. In *International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization*, 2003. doi: 10.1109/CGO.2003.1191551.

Carlini, N. and Wagner, D. A. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy*, 2017. doi: 10.1109/SP.2017.49.

Fiat, A., Karp, R. M., Luby, M., McGeoch, L. A., Sleator, D. D., and Young, N. E. Competitive paging algorithms. *Journal of Algorithms*, 1991. doi: 10.1016/0196-6774(91)90041-V.

Fiat, A., Rabani, Y., and Ravid, Y. Competitive k-server algorithms. *Journal of Computer and System Sciences*, 1994. doi: 10.1016/S0022-0000(05)80060-1.

Geirhos, R., Jacobsen, J.-H., Michaelis, C., Zemel, R., Brendel, W., Bethge, M., and Wichmann, F. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020. doi: 10.1038/s42256-020-00257-z.

Henning, J. L. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News*, 2006. doi: 10.1145/1186736.1186737.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural Computation*, 1997. doi: 10.1162/neco.1997.9.8.1735.

Jain, A. and Lin, C. Back to the future: Leveraging Belady's algorithm for improved cache replacement. In *IEEE/ACM Annual International Symposium on Computer Architecture*, 2016. doi: 10.1109/ISCA.2016.17.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. The case for learned index structures. In *SIGMOD International Conference on Management of Data*, 2018. doi: 10.1145/3183713.3196909.

Liu, E. Z., Hashemi, M., Swersky, K., Ranganathan, P., and Ahn, J. An imitation learning approach for cache replacement. In *International Conference on Machine Learning*, 2020.

Lykouris, T. and Vassilvitskii, S. Competitive caching with machine learned advice. In *International Conference on Machine Learning*, 2018.

Mitzenmacher, M. and Vassilvitskii, S. *Algorithms with Predictions*. Cambridge University Press, 2021. doi: 10.1017/9781108637435.037.

Moreno-Torres, J. G., Raeder, T., Alaíz-Rodríguez, R., Chawla, N. V., and Herrera, F. A unifying view on dataset shift in classification. *Pattern Recognition*, 2012. doi: 10.1016/j.patcog.2011.06.019.

Purohit, M., Svitkina, Z., and Kumar, R. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*, 2018.

Rohatgi, D. Near-optimal bounds for online caching with machine learned advice. In *ACM-SIAM Symposium on Discrete Algorithms*, 2020. doi: 10.1137/1.9781611975994.112.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

Seo, M. J., Kembhavi, A., Farhadi, A., and Hajishirzi, H. Bidirectional attention flow for machine comprehension. In *International Conference on Learning Representations*, 2017.

Shi, Z., Huang, X., Jain, A., and Lin, C. Applying deep learning to the cache replacement problem. In *IEEE/ACM International Symposium on Microarchitecture*, 2019. doi: 10.1145/3352460.3358319.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Wei, A. Better and simpler learning-augmented online caching. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 2020. doi: 10.4230/LIPIcs.APPROX/RANDOM.2020.60.

Weng, L., Chen, P.-Y., Nguyen, L., Squillante, M., Boopathy, A., Oseledets, I., and Daniel, L. PROVEN: Verifying robustness of neural networks with a probabilistic approach. In *International Conference on Machine Learning*, 2019.

Yan, G. and Li, J. RL-Bélády: A unified learning framework for content caching. In *ACM International Conference on Multimedia*, 2020. doi: 10.1145/3394171.3413524.