ROBUST LEARNING-AUGMENTED CACHING AN EXPERIMENTAL STUDY

Jakub Chłędowski[*,a] **Adam Polak**[*,b] **Bartosz Szabucki**[*,a] **Konrad Żołna**[*,a]

[a]Jagiellonian University, Kraków, Poland [b]EPFL, Lausanne, Switzerland
{jakub.chledowski,bartosz.szabucki,konrad.zolna}@gmail.com, adam.polak@epfl.ch

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 (ML) to decide what to evict by discovering implicit input patterns and predicting the future. While ML 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 ML predictors robust. We are the first to comprehensively evaluate these learning-augmented algorithms on real-world caching datasets and state-of-the-art ML 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 AND RELATED WORK

We study a key optimization problem arising in caching: *Which item to evict from the cache in order to make room for a new item?* The goal is to minimize the number of *cache misses*, i.e., situations when the requested item is no longer present in the cache and needs to be reloaded.[1]

In the *offline* scenario, i.e., when we know in advance the sequence of requested items, the problem can be solved optimally by always evicting the item which will reappear the furthest in the future (Belady, 1966). 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 $\mathcal{O}(\log k \cdot \mathrm{OPT})$ cache misses on inputs which the optimal offline algorithm serves with OPT misses. On the other hand, real-world applications usually employ simpler heuristics, e.g. the gold standard Least Recently Used (LRU). Recent machine learning (ML) 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, ML algorithms tend to work well on typical inputs but can perform arbitrarily badly when in the worst case. To address that issue, Lykouris & Vassilvitskii (2018) proposed coupling an ML *predictor* with a *learning-augmented algorithm*. In their setup for caching, after each request, the predictor has to forecast when the requested item will be requested again – the value called *reuse distance*. They define the prediction error η_{reuse} to be the total $L1$ distance between the actual and predicted reuse distances. Their learning-augmented algorithm, PredictiveMarker 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] Subsequent algorithms

[*]Equal contribution.

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

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

for this setup, LMarker and LNonMarker by Rohatgi (2020), and BlindOracle by Wei (2020), have improved dependencies on the prediction error η_{reuse}.

Antoniadis et al. (2020) proposed an alternative setup, differing in the type of predictions and error metric. 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: RobustFtP for general MTS, and Trust&Doubt specifically for caching, with a better dependence on prediction error.

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 *deterministic* combiner simulates the two algorithms, keeps track of their respective costs up to date, and switches 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). Wei (2020) and Antoniadis et al. (2020) use an idea of Blum & Burch (2000) to provide a more intricate *randomized* combiner, which in theory has a lower overhead.

LNonMarker, BlindOracle, RobustFtP are built using combiners. We will consider two variants of each of them, with the deterministic and the randomized combiner (denoted by D and R, respectively). The remaining three algorithms achieve robustness by different means. Notably, BlindOracle and RobustFtP are based on essentially the same simple algorithmic idea, i.e., they apply a combiner to (1) robust Marker and (2) consistent following the predictions (for its respective setup).

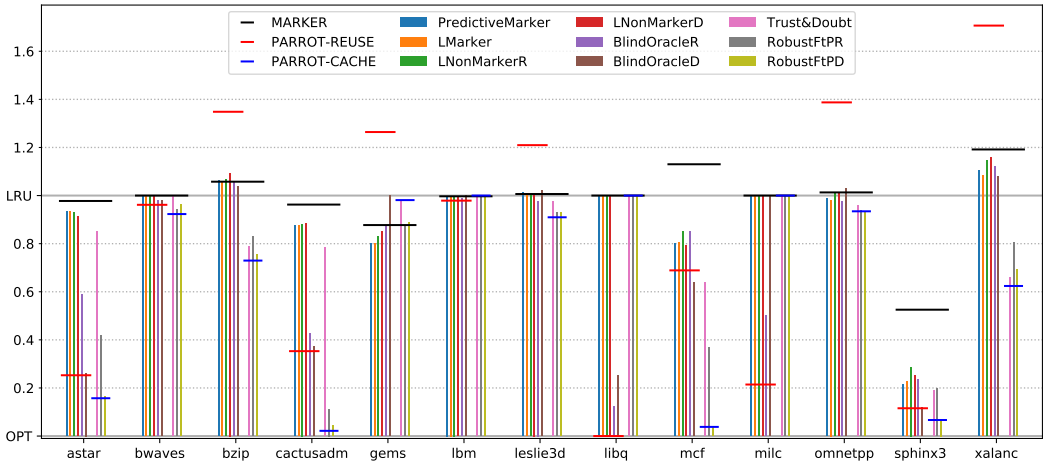
Motivation. A direct comparison of the above learning-augmented caching algorithms is problematic. Admittedly, worst-case competitive ratios of algorithms within the same setup can be compared, but such theoretical comparison between the two setups is implausible. The final competitive ratios depend on coupled predictors, with different error measures that cannot be translated between the setups. A priori, it is not clear which of the two types of predictors is easier to train well.

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 adhoc predictors instead of fully-fledged machine learning models. The following question remains wide open: ***Are learning-augmented caching algorithms practical?***

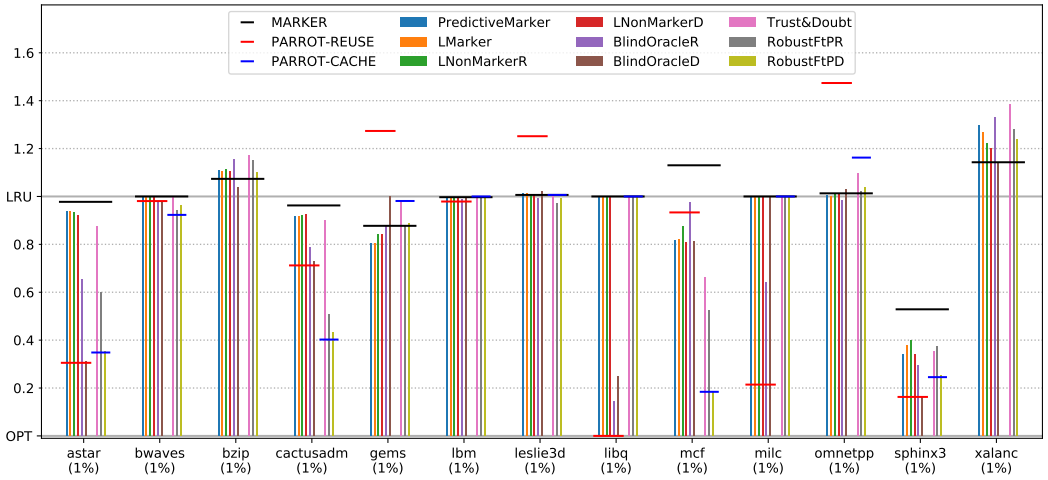
Our study. In this paper, we set out to answer this question experimentally. We compare all six above learning-augmented caching algorithm. We use benchmark dataset from the 2nd Cache Replacement Championship (CRC, 2017), also used by Shi et al. (2019) and Liu et al. (2020) (for details see Section B in Supplementary Material).

For predictions, we use state-of-the-art ML-based caching algorithm Parrot (Liu et al., 2020). It is an imitation learning algorithm that tries to mimic Belady's oracle policy with a deep neural network. The model has two prediction heads, 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. 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. For details of our experimental setup see Section C in Supplementary Material.

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 training data. That lets us evaluate if the theoretical robustness guarantees play a role in practice.



(a) Predictors trained on full datasets.



(b) Underperforming predictors trained on 1% of data.

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. For details see Section 2.

2 Experiments and Results

As already mentioned, 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. We couple each learning-augmented caching algorithm with one of the predictors, depending on the nature of the algorithm. Then, we impair the training procedure to obtain underperforming predictors, by heavily subsampling the training sets, leaving the first 1% of requests available to the models. The results are presented in Figures 1(a) and 1(b), for full and limited datasets, respectively.

To make comparison across all datasets easier, we normalize scores so that 1 indicates the performance of LRU and 0 corresponds to Belady's OPT. Therefore, the lower score, the better. See Section D in Supplementary Material for a detailed description of the normalization formula used.

State-of-the-art predictors. Figure 1(a) confirms results by Liu et al. (2020). PARROT-CACHE is almost always better than the classical LRU baseline and 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. The versions of these 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. 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. Next, we test these methods coupled with underperforming predictors.

Underperforming predictors. Interestingly, when trained on 1% of the data, PARROT-REUSE and PARROT-CACHE are still sometimes able to find policies better than MARKER (see Figure 1(b)). In these cases, the best learning-augmented caching algorithms also perform better than this baseline.

In most cases, however, the predictors overfit to severely limited training data. Both PARROT-CACHE and PARROT-REUSE lag behind the classical method for bzip, gems, leslie3d, omnetpp and xalanc. In these cases, all the augmented algorithms perform comparably with MARKER, which empirically proves their robustness.

In short, together with conclusions from the previous subsection, the two simplest methods (BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$) 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.

A closer look into mcf dataset. 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. The results are shown in Figure 2.

When a predictor underperforms MARKER, the performance of the coupled augmented algorithms (with exception of the Trust&Doubt) remain on par with the classical algorithm. As expected, the performance of both predictors improves along with the growth of the available data. The performance of all the augmented algorithms exhibit the same trend.

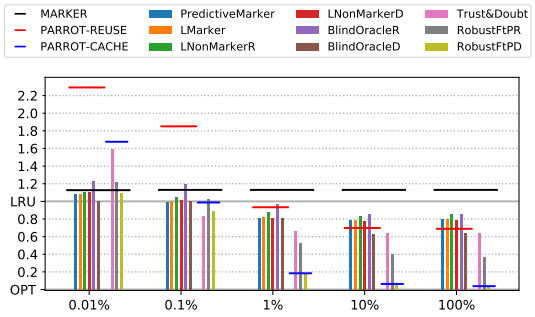


Figure 2: **A closer look at mcf subsamples.**

The most notable result here is how closely the RobustFtP$^{\mathrm{D}}$ follows improvements in the performance of PARROT-CACHE. Overheads for remaining algorithms remain 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.

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 the two simplest learning-augmented algorithms – BlindOracle$^{\mathrm{D}}$ and RobustFtP$^{\mathrm{D}}$ – 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 using learning-augmented algorithms to benefit from the robustness of the classical online algorithms.

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.

ACKNOWLEDGMENTS

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/`.

Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. Online metric algorithms with untrusted predictions. In *International Conference on Machine Learning*, 2020.

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

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

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

Derek Bruening, Timothy Garnett, and Saman Amarasinghe. 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.

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

Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 1991. doi: 10.1016/0196-6774(91)90041-V.

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

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2020. doi: 10.1038/s42256-020-00257-z.

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

Akanksha Jain and Calvin Lin. 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.

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *SIGMOD International Conference on Management of Data*, 2018. doi: 10.1145/3183713.3196909.

Evan Zheran Liu, Milad Hashemi, Kevin Swersky, Parthasarathy Ranganathan, and Junwhan Ahn. An imitation learning approach for cache replacement. In *International Conference on Machine Learning*, 2020.

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

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

Jose G. Moreno-Torres, Troy Raeder, Rocío Alaíz-Rodríguez, Nitesh V. Chawla, and Francisco Herrera. A unifying view on dataset shift in classification. *Pattern Recognition*, 2012. doi: 10.1016/j.patcog.2011.06.019.

Manish Purohit, Zoya Svitkina, and Ravi Kumar. Improving online algorithms via ML predictions. In *Advances in Neural Information Processing Systems*, 2018.

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

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

Zhan Shi, Xiangru Huang, Akanksha Jain, and Calvin Lin. Applying deep learning to the cache replacement problem. In *IEEE/ACM International Symposium on Microarchitecture*, 2019. doi: 10.1145/3352460.3358319.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations*, 2014.

Alexander Wei. 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.

Lily Weng, Pin-Yu Chen, Lam Nguyen, Mark Squillante, Akhilan Boopathy, Ivan Oseledets, and Luca Daniel. PROVEN: Verifying robustness of neural networks with a probabilistic approach. In *International Conference on Machine Learning*, 2019.

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

SUPPLEMENTARY MATERIAL

A SUMMARY OF ALGORITHMS

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	$\min(\log k, \sqrt{\eta_{\text{reuse}}/\text{OPT}})$	n/a	Lykouris & Vassilvitskii (2018)
LMarker	reuse	$\min(\log k, \log(\eta_{\text{reuse}}/\text{OPT}))$	n/a	Rohatgi (2020)
LNonMarker$^{\text{D}}$	reuse	$\min(\log k, {}^{\log k}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	det.	Rohatgi (2020)
LNonMarker$^{\text{R}}$	reuse	$\min(\log k, {}^{\log k}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	rand.	Rohatgi (2020)
BlindOracle$^{\text{D}}$	reuse	$\min(\log k, {}^{1}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	det.	Wei (2020)
BlindOracle$^{\text{R}}$	reuse	$\min(\log k, {}^{1}/_{k} \cdot \eta_{\text{reuse}}/\text{OPT})$	rand.	Wei (2020)
RobustFtP$^{\text{D}}$	cache	$\min(\log k, \eta_{\text{cache}}/\text{OPT})$	det.	Antoniadis et al. (2020)
RobustFtP$^{\text{R}}$	cache	$\min(\log k, \eta_{\text{cache}}/\text{OPT})$	rand.	Antoniadis et al. (2020)
Trust&Doubt	cache	$\min(\log k, \log(\eta_{\text{cache}}/\text{OPT}))$	n/a	Antoniadis et al. (2020)

B 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*[3] and filtering the accesses[4] to those sets. The first 80% of this sequence is used for training, followed 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 made the datasets publicly available[5].

C 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. Our code is publicly available at GitHub[6].

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 hyper-parameter was changed.

[3] 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 $k = 16$.)

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

[5] https://www.dropbox.com/sh/nnhe916fzg3cfwx/AACKWJFx2Q3gc-JveYktj3GQa?dl=0

[6] https://github.com/chledowski/ml_caching_with_guarantees

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%

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.

We ran the total of 53 trainings and evaluations. All but 2 runs described below were run using 20000 steps, with evaluation every 5000 steps.

For datasets from {astar, bwaves, bzip, cactusadm, gems, leslie3d, omnetpp, sphinx3, xalanc}, we ran 4 experiments on the first 0.1%, 1%, 10% and 100% records of the dataset.

For datasets from {lbm, libq, milc} we ran only two experiments, on 1% and 100% of the dataset. We have decided that running them will not give us any additional information, as the hit rate of PARROT-CACHE on both 1% and 100% were upper bounded by 0.01%.

For mcf, we ran 11 experiments in total. The first 5 were runs on 0.01%, 0.1%, 1%, 10% and 100% of the dataset, without dagger. For comparison, we have run additional six experiments with a larger number of steps and dagger. We have achieved comparable cache hit rates, therefore, did not evaluate with learning-augmented algorithms.

On average, each experiment took around one day on a Tesla V100 GPU, occupying at most 3GB of the GPU memory.

D RESULTS NORMALIZATION

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[7] 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.

E FULL RESULTS

We present the full cache hit rates of PARROT-REUSE across all evaluated datasets and fractions in table 3. The detailed non-normalized competitive ratios are shown in table 4.

[7]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.

Table 3: **Cache hit rates of PARROT-REUSE.** Detailed results.

DATASET	FRACTION	DAGGER	STEPS	EVAL FREQ	CACHE REQUESTS	CACHE REQUESTS IN TRAIN SET	HIT RATE CKPT 0 ON VAL	BEST NON 0 CHECKPOINT	HIT RATE CKPT BEST ON VAL	LAST CHECKPOINT	HIT RATE CKPT LAST ON VAL	HIT RATE CKPT BEST NON 0 ON TEST
ASTAR	0.1%	FALSE	20001	5000	640,032	1,154	8.25%	5000	13.32%	20000	11.83%	14.05%
ASTAR	1%	FALSE	20001	5000	640,032	11,540	8.25%	5000	25.09%	20000	24.55%	25.75%
ASTAR	10%	FALSE	20001	5000	640,032	115,404	8.25%	10000	30.26%	20000	30.05%	30.99%
ASTAR	100%	FALSE	20001	5000	640,032	1,154,048	8.25%	20000	31.53%	20000	31.53%	32.17%
BWAVES	0.1%	FALSE	20001	5000	640,032	570	0.02%	5000	0.03%	20000	0.02%	0.00%
BWAVES	1%	FALSE	20001	5000	640,032	5,703	0.02%	10000	1.93%	20000	1.62%	0.30%
BWAVES	10%	FALSE	20001	5000	640,032	57,036	0.02%	20000	0.03%	20000	0.03%	0.00%
BWAVES	100%	FALSE	20001	5000	640,032	570,368	0.02%	10000	3.21%	20000	1.21%	0.32%
BZIP	0.1%	FALSE	20001	5000	640,032	167	52.05%	20000	48.02%	20000	48.02%	53.23%
BZIP	1%	FALSE	20001	5000	640,032	1,676	52.05%	5000	42.69%	20000	41.94%	45.12%
BZIP	10%	FALSE	20001	5000	640,032	16,768	52.05%	5000	57.09%	20000	52.84%	64.28%
BZIP	100%	FALSE	20001	5000	640,032	167,680	52.05%	5000	63.50%	20000	61.29%	68.44%
CACTUSADM	0.1%	FALSE	20001	5000	640,032	221	0.32%	20000	4.81%	20000	4.81%	0.45%
CACTUSADM	1%	FALSE	20001	5000	640,032	2,219	0.32%	15000	8.43%	20000	8.42%	20.16%
CACTUSADM	10%	FALSE	20001	5000	640,032	22,195	0.32%	5000	25.88%	20000	24.17%	30.93%
CACTUSADM	100%	FALSE	20001	5000	640,032	221,952	0.32%	5000	25.30%	20000	24.98%	32.97%
GEMS	0.1%	FALSE	20001	5000	640,032	723	2.96%	15000	3.14%	20000	3.11%	3.09%
GEMS	1%	FALSE	20001	5000	640,032	7,234	2.96%	20000	3.12%	20000	3.12%	3.09%
GEMS	10%	FALSE	20001	5000	640,032	72,345	2.96%	20000	3.13%	20000	3.13%	3.08%
GEMS	100%	FALSE	20001	5000	640,032	723,456	2.96%	20000	1.77%	20000	1.77%	3.10%
LBM	1%	FALSE	20001	5000	640,032	7,820	0.02%	5000	0.19%	20000	0.13%	0.00%
LBM	100%	FALSE	20001	5000	640,032	782,080	0.02%	5000	1.22%	20000	0.29%	0.00%
LESLIE3D	0.1%	FALSE	20001	5000	640,032	716	0.81%	15000	1.29%	20000	1.01%	9.10%
LESLIE3D	1%	FALSE	20001	5000	640,032	7,160	0.81%	5000	1.62%	20000	1.21%	9.31%
LESLIE3D	10%	FALSE	20001	5000	640,032	71,603	0.81%	5000	2.09%	20000	1.00%	11.73%
LESLIE3D	100%	FALSE	20001	5000	640,032	716,032	0.81%	5000	4.99%	20000	4.89%	11.41%
LIBQ	1%	FALSE	20001	5000	640,032	5,798	0.00%	5000	0.00%	20000	0.00%	0.00%
LIBQ	100%	FALSE	20001	5000	640,032	579,840	0.00%	10000	0.01%	20000	0.00%	0.01%
MCF	0.01%	FALSE	20001	5000	640,032	296	2.61%	5000	15.79%	20000	11.02%	15.27%
MCF	0.01%	TRUE	20001	5000	640,032	296	2.61%	5000	15.79%	20000	11.02%	15.28%
MCF	0.1%	FALSE	20001	5000	640,032	2,965	2.61%	15000	22.19%	20000	22.16%	27.33%
MCF	0.1%	TRUE	20001	5000	640,032	2,965	2.61%	10000	21.64%	20000	21.40%	27.02%
MCF	1%	FALSE	20001	5000	640,032	29,655	2.61%	5000	40.00%	20000	39.44%	41.31%
MCF	1%	TRUE	20001	5000	640,032	29,655	2.61%	5000	40.35%	20000	39.93%	41.42%
MCF	10%	FALSE	20001	5000	640,032	296,550	2.61%	5000	42.09%	20000	41.49%	43.45%
MCF	100%	FALSE	20001	5000	640,032	2,965,504	2.61%	20000	43.05%	20000	43.05%	43.89%
MCF	100%	TRUE	20001	5000	640,032	2,965,504	2.61%	15000	42.65%	20000	42.49%	43.46%
MCF	100%	FALSE	120001	30000	3,840,032	2,965,504	2.61%	30000	43.10%	120000	43.10%	43.89%
MCF	100%	TRUE	120001	30000	3,840,032	2,965,504	2.61%	30000	42.60%	120000	42.60%	43.11%
MILC	1%	FALSE	20001	5000	640,032	5,568	0.22%	10000	0.23%	20000	0.23%	0.01%
MILC	100%	FALSE	20001	5000	640,032	556,800	0.22%	5000	0.34%	20000	0.23%	0.01%
OMNETPP	0.1%	FALSE	20001	5000	640,032	555	8.24%	5000	16.92%	20000	16.60%	17.90%
OMNETPP	1%	FALSE	20001	5000	640,032	5,555	8.24%	10000	13.85%	20000	13.62%	16.85%
OMNETPP	10%	FALSE	20001	5000	640,032	55,552	8.24%	5000	18.60%	20000	14.83%	19.28%
OMNETPP	100%	FALSE	20001	5000	640,032	555,520	8.24%	5000	22.23%	20000	15.46%	21.91%
SPHINX3	0.1%	FALSE	20001	5000	640,032	328	36.60%	20000	36.33%	20000	36.33%	53.78%
SPHINX3	1%	FALSE	20001	5000	640,032	3,287	36.60%	5000	43.84%	20000	42.71%	59.48%
SPHINX3	10%	FALSE	20001	5000	640,032	32,870	36.60%	5000	45.68%	20000	45.49%	66.55%
SPHINX3	100%	FALSE	20001	5000	640,032	328,704	36.60%	20000	55.18%	20000	55.18%	70.54%
XALANC	1%	FALSE	20001	5000	640,032	691	14.73%	15000	15.02%	20000	14.46%	32.38%
XALANC	10%	FALSE	20001	5000	640,032	6,912	14.73%	15000	17.49%	20000	16.91%	47.31%
XALANC	100%	FALSE	20001	5000	640,032	69,120	14.73%	5000	27.01%	20000	26.42%	49.71%

Table 4: **Competitive ratios.** Detailed results.

DATASET	FRACTION	OPT	LRU	MARKER	RANDOM	PARROT-REUSE	PREDICTIVEMARKER	LMARKER	LNONMARKERR	LNONMARKERD	BLINDORACLER	BLINDORACLED	PARROT-CACHE	TRUST&DOUBT	ROBUSTFTPR	ROBUSTFTPD
ASTAR	0.1%	1.00	1.53	1.52	1.47	1.26	1.51	1.51	1.51	1.51	1.46	1.27	1.37	1.53	1.48	1.38
ASTAR	1%	1.00	1.53	1.52	1.47	1.16	1.50	1.50	1.50	1.49	1.35	1.17	1.19	1.47	1.32	1.19
ASTAR	10%	1.00	1.53	1.52	1.47	1.15	1.50	1.50	1.50	1.49	1.33	1.15	1.10	1.45	1.23	1.11
ASTAR	100%	1.00	1.53	1.52	1.47	1.14	1.50	1.50	1.50	1.49	1.31	1.14	1.08	1.46	1.22	1.09
BWAVES	0.1%	1.00	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
BWAVES	1%	1.00	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
BWAVES	10%	1.00	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
BWAVES	100%	1.00	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05	1.05
BZIP	0.1%	1.00	1.88	1.94	2.26	2.96	2.00	1.98	1.97	1.99	2.03	1.93	2.43	1.99	2.01	1.99
BZIP	1%	1.00	1.88	1.95	2.24	2.92	1.98	1.98	1.98	1.98	2.02	1.92	2.85	2.03	2.02	1.97
BZIP	10%	1.00	1.88	1.95	2.24	2.39	1.93	1.92	1.94	1.96	1.96	1.93	1.86	1.81	1.87	1.85
BZIP	100%	1.00	1.88	1.94	2.26	2.19	1.94	1.93	1.94	1.97	1.94	1.92	1.65	1.70	1.73	1.67
CACTUSADM	0.1%	1.00	1.51	1.49	1.44	1.46	1.47	1.47	1.47	1.47	1.47	1.47	1.50	1.51	1.49	1.50
CACTUSADM	1%	1.00	1.51	1.49	1.44	1.36	1.46	1.46	1.47	1.47	1.40	1.37	1.20	1.46	1.26	1.22
CACTUSADM	10%	1.00	1.51	1.49	1.44	1.19	1.45	1.45	1.45	1.45	1.23	1.20	1.04	1.41	1.10	1.05
CACTUSADM	100%	1.00	1.51	1.49	1.44	1.18	1.44	1.44	1.45	1.45	1.22	1.19	1.01	1.40	1.06	1.02
GEMS	0.1%	1.00	1.11	1.09	1.09	1.14	1.09	1.09	1.09	1.09	1.09	1.11	1.10	1.10	1.09	1.09
GEMS	1%	1.00	1.11	1.09	1.09	1.14	1.09	1.09	1.09	1.09	1.09	1.11	1.10	1.10	1.09	1.09
GEMS	10%	1.00	1.11	1.09	1.10	1.14	1.09	1.09	1.09	1.09	1.09	1.11	1.10	1.10	1.09	1.10
GEMS	100%	1.00	1.11	1.09	1.09	1.13	1.09	1.09	1.09	1.09	1.09	1.11	1.10	1.10	1.09	1.09
LBM	1%	1.00	1.33	1.33	1.30	1.32	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33
LBM	100%	1.00	1.33	1.33	1.30	1.32	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33	1.33
LESLIE3D	0.1%	1.00	1.31	1.31	1.31	1.39	1.32	1.32	1.32	1.32	1.31	1.32	1.32	1.32	1.31	1.31
LESLIE3D	1%	1.00	1.31	1.31	1.31	1.39	1.31	1.31	1.31	1.31	1.31	1.32	1.31	1.31	1.30	1.31
LESLIE3D	10%	1.00	1.31	1.31	1.31	1.37	1.31	1.31	1.31	1.31	1.30	1.32	1.28	1.30	1.28	1.28
LESLIE3D	100%	1.00	1.31	1.31	1.31	1.38	1.31	1.31	1.31	1.31	1.30	1.32	1.28	1.30	1.29	1.29
LIBQ	1%	1.00	1.06	1.06	1.06	1.00	1.06	1.06	1.06	1.06	1.01	1.01	1.06	1.06	1.06	1.06
LIBQ	100%	1.00	1.06	1.06	1.06	1.00	1.06	1.06	1.06	1.06	1.01	1.01	1.06	1.06	1.06	1.06
MCF	0.01%	1.00	1.32	1.36	1.43	1.72	1.34	1.34	1.35	1.35	1.39	1.32	1.53	1.50	1.38	1.35
MCF	0.1%	1.00	1.32	1.36	1.43	1.58	1.31	1.32	1.33	1.32	1.38	1.32	1.31	1.26	1.32	1.28
MCF	1%	1.00	1.32	1.36	1.43	1.29	1.26	1.26	1.28	1.26	1.31	1.26	1.06	1.21	1.17	1.06
MCF	10%	1.00	1.32	1.36	1.43	1.22	1.25	1.25	1.26	1.25	1.27	1.20	1.02	1.20	1.13	1.02
MCF	100%	1.00	1.32	1.36	1.43	1.22	1.25	1.25	1.27	1.25	1.27	1.20	1.01	1.20	1.12	1.01
MILC	1%	1.00	1.01	1.01	1.01	1.00	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
MILC	100%	1.00	1.01	1.01	1.01	1.00	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01	1.01
OMNETPP	0.1%	1.00	1.38	1.39	1.43	1.57	1.38	1.38	1.38	1.39	1.38	1.39	1.43	1.39	1.38	1.39
OMNETPP	1%	1.00	1.38	1.39	1.43	1.56	1.38	1.38	1.39	1.39	1.38	1.39	1.44	1.42	1.39	1.40
OMNETPP	10%	1.00	1.38	1.39	1.43	1.55	1.38	1.38	1.38	1.38	1.37	1.39	1.40	1.40	1.38	1.39
OMNETPP	100%	1.00	1.38	1.39	1.43	1.53	1.38	1.38	1.39	1.39	1.37	1.39	1.36	1.37	1.36	1.36
SPHINX3	0.1%	1.00	3.46	2.29	1.88	1.43	1.56	1.59	1.77	1.68	1.72	1.44	1.83	2.09	2.11	1.86
SPHINX3	1%	1.00	3.46	2.30	1.89	1.40	1.84	1.93	1.98	1.84	1.72	1.41	1.60	1.87	1.92	1.62
SPHINX3	10%	1.00	3.46	2.30	1.88	1.36	1.58	1.65	1.76	1.67	1.66	1.37	1.32	1.63	1.64	1.33
SPHINX3	100%	1.00	3.46	2.29	1.87	1.28	1.52	1.56	1.71	1.62	1.58	1.29	1.16	1.47	1.49	1.17
XALANC	1%	1.00	1.27	1.30	1.46	1.87	1.35	1.34	1.33	1.32	1.35	1.30	1.56	1.37	1.34	1.33
XALANC	10%	1.00	1.27	1.32	1.47	1.48	1.28	1.29	1.30	1.29	1.31	1.29	1.22	1.22	1.24	1.24
XALANC	100%	1.00	1.27	1.32	1.47	1.45	1.29	1.29	1.31	1.31	1.30	1.29	1.17	1.18	1.21	1.18

F More on 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.