MemMAP: Compact and Generalizable Meta-LSTM Models for Memory Access Prediction

Ajitesh Srivastava[1(✉)], Ta-Yang Wang[1], Pengmiao Zhang[1], Cesar Augusto F. De Rose[2], Rajgopal Kannan[3], and Viktor K. Prasanna[1]

[1] University of Southern California, Los Angeles, CA 90089, USA
{ajiteshs,tayangwa,pengmiao,prasanna}@usc.edu
[2] Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
cesar.derose@pucrs.br
[3] US Army Research Lab-West, Playa Vista, USA
rajgopal.kannan.civ@mail.mil

Abstract. With the rise of Big Data, there has been a significant effort in increasing compute power through GPUs, TPUs, and heterogeneous architectures. As a result, many applications are memory bound, i.e., they are bottlenecked by the movement of data from main memory to compute units. One way to address this issue is through data prefetching, which relies on accurate prediction of memory accesses. While recent deep learning models have performed well on sequence prediction problems, they are far too heavy in terms of model size and inference latency to be practical for data prefetching. Here, we propose extremely compact LSTM models that can predict the next memory access with high accuracy. Prior LSTM based work on access prediction has used orders of magnitude more parameters and developed one model for each application (trace). While one (specialized) model per application can result in more accuracy, it is not a scalable approach. In contrast, our models can predict for a class of applications by trading off specialization at the cost of few retraining steps at runtime, for a more generalizable compact meta-model. Our experiments on 13 benchmark applications demonstrate that three compact meta-models can obtain accuracy close to specialized models using few batches of retraining for majority of the applications.

Keywords: LSTM · Compression · Meta-learning

1 Introduction

Prefetching is critical in reducing program execution time through hiding the latency due to data movement. Especially, with the advent of GPUs, TPUs, and heterogeneous architectures that accelerate computation, the bottleneck is

T.-Y. Wang and P. Zhang—Equal contribution.

H. W. Lauw et al. (Eds.): PAKDD 2020, LNAI 12085, pp. 57–68, 2020.
https://doi.org/10.1007/978-3-030-47436-2_5

shifting towards memory performance. The central aspect of prefetching is to be able to accurately predict future memory accesses. This can be seen as a sequence prediction task, which in theory, is well-suited for machine learning. Specifically, LSTM (Long-Short Term Memory) based Deep Learning has shown tremendous success in sequence prediction tasks like text prediction [4], along with other natural language tasks such as part of speech tagging [11] and grammar learning [14]. Since memory accesses have an underlying grammar similar to natural language, such models are naturally applicable to learning accesses. Recent work [5,13,15] has shown that LSTM based methods indeed lead to higher accuracy than those used in traditional prefetchers.

However, in reality, LSTM based prefetchers are far from becoming practical due to their extremely high memory and computation requirements. For instance, the models proposed in [5] can have more than a million parameters. Such a large number of parameters (and thus computations) make it infeasible to implement a prefetcher based on LSTM, as to be useful, these predictions need to be faster than accessing the sequence of memory addresses without any prefetching. Recent work [13] proposes an encoding method that reduces the size of the LSTM model to few thousands of parameters. They also show that such high compression can be achieved without any significant loss in accuracy. As a result, inference can be fast and models can be retrained quickly on demand, when there is a drastic change in access patterns. The drawback of this approach is that it requires training one model each for all applications. This is not a scalable solution as the number of applications grow, the total size of the models (storage required on the memory controller where these models will reside) grows linearly, thus defeating the purpose of having compact models. Further, such models do not apply to applications that have not been seen in training.

To address these shortcomings in making deep learning based prefetchers realistic, we develop a new approach - we show that using a small number of compact models (termed MemMAP) is sufficient to *adaptively* and *accurately* predict on a diverse set of applications of interest, i.e. these models can also generalize to applications not seen during training. Our approach relies on identifying clusters of applications that are similar, and then training a meta-model [3] for each cluster. MemMAP for high scalability (adaptability to multiple applications) and generalizability at the cost of small loss in accuracy and need for few retraining steps. Through extensive experiments on PARSEC [1] benchmark, which has diverse applications, we demonstrate that our approach leads to accurate, adaptable, and generalizable prediction access models. Using only three compact models of size 24K parameters each, we are able to perform on par with specialized models for 13 applications. We envision that in a real system implementation, the memory controller will run all three models concurrently, and use the model that produces better accuracy over last few accesses. Note that, in this paper, our objective is not to develop a full scale prefetcher, but to design a small set of highly accurate and compact LSTM based access prediction model to enable a realistic prefetcher implementation. A prefetcher built on

top of our approach and its hardware implementation will be explored in future work. Specifically, our contributions are as follows:

- We improve upon the state-of-the-art compressed LSTM models for access predictions, eliminating its necessity of one model per application (trace);
- We propose a clustered meta-learning-based approach to obtain more general prediction models that can achieve high accuracy after a small number of gradient steps and can even generalize to unseen/new applications;
- We experimentally demonstrate that our approach is accurate, adaptable, and generalizable – with a reduced number of models, we can achieve the same level of accuracy as the specialized (one model per application) approach with a much smaller memory footprint.

2 Related Work

Several prior works have proposed LSTM for memory access prediction [5,15]. In [12], the authors propose the use of logistic regression, and decision tree models to enhance prefetching. The authors in [7] evaluate various machine learning models on their ability to improve prefetching for data center applications. Neural networks and decision trees where shown to achieve the highest performance in this application domain. The work in [9,10], and [6] presents an extensive evaluation of LSTM for prefetching, achieving similar performance improvements as the other LSTM based approaches. Among the related work [5] has received significant attention. Their approach is impractical to be directly applied for prefetching, and as stated by the authors, is only a first step towards an LSTM-based prefetcher. They, and several state-of-the-art machine learning based access predictors perform the training on cache misses as it reduces the size of training. However, an accurate prefetcher will change the distribution of cache misses and hence invalidate its own trained model. Secondly, to achieve higher accuracy, some online training is necessary to learn application specific patterns. Their models are extremely large to be used for real-time inference or online retraining. Even after considering labels for predictions that cover 50% of the data (leading to a compulsory accuracy loss of 50%), the number of labels can be of the order of 10K. This, in turn, with a small hidden layer of size 100 will lead to a model with more than million parameters. Instead, we propose to use a small ensemble of highly compact LSTM models.

In [13] a compact LSTM based prediction model was proposed. Extremely high compression of LSTM model was achieved through encoding of the labels (jumps in memory accesses 'deltas'). The approach is based on the observation that the number of parameters are dominated by the output layer. Therefore, for label set of size n, they create the output layer with $\log n$ nodes each of which can take a 0 or 1 value. This network is trained to predict a multi-label output with $\log n$ labels, which is the binary representation of the delta instead of a single label (1 out of n) representing the delta itself. This technique led to around 1000× compression. On the other hand, in the process of compression, the prediction

problem is made harder due to the fact that all the $\log n$ bits need to be predicted correctly for the right memory access prediction. Yet, the experiments confirm that the loss in accuracy due to 1000× compression is negligible. While training one model for each application is possible and leads to highly specialized and accurate models [13], it is not a scalable solution. Further a specialized model does not generalize to other applications (see Fig. 1). In this work, we apply the same compression techniques presented in [13], but use meta LSTM models to avoid the need for one model per application. We also propose a clustered meta-learning-based approach to obtain more general prediction models that can achieve comparable accuracy as previous techniques after a small number of gradient steps and can even generalize to unseen/new applications. This results in a much smaller memory footprint compared to related work, allowing its implementation in hardware.

3 MemMAP Approach

We see the problem of access prediction as a sequence prediction problem, where the task is to predict the "delta", i.e., the jump in address with respect to the current address. This reduces the number of labels, i.e., possible outcomes for the predictions. Further, it accounts for the fact that often an application has similar jumps in addresses, even though it may start from a different memory location. Prior work [5,13] has taken the same approach of classifying deltas for the same reasons. Next we will explain the modeling of MemMAP.

3.1 Compression

For an LSTM model to be realistically used for prefetching, it needs to have low latency and should require small amount of computation. These factors are closely related to the size (number of parameters) of the model. As shown in [13], the size (number of parameters) of the simple LSTM model for memory access prediction is dominated by the dense last layer. Few thousands of output layers may lead to slowing down of inference due to large number of parameters in the final layer. Instead of using the deltas (jumps in memory accesses) directly as labels, the approach in [13] predicts the binary representation of deltas, converting the problem from a single label (1 out of n) prediction problem to a multi-label prediction problem ($\log n$ labels). Using this technique, we obtained an LSTM architecture which has $23,944$ parameters.

3.2 Meta-learning

The other dimension of reducing the overhead of memory access prediction is to reduce the number of models required for all the applications of interest. While training one model for each application leads to highly specialized and accurate models [13], it is not a scalable solution. Further a specialized model does not generalize to other applications. To demonstrate this, we trained specialized

models as in prior work [13], and tested them on other applications. Figure 1 shows one such instance, where the model was trained using the application "Swaption" and then tested on other applications of PARSEC benchmark. The results clearly indicate that the models are not generalizable.

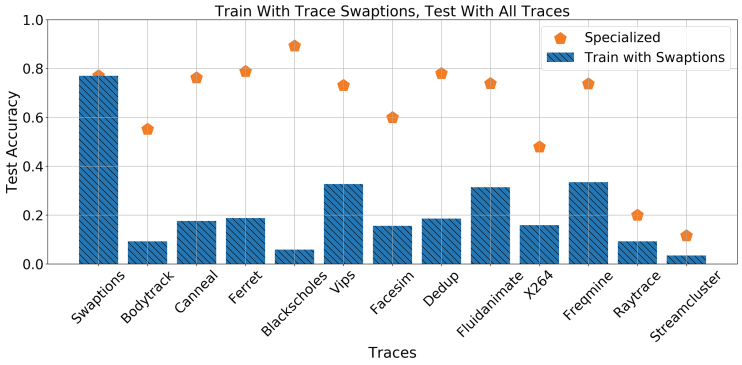

Fig. 1. Model obtained from one application do not generalize to other applications. The model was trained on the application 'swaption' and tested on all the applications in the PARSEC benchmark. The dots represent the accuracy achieved by training on the respective applications, provided as the reference accuracy.

Therefore, there is a need for creating a more general model that can work well for a class of applications, thus eliminating the size requirement of one model per applications and possibly generalizing to unseen applications. From the huge variations in accuracies seen in the plots, it is also clear that different patterns exist in different applications. This indicates that one model may not readily apply to all applications, and instead may requires some retraining. With the goal of obtaining a general model that quickly adapts to a chosen application, we use Model-Agnostic Meta-Learning [3] that samples batches from a set of applications to train one meta-LSTM model (Algorithm 1). First, we sample a set of applications and from each we prepare a batch of memory accesses. This batch is used to calculate loss and update adapted parameters from meta-parameters. Then from this mixed set of applications, a batch is prepared to compute the loss which is used to update the meta-model parameters. At termination, a meta-model is obtained which can adapt to all the tasks used in this training with few retraining steps.

3.3 Ensemble Meta-learning

While in the ideal scenario, we would like one meta-model to be enough, in reality, the application traces may vary drastically, making it difficult for one model to adapt to all the applications. Instead, we propose to use a small ensemble of meta-models that can cover all the applications. Our intuition is that it is better

Algorithm 1. Doubly Compressed LSTM with MAML

function MAML-DCLSTM(S)
  S: A set of applications
  Initialize θ and initial parameters α, β
  for $k \leftarrow 1$ to N_{epoch} **do**
    Sample batch of applications $A_i \sim S$
    for all A_i **do**
      Sample a batch D of m accesses from A_i
      Evaluate $\nabla_\theta L_{A_i}(f_\theta)$ using D, where L_{A_i} is the binary cross-entropy loss
      Compute the adapted parameters: $\theta_i' = \theta - \alpha \nabla_\theta L_{A_i}(f_\theta)$
      Sample accesses D_i' from A_i for the meta-update
    Update $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{A_i \sim S} L_{A_i}(f_{\theta_i'})$ using each D_i' and L_{A_i}
    return θ

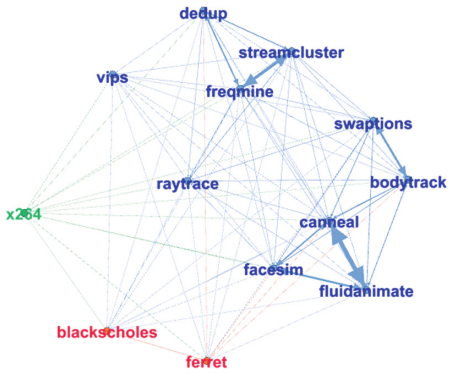

Fig. 2. Clusters obtained from PARSEC benchmarks.

to have similar applications for one meta-model, and so we train one meta-model for each set of similar applications. We construct the similarity matrix of the given set of application traces using soft-DTW [2] and then apply k-means to cluster the memory accesses. Soft-DTW is a differentiable approximation of DTW (Dynamic Time Warping). A smoothing parameter γ is introduced to the original min operation in DTW to create a generalized min operator. It can acquire better minima due to its better convexity properties in processing time-series data. As a pre-processing step, we convert the memory accesses into decimal values. Then they are standardized through subtracting the mean and dividing by the standard deviation. These standardized trace chunks are fed into a k-means clustering algorithm that uses soft-DTW to calculate the distance. The parameter k (number of clusters, i.e., number of meta-models) of k-means is chosen based on the memory available for storing the access prediction models. For our experiments, we have chosen $k = 3$ (see Sect. 4.2).

We consider the meta-model obtained for each cluster as a representative of a class of applications. In real implementation, all k (one for each of the k clusters) meta-models will work in parallel to predict the memory accesses, and

Algorithm 2. Doubly Compressed LSTM with cluster based MAML

```
function C-MAML-DCLSTM(S)
    Clustering applications in S into a collection of sets {S_i}_{i=1}^k
    for i ← 1 to k do
        θ_i ← MAML-DCLSTM(S_i)
    return {θ_i}
```

as more of the memory trace is seen, with few retraining steps, we will be able to identify which of the k models is more accurate. That model will be chose to continue inference, until the accuracy drops below a desired level. In that scenario, parallel retraining for all k meta models will resume. We believe that such retraining and switching between meta-models is essential as the program may go through a drastic change in access pattern. Similar concept of online retraining has been considered in [13].

4 Experiments

4.1 Datasets

We conducted extensive experimentation on the PARSEC benchmark [1], which was specifically chosen because of its diverse set of applications. The Intel Pin [8] tool was used to obtain memory access traces for each application. As mentioned earlier, instead of actual memory locations, we transform the memory traces to sequences of deltas by subtracting consecutive hexadecimal memory address and converting them to integer. The reason for this is to allow the model to predict memory locations for any future execution of the same application, since the relative memory differences are expected to stay consistent [5,13]

4.2 Model Settings

We used the doubly compressed LSTM (DCLSTM) architecture as described in [13]. It has an embedding layer with 10 units, followed by an LSTM layer with 50 units, followed by a dense layer with 50 units, and 15 outputs to represent up to 2^{15} most frequent deltas. We also used a dropout of 10%, look back window 3 (i.e., takes last three access predictions as input), 20 training epochs, a batch size 256, and 50-50 train/test split. We used sigmoid activation function and binary cross entropy loss function. This architecture is trained differently by different models as described below[1].

– Specialized: This is the DCLSTM model trained for one application. Ideally, this would be the best performing model, but it cannot be generalized. We will use the accuracies obtained from the specialized model as reference to compare other models on the given applications that are trained to adapt to multiple applications.

[1] The code is available at: https://github.com/MemMAP/MemMAP.

– Concatenated: This DCLSTM model is trained by simply concatenating the training traces from all applications.
– MAML-DCLSTM: This is a meta-model where the weights are learned using Algorithm 1.
– C-MAML-DCLSTM: This is a meta-model obtained from Algorithm 2. Instead of training with all the applications, this is trained with applications that belong to the same cluster. Three such models were trained based on the three clusters obtained from PARSEC (see Fig. 2).

4.3 Results

The goal of our experiments is to show that our cluster-based compact meta-LSTM models are: (a) Accurate – produce accuracy comparable to specialized models; (b) Adaptable – quickly adapt, i.e., specialize themselves for the given application; and (c) Generalizable – adapt to high accuracy even when the application was never seen before. The following results discuss these aspects.

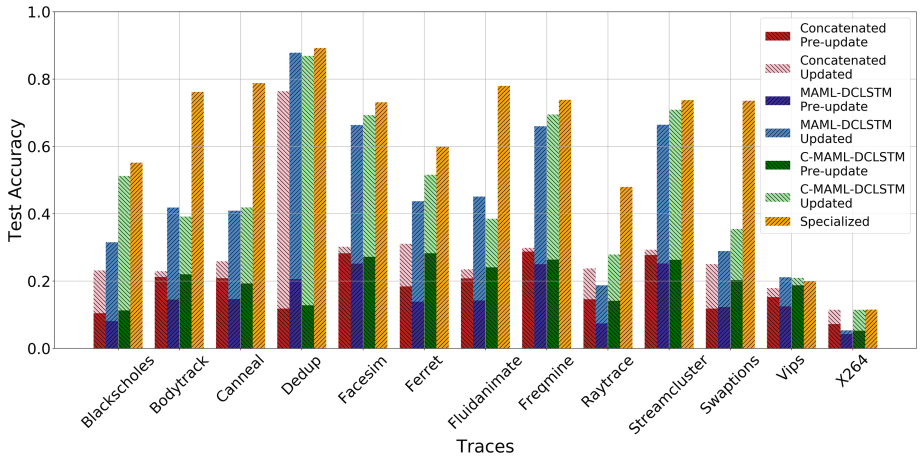

Fig. 3. Accuracy of the models pre- and post-retraining.

Figure 3 shows the accuracy results of all the methods. The specialized model serves as a reference for the ideal accuracy we wish to achieve. For concatenated model, MAML-DCLSTM and C-MAML-DCLSTM, we compared the model performance before retraining (pre-update) and after retraining (updated) by specific trace. In the experiment for pre-update models, we use 200K accesses for training and the next 200K for testing. For retraining, we use unseen 200K accesses of specific trace to retrain the existing pre-update models to get updated models for each trace. Then, we test them with the next 200K accesses in the trace. As shown in Fig. 3, the accuracies of all pre-update models are improved after retraining. In most cases (11 out of 13), MAML-DCLSTM models achieve higher accuracy than concatenated models, even when they start with lower

pre-update accuracy. This shows that the meta-model learns fast with a more general initialization. C-MAML-DCLSTM models gain a similar level of raises as MAML-DCLSTM. Due to the higher similarity of traces in the same cluster, C-MAML-DCLSTM models usually have higher pre-update accuracy. As a result, in 9 traces, C-MAML-DCLSTM outperform MAML-DCLSTM and in 3 traces they perform similarly. Overall, C-MAML-DCLSTM results in accuracies close to the specialized models in 9 out of 13 traces.

Figure 4 shows how retraining starting from various models improves the accuracy as more of the trace is seen. We compared the performance of concatenated, MAML-DCLSTM, C-MAML-DCLSTM, and specialized models by testing on two applications: Raytrace and Streamcluster. Note that, specialized models are used for reference, and we do not performing any retraining for them. We used 256 memory accesses for a batch of training and calculated test accuracy on the next 10K samples in rolling windows. Retraining is performed beginning from the weights of the neural network from the previous training batch. Based on the plots, although both MAML and concatenated models have similar result on some applications, the accuracy per batch on other traces such as Blackscholes, Ferret, and Streamcluser indicate that MAML-DCLSTM model learns faster than concatenated model, and C-MAML-DCLSTM performs better than MAML-DCLSTM. One can see that the relationship between these three models is clear for stable applications, while the others fluctuate a little. It seems that both C-MAML-DCLSTM and MAML-DCLSTM model can adapt to the stable applications rapidly and C-MAML-DCLSTM has the best adaptability. There are some challenging traces such as Vips and X264 on which even specialized model failed to achieve high accuracy. It is possible that the memory accesses of these applications vary considerably, and so prediction is extremely hard. In four out of 13 applications, the accuracy of C-MAML-DCLSTM is significantly less than specialized model. Improved clustering and more meta-models may be necessary for improving on these traces.

Figure 5 shows the comparison of how generalizable the models are. We split the applications in the same cluster into the training (Bodytrack, Canneal, Dedup, Facesim, Fluidanimate, Freqmine, Swaptions, Vips) and test sets (Raytrace and Streamcluster), the training set was used to build the meta-model using C-DCLSTM-MAML and concatenated model, and then we tested on the test applications to compare the performance of these two models for generalizability. We collected batches of 256 memory accesses for training and calculated test accuracy on next 10K samples in rolling windows. We performed retraining starting from the weights of the neural network from the previous training batch. The performance of C-MAML-DCLSTM improved after several memory accesses for both Raytrace and Streamcluster, which demonstrates that it can quickly generalize to unseen applications in the same cluster. Although the test accuracy for concatenated model did increase after several memory accesses in the case of Raytrace, it performed poorly in the case of Streamcluser. Furthermore, the C-MAML-DCLSTM model can obtain accuracy close to specialized models using only a small number of batches.

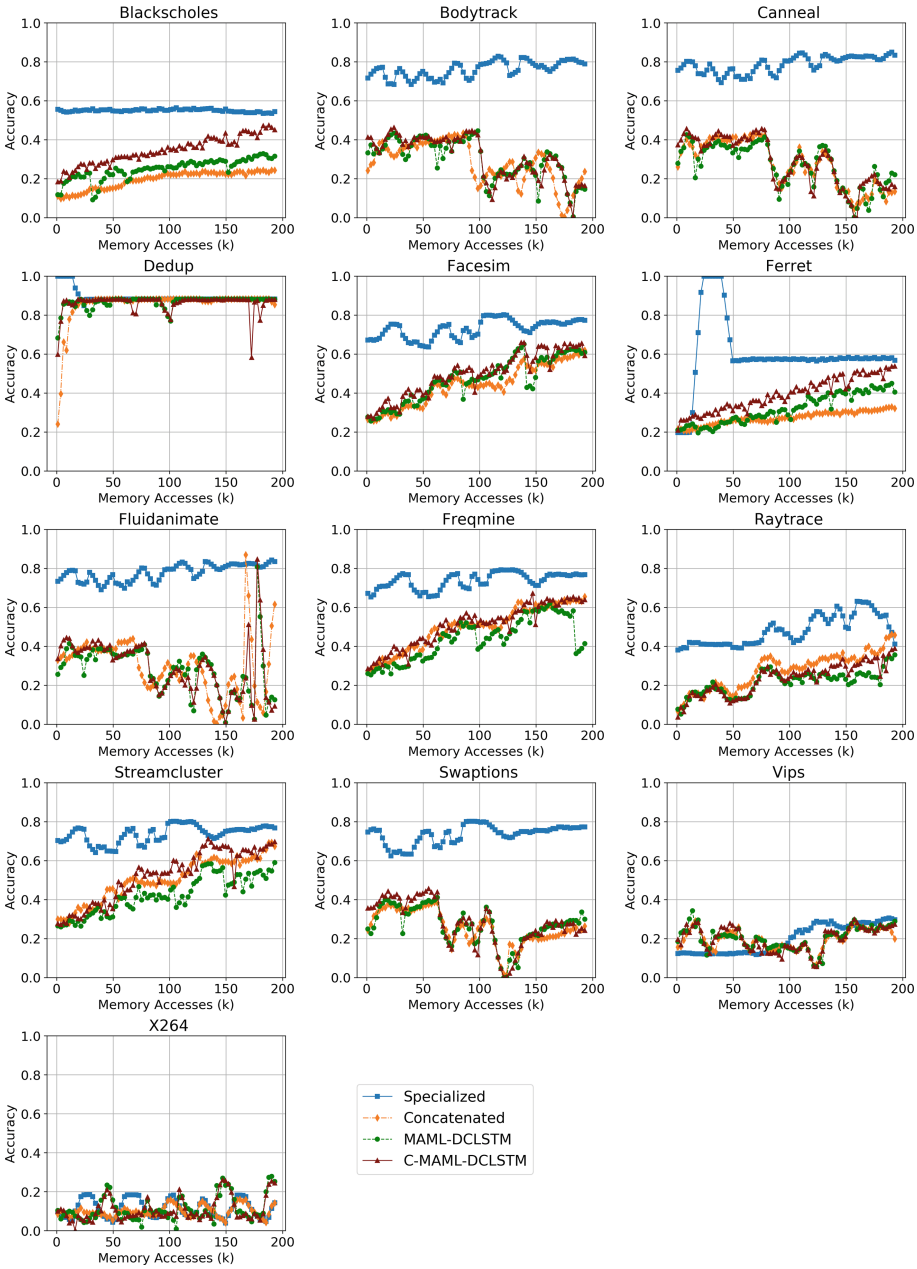

Fig. 4. Adaptability results for our meta-models.

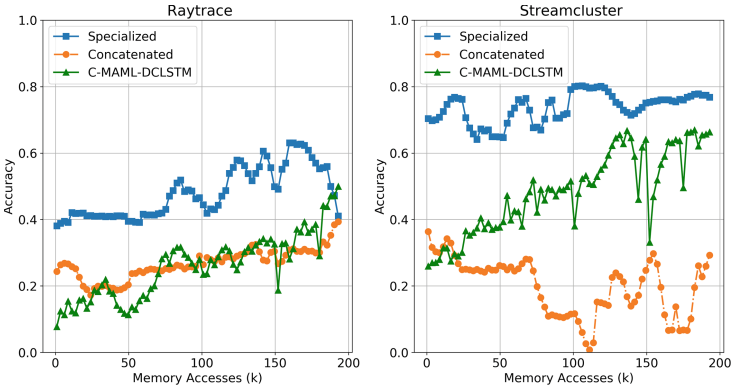

Fig. 5. Generalizability results for our meta-models.

5 Conclusions

We have proposed MemMAP a meta-model approach to predicting memory accesses, a central aspect of prefetchers, necessary to improve memory performance. We addressed the impracticality of current deep learning models in prefetching due to their high storage requirement. We improved upon the state-of-the-art, which although does provide compact LSTM models, it requires one model for each application. Such an approach does not scale to large number of applications. It also does not generalize to applications not seen before. We propose to use a clustering based meta-learning approach, where the applications are first clustered and then a meta-model is trained for each cluster. While, it is possible to train one model for all applications, the accuracy was typically lower. Our approach exploits the trade-offs between total model size, accuracy, and retraining steps. We showed that three models (with $3 \times 24K$ parameters) can achieve high accuracy quickly for 13 diverse applications. We show that our approach is accurate for majority of applications in the benchmarks, it adapts quickly with retraining of only one epoch with increasing number of accesses, and it can generalize to applications that were not seen during training. In future work, we will explore other clustering approaches to identify the ideal set of meta-models, and their hardware implementation.

Acknowledgements. This work is supported by Google Faculty Research Award, Air Force Research Laboratory grant number FA8750-18-S-7001, and National Science Foundation award number 1912680. The authors would like to thank Dr. Angelos Lazaris for useful discussions.

References

1. Bienia, C., Kumar, S., Singh, J.P., Li, K.: The parsec benchmark suite: characterization and architectural implications. In: Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, PACT 2008, pp. 72–81. ACM, New York (2008)
2. Cuturi, M., Blondel, M.: Soft-DTW: a differentiable loss function for time-series. In: Proceedings of the 34th International Conference on Machine Learning, ICML 2017, vol. 70, pp. 894–903. JMLR.org (2017)
3. Finn, C., Abbeel, P., Levine, S.: Model-agnostic meta-learning for fast adaptation of deep networks. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 1126–1135. JMLR.org (2017)
4. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: continual prediction with LSTM (1999)
5. Hashemi, M., et al.: Learning memory access patterns, March 2018
6. Hashemi, M., et al.: Learning memory access patterns. CoRR, abs/1803.02329 (2018)
7. Liao, S., Hung, T., Nguyen, D., Chou, C., Tu, C., Zhou, H.: Machine learning-based prefetch optimization for data center applications. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, pp. 1–10, November 2009
8. Luk, C.-K., et al.: Building customized program analysis tools with dynamic instrumentation. SIGPLAN Not. **40**(6), 190–200 (2005)
9. Narayanan, A., Verma, S., Ramadan, E., Babaie, P., Zhang, Z.L.: DeepCache: a deep learning based framework for content caching, pp. 48–53, August 2018
10. Peled, L., Weiser, U., Etsion, Y.: A neural network memory prefetcher using semantic locality, March 2018
11. Plank, B., Søgaard, A., Goldberg, Y.: Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. arXiv preprint arXiv:1604.05529 (2016)
12. Rahman, S., Burtscher, M., Zong, Z., Qasem, A.: Maximizing hardware prefetch effectiveness with machine learning. In: 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems, pp. 383–389, August 2015
13. Srivastava, A., Lazaris, A., Brooks, B., Kannan, R., Prasanna, V.K.: Predicting memory accesses: the road to compact ML-driven prefetcher. In: Proceedings of the International Symposium on Memory Systems, pp. 461–470. ACM (2019)
14. Vinyals, O., Kaiser, Ł., Koo, T., Petrov, S., Sutskever, I., Hinton, G.: Grammar as a foreign language. In: Advances in Neural Information Processing Systems, pp. 2773–2781 (2015)
15. Zeng, Y., Guo, X.: Long short term memory based hardware prefetcher: a case study. In: Proceedings of the International Symposium on Memory Systems, pp. 305–311. ACM, October 2017