International Journal of Data Science and Analytics
https://doi.org/10.1007/s41060-021-00268-y

REGULAR PAPER

C-MemMAP: clustering-driven compact, adaptable, and generalizable meta-LSTM models for memory access prediction

Pengmiao Zhang[1] · Ajitesh Srivastava[1] · Ta-Yang Wang[1] · Cesar A. F. De Rose[2] · Rajgopal Kannan[3] · Viktor K. Prasanna[1]

Received: 16 December 2020 / Accepted: 1 June 2021

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 clustering-driven compact LSTM models that can predict the next memory access with high accuracy. We introduce a novel clustering approach called Delegated model that can reliably cluster the applications. For each cluster, we train a compact meta-LSTM model that can quickly adapt to any application in the cluster. 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 clustering-driven ensemble compact meta-models can obtain accuracy close to specialized models using few batches of retraining for majority of the applications.

Keywords LSTM · Compression · Clustering · 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 shifting towards memory performance. The central aspect of prefetching is to be able to accurately predict future memory accesses. Memory access prediction problem is typically modeled as a classification problem, targeted to predict the future memory access addresses from program history information such as memory accesses and program counters (PC). Due to the extremely large value space of absolute memory addresses, deltas, defined as the address difference between two consecutive memory accesses, are more commonly used for both hardware prefetchers [16] and ML-based memory access predictors [21]. Therefore, the memory access prediction task can be formulated as predicting the next delta given a sequence of history deltas. This can be seen as a sequence prediction task, which in theory, is well suited for machine learning. Specifically, LSTM (long short-term

This paper is an extended version of the long paper which appeared in PAKDD'2020—"MemMAP: Compact and Generalizable Meta-LSTM Models for Memory Access Prediction" [18].

✉ Pengmiao Zhang
pengmiao@usc.edu

Ajitesh Srivastava
ajiteshs@usc.edu

Ta-Yang Wang
tayangwa@usc.edu

Cesar A. F. De Rose
cesar.derose@pucrs.br

Rajgopal Kannan
rajgopal.kannan.civ@mail.mil

Viktor K. Prasanna
prasanna@usc.edu

1 University of Southern California, Los Angeles, CA 90089, USA

2 Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil

3 US Army Research Lab, Los Angeles, USA

memory)-based deep learning has shown tremendous success in sequence prediction tasks like text prediction [6], along with other natural language tasks such as part of speech tagging [14] and grammar learning [19]. Since memory accesses have an underlying grammar similar to natural language, such models are naturally applicable to learning accesses. Recent work [7,17,20] has shown that LSTM-based methods indeed lead to higher accuracy than those used in traditional prefetchers. For instance, [21] trains one LSTM model for each application that learns the pattern of past three address deltas and predict the coming delta. The predicted delta can be converted back to memory address and can serve as a temporal reference for a hardware prefetcher.

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 [7] 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 [17] 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 proposed using a small number of compact meta-models to predict the memory access for a class of applications in our prior work [18]. We showed that the meta-models are 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 [18]. However, little attention is given to obtaining good clustering. In this work, we propose C-MemMAP—clustering-driven meta-LSTM models for memory access prediction. It uses a novel clustering approach termed Delegated model (DM) clustering. This approach uses a trained parametric model, compact LSTM in this work, as a delegate of the original sequence for further clustering task. This ensures that similarity between traces is defined by similarity in next access prediction, as the LSTM models can themselves be assumed to be a representation for trace patterns. Our approach makes the ML-based memory access prediction more practical and scalable for a prefetcher by reducing both the number and size of the models. First, the number of models does not grow when the number of applications increases. Instead of training one specialized model for each application, we train a small number of meta-models based on the application clustering. Second, we implement doubly compressed LSTM (DCLSTM) [17] in our meta-model to predict memory access. DCLSTM reduces the number of model parameters by predicting the binary representation of the output values, which achieves a $n/\log n$ ratio of compression with negligible accuracy compromise. As a result, C-MemMAP with compact model size achieves high adaptability and generalizability at the cost of a 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. The overall prefetcher architecture utilizing our model is described in [21]. 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 clustering-driven 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 improve our clustering technique from previous work with a novel Delegated model approach to detect applications with similar patterns that works reliable as an upstream of meta-models;
- 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 [7,20]. In [15], the authors propose the use of logistic regression, and decision tree models to enhance prefetching. The authors in [10] evaluate various machine learning models on their ability to improve prefetching for data center applications. Neural networks and decision trees were shown to achieve the highest performance in this application domain. The work in [13], [12], and [8] presents an extensive evaluation of LSTM for prefetching, achieving similar performance improvements as the other LSTM-based approaches. Among the related work, [7] 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 [17], 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 is 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 $1000\times$ 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 $1000\times$ compression is negligible. While training one model for each application is possible and leads to highly specialized and accurate models [17], it is not a scalable solution. Further, a specialized model does not generalize to other applications (see Fig. 2). In this work, we apply the same compression techniques presented in [17], 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.

In [18], the meta-LSTM approach for memory access prediction has been proposed. While in [18] training and testing sequences for each application are from the same trace, a model cannot train and test on the same trace in a practical setting. A practical approach would learn the patterns through profiling by running the application and then using the learned patterns in the future reruns to accelerate the program by prefetching. Also, clustering approaches as well as their influence in supporting meta-models are not fully explored. In this work, we extend the meta-model working scope to the rerun of PARSEC applications. Further, we propose a Delegated model clustering approach that can learn the latent patterns from the whole trace. We compared the model performance under different clustering approaches. Results show that Delegated model clustering is more reliable when application configurations change.

3 C-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 [7,17] has taken the same approach of classifying deltas for the same reasons. Next, we will explain the modeling of C-MemMAP.

Figure 1 illustrates the overall framework of the proposed C-MemMAP approach. The key component that is responsible for memory access prediction is DCLSTM (doubly compressed LSTM) models that input the memory access delta sequence D_i from Application A_i and output the next predicted delta Δ_i. To compress the model size, we applied meta-learning technique on DC-LSTM so that one model can adapt to more than one application. To maintain the prediction accuracy, we design a clustering step so that each meta-model handles applications with a similar pattern. We propose delegated model clustering algorithm that is trained using weights from offline-trained specialized models. In this way, we largely reduce the model size with a small loss in accuracy. We will introduce all the components of the C-MemMAP framework in detail in the following subsections.

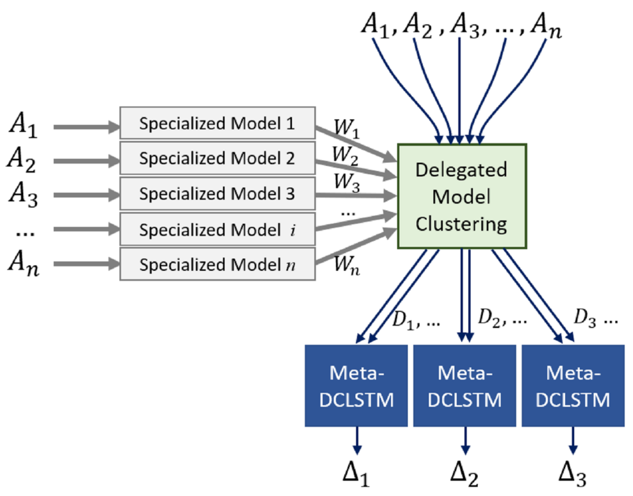


Fig. 1 Overall framework of clustering-driven meta-LSTM models for memory access prediction

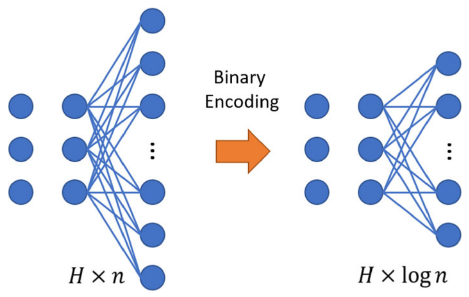


Fig. 3 Using binary encoding to highly reduce the dimension of the output dense layer in a classification model

3.1 Binary encoding compression

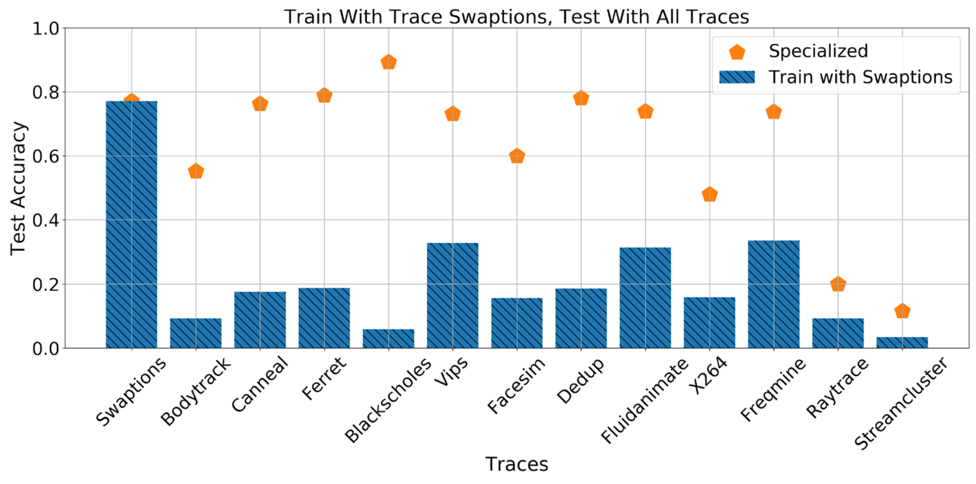
For an LSTM model to be realistically used for prefetching, it needs to have low latency and should require small amount of computation. These factors are closely related to the size (number of parameters) of the model. The size (number of parameters) of the simple LSTM model for memory access prediction is dominated by the dense last layer. Few thousands of output layers may lead to slowing down of inference due to a large number of parameters in the final layer. [17] proposed a model compressing approach that uses binary encoding to highly reduce the dimension of the output dense layer in a classification model, as is shown in Fig. 3. Applying this idea to the memory access prediction task, the authors in [17] proposed doubly compressed LSTM (DCLSTM) that uses a binary representation of deltas for both the input side and the output side, instead of using the deltas (jumps in memory accesses) directly. This approach converts 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 that 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 [17], 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 [17], and tested them on other applications. Figure 2 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.

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.

Fig. 2 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



Algorithm 1 Doubly Compressed LSTM with MAML

```
1: function MAML- DCLSTM($S$)
2:     $S$: A set of applications
3:     Initialize $\theta$ and initial parameters $\alpha, \beta$
4:     for $k \leftarrow 1$ to $N_{\text{epoch}}$ do
5:         Sample batch of applications $A_i \sim S$
6:         for all $A_i$ do
7:             Sample a batch $D$ of $m$ accesses from $A_i$
8:             Evaluate $\nabla_\theta L_{A_i}(f_\theta)$ using $D$, where $L_{A_i}$ is the binary cross-entropy loss
9:             Compute the adapted parameters: $\theta'_i \leftarrow \theta - \alpha \nabla_\theta L_{A_i}(f_\theta)$
10:            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 $L_{A_i}$
11:        return $\theta$
```

This indicates that one model may not readily apply to all applications, and instead may require some retraining. With the goal of obtaining a general model that quickly adapts to a chosen application, we apply model-agnostic meta-learning (MAML) [5] to train a meta model that is prepared for fast adaptation. There are two steps in the process of MAML training. For the first step, the model learns an initial point θ for a classifier f_θ and can be optimized via gradient descent on loss $L_{\mathcal{T}_i}$. The update method is shown in Eq. 1, where α is the learning rate and $\mathcal{T}_i$ refers to the sampled tasks. For the second step, the model updates the meta-parameters using a collection of updated model weights θ' via gradient descent with learning rate β, as is shown in Eq. 2.

$$\theta'_i = \theta - \alpha \nabla_\theta L_{\mathcal{T}_i}(f_\theta) \tag{1}$$

$$\theta = \theta - \beta \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} L_{\mathcal{T}_i}(f_{\theta'_i}) \tag{2}$$

Applying MAML on our memory access prediction task for different applications, we sample 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 Clustering

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 to have similar applications for one meta-model, and so we train one meta-model for each set of similar applications. We compare two clustering approaches to detect the sets of similar applications: 1) our previous approach of soft-DTW k-means clustering, and 2) proposed Delegated model clustering.

3.3.1 Soft-DTW K-means clustering

We construct the similarity matrix of the given set of application traces using soft-DTW [4] and then apply k-means to cluster the memory accesses. Soft-DTW is a differentiable approximation of DTW (dynamic time warping) as is shown in Eq. 3, where X is a sequence with length n, Y is a sequence with length m, $\pi = [\pi_0, \ldots, \pi_K]$ is a path, where $\pi_k = (i_k, j_k)$ satisfies:

- boundary condition: $0 \leq i_k \leq n$, $0 \leq j_k \leq m$, $\pi_0 = (0, 0)$, and $\pi_K = (n-1, m-1)$
- monotonicity condition: $i_{k-1} \leq i_k$, $j_{k-1} \leq j_k$
- continuity condition: $i_k - i_{k-1} \leq 1$, $j_k - j_{k-1} \leq 1$

A smoothing parameter γ is introduced to the original min operation in DTW to create a generalized min operator, as is shown in Eq. 4. 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.

$$\text{soft-DTW}_\gamma(X, Y) = \text{soft-}\overset{\gamma}{\underset{\pi}{\min}} \sum_{(i,j)\in\pi} \left\| X_i, Y_j \right\|^2 \tag{3}$$

$$\text{soft-}\underset{\gamma}{\min}(a_1, \ldots, a_n) = -\gamma \log \sum_i e^{-a_i/\gamma} \tag{4}$$

3.3.2 Delegated model clustering

We propose a novel Delegated model (DM) clustering approach based on the assumption that the clustering of sequences can be achieved by the clustering of trained parametric models (delegated models). For each application, the trained specialized model (see Sect. 4.2) has learned the patterns from the trace and the information is stored in the model weights. Thus, the specialized DCLSTM models can serve as delegated models and they represent the original memory traces in the further clustering step. The Delegated model clustering algorithm is shown in Algorithm 2. We concatenate the different types of weights in DCLSTM models and use principal component analysis (PCA) to reduce the dimension. K-means is applied to cluster the dimension-reduced model weights as the last step. DM clustering has some advantages compared to the sequence-based soft-DTW

K-means approach. First, for long sequences like memory traces, DTW requires a large memory space and a long fitting time. As a result, we need to sample pieces of sequence from the whole trace. In contrast, our approach is based on LSTM that processes a sequence recurrently, which considerably saves the memory space and thus can deal with much longer sequences. Second, model weights directly reflect the latent attributes of the sequence for the target task of prediction while DTW can only compare the shape of pieces; this makes Delegated model method can better fit the rerun of applications under different configurations. DM approach can be generalized to other problems, and the delegated model can also be various types of models, such as RNN, GRU, and TCN [3,9].

Algorithm 2 Delegated model Clustering

```
1: $S$: A set of applications
2: $C$: Clusters of applications
3: function DM- CLUSTERING($S$)
4:     for Application $A_i$ in $S$ do
5:         Train a doubly compressed LSTM as the delegated model $DM_i$
6:         $DMW_i \leftarrow$ model weights matrix of $DM_i$
7:         Weight dimension reduced to $d$: $DMW_i^d \leftarrow$ PCA($DMW_i$, $d$)
8:     Clustering to $k$ sets: $C \leftarrow$ k-means($DMW$, $k$)
9:     return $C$
```

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.

3.4 Ensemble meta-learning

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 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 [17].

Algorithm 3 Doubly Compressed LSTM with cluster based MAML

```
1: function C- MAML- DCLSTM($S$)
2:     Clustering applications in $S$ into a collection of sets $\{S_i\}_{i=1}^k$
3:     for $i \leftarrow 1$ to $k$ do
4:         $\theta_i \leftarrow$ MAML-DCLSTM($S_i$)
5:     return $\{\theta_i\}$
```

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 [11] 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 [7,17].

We evaluate the performance of all models using two branches of memory traces generated from PARSEC benchmark:

– Consistent configuration. Assume for a single application, the configuration does not change during running or when restarted. In this case, we split a trace for model training, testing, and retraining.
– Inconsistent configuration. Assume that the configuration of an application will be different during running or when restart. In this case, we generate different pieces of traces by rerunning the benchmark under different configurations. The training set and testing set are from different traces under inconsistent configurations. The retraining is processed using the increasingly seen data from the testing trace.

4.2 Model settings

We used the doubly compressed LSTM (DCLSTM) architecture as described in [17]. 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/C-MemMAP.

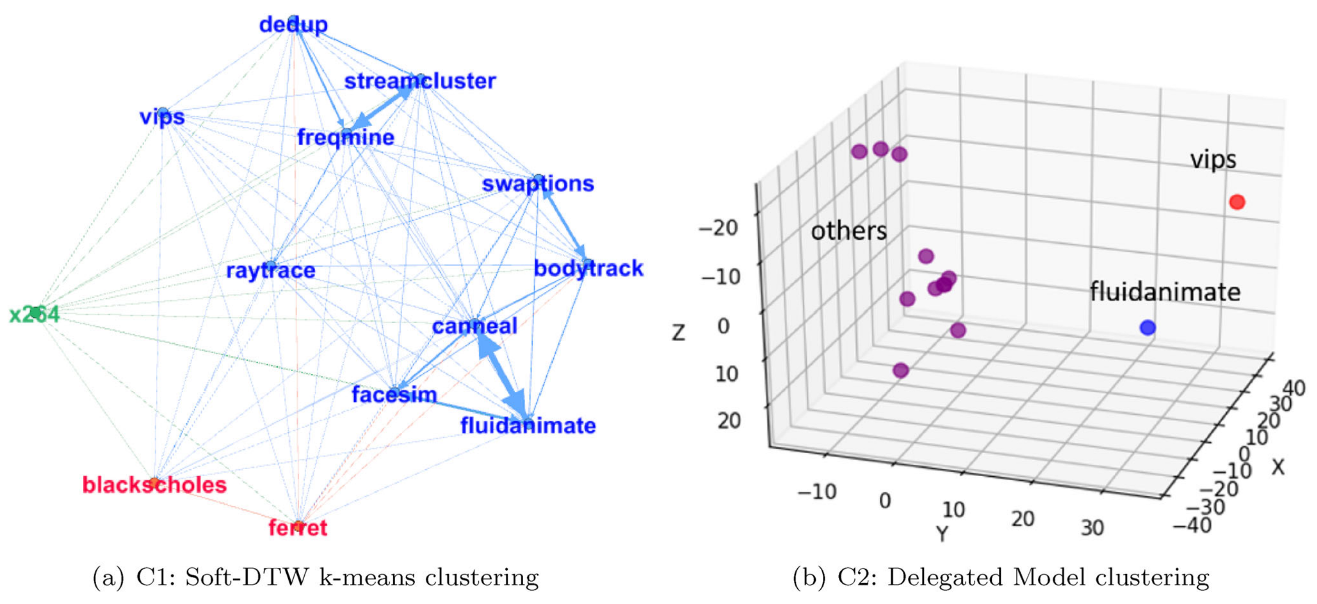


Fig. 4 Clusters obtained from PARSEC benchmarks

- 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 3. 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. For our experiments, we have chosen $k = 3$ based on the assumption that three meta-models work in parallel in a system. We discuss two clustering approaches in Sect. 3.4. For consistent configuration traces, the two clustering approaches agree with each other. For inconsistent configuration traces, while the soft-DTW k-means clustering keeps the same, Delegated model shows a different result. The clustering results are shown in Fig. 4, where C1 refers to soft-DTW k-means clustering and C2 refers to delegated model clustering.

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. We evaluate the performance of the proposed approach using two branches of traces introduced in Sect. 4.1: consistent configuration traces and inconsistent configuration traces.

4.3.1 Consistent configuration

Under the same configuration, adaptability and generalizability are decoupled and can be evaluated separately. Also, since two clustering approaches agree on the same results, we can focus on the general improvement acquired from clustering and ignore the influence of different clustering methods.

Figure 5 shows the accuracy results of all the methods under consistent configuration. 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. 5, 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. While for some applications the accuracy drops are

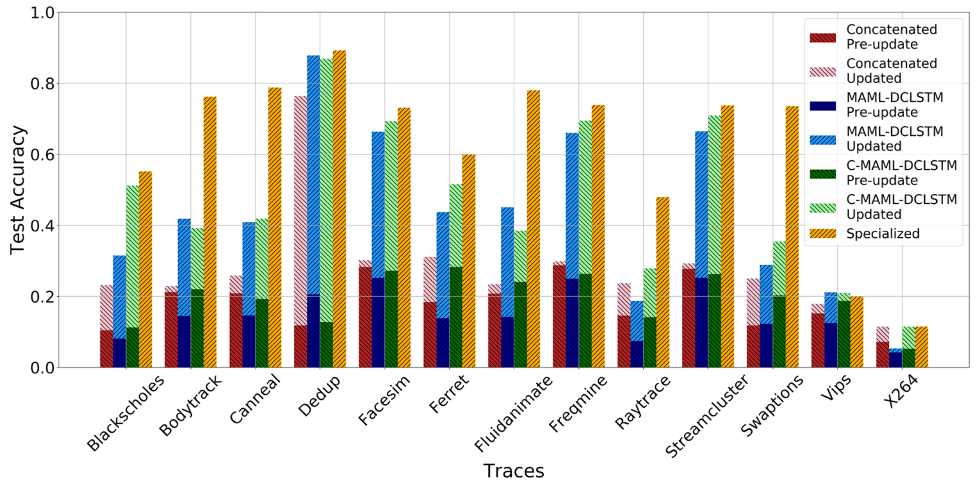


Fig. 5 Accuracy of the models pre- and post-retraining under consistent configuration

notable, the performance is still better than the non-neural-network methods (e.g., Last Access Prediction and Naïve Bayes) explored in [17]. Besides, with the increase in seen data and the continued retraining, the accuracy will grow and approach the specialized models.

Figure 6 shows how retraining starting from various models improves the accuracy as more of the trace is seen under consistent configuration. 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 Streamcluster indicates 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 models 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 7 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 Streamcluster. Furthermore, the C-MAML-DCLSTM model can obtain accuracy close to specialized models using only a small number of batches.

4.3.2 Inconsistent configuration

Each application rerun three times under different configurations and generate traces $T1$, $T2$, and $T3$. We use $T1$ and $T2$ as training traces and $T3$ as the testing trace. The performance of a model under inconsistent configuration reveals both its adaptability and generalizability. Due to the clustering difference between the two approaches (Sect. 3.4), the influence of clustering approach can be explored under this set of experiments.

Figure 8 shows the accuracy results of all the methods under inconsistent configuration traces. Though a specialized model is still trained and tested by a single application, the training and testing traces are different. The inconsistency leads to generally lower accuracies than the consistent configuration results. For the concatenated model, 2 training traces of all applications, each with 100k deltas, are concatenated as the training data. For MAML-DCLSTM, all training traces are fed into one meta-model as different tasks. For C1/C2-MAML-DCLSTM, there are k ($k = 3$)

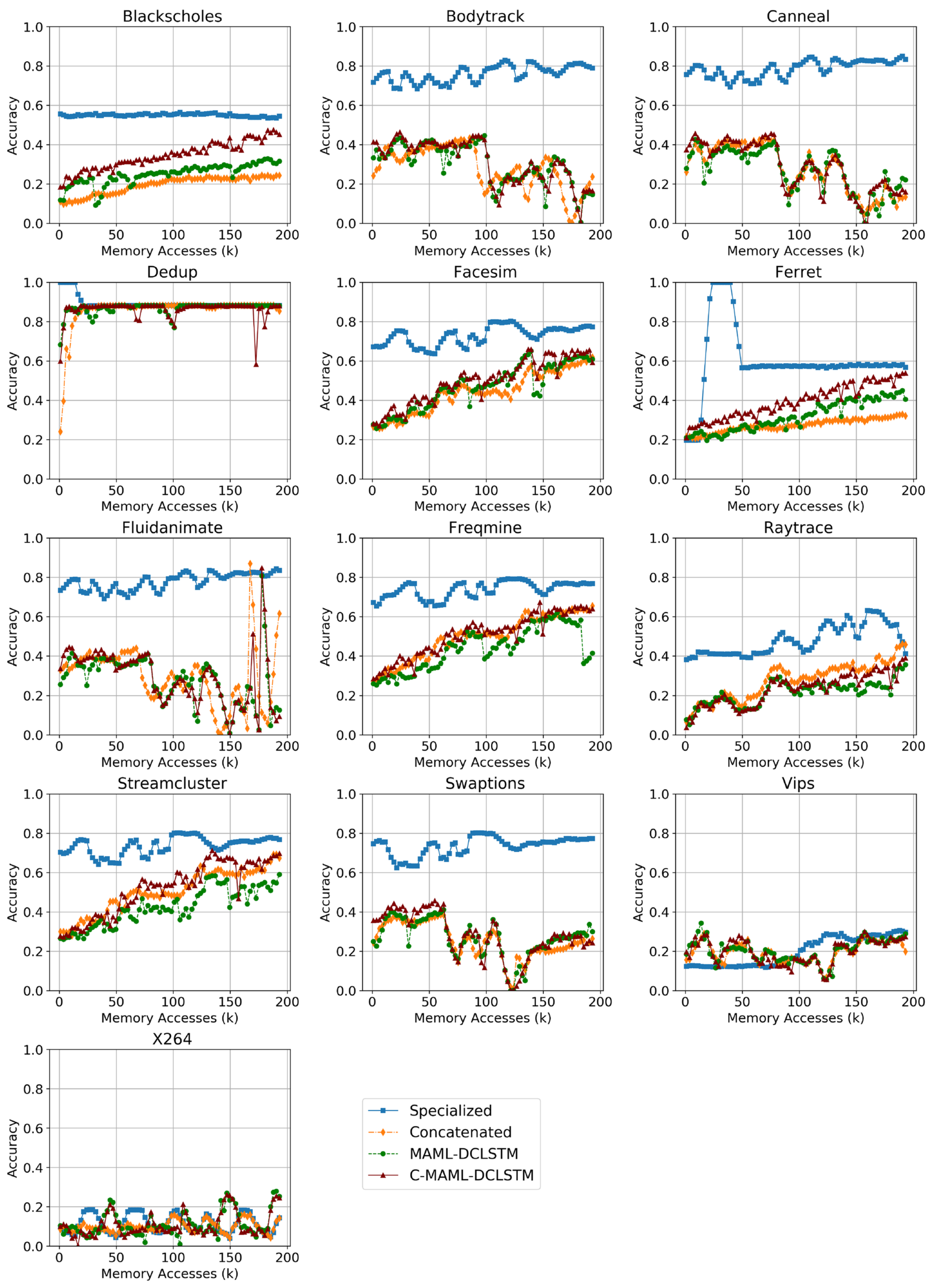


Fig. 6 Adaptability results for our meta-models under consistent configuration

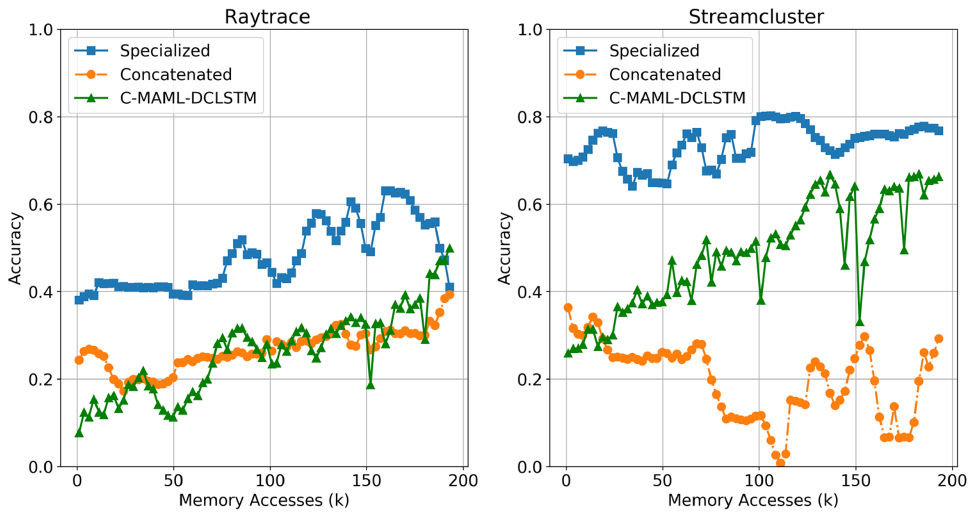


Fig. 7 Generalizability results for our meta-models

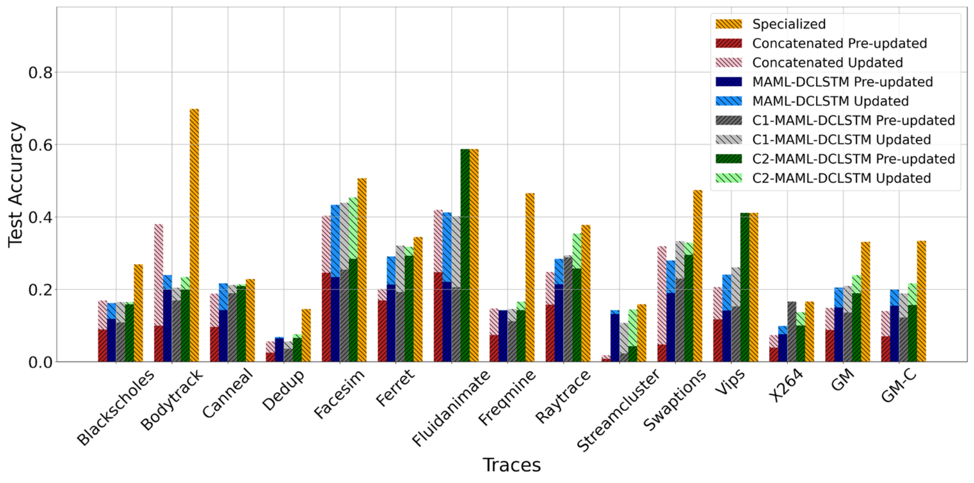


Fig. 8 Accuracy of the models pre- and post-retraining under inconsistent configuration

meta-models and the training traces under the same cluster are fed into the corresponding models. C1 means the clusters acquired from the sequence-based soft-DTW k-means approach, and C2 means the clusters acquired from the Delegated model approach. We compared the model performance before retraining (pre-update) and after retraining (updated) by testing trace. We show the geometric mean of all the model performance in GM. GM shows that C2-MAML-DCLSM has the best performance for both pre- and post-updated, while the concatenated model has the lowest mean accuracy. However, there is a flaw in using the geometric mean of all applications to compare the performance of models. The two clustering results (C1 and C2) have different minor clusters (application member < 3) that achieve high accuracies. This results in higher influence of these applications in computing mean. Therefore, we show the geometric mean of only the common applications in the largest cluster for both C1 and C2 in GM-C. CM-C shows that C2-MAML-DCLSTM achieves 15% higher accuracy than C1-MAML-DCLSTM, who even shows lower accuracy than MAML-DCLSTM without clustering. This result demonstrates that the clustering approach is vital in C-MAML-DCLSTM performance and an inappropriate clustering even hinders the meta-model training. Delegated model clustering (C2) extracts abstract information from the whole trace and is more reliable to cluster rerun traces under inconsistent configuration.

Figure 9 shows one epoch retraining using unseen traces under inconsistent configurations. The retraining batch size and other settings are the same as consistent configuration experiments. Specialized models are still used for reference. From the plots, we can observe that, in many cases, C2-MAML-DCLSM shows higher starting points (Dedup, Facesim, and Freqmine) or achieves higher accuracies more quickly (Blackscholes, Bodytrack, Canneal, Ferret, and Raytrace) than concatenated models. This illustrates that C2-MAML-DCLSM adapts faster than concatenated models. C1-MAML-DCLSTM performs best only for X264. In some cases (Fluidanimate and Freqmine) it performs even worse than MAML-DCLSTM without clustering. This performance drop caused by inappropriate clustering is also observed and discussed in the accuracy evaluation shown in Fig. 8. We can also observe that the adaptability process

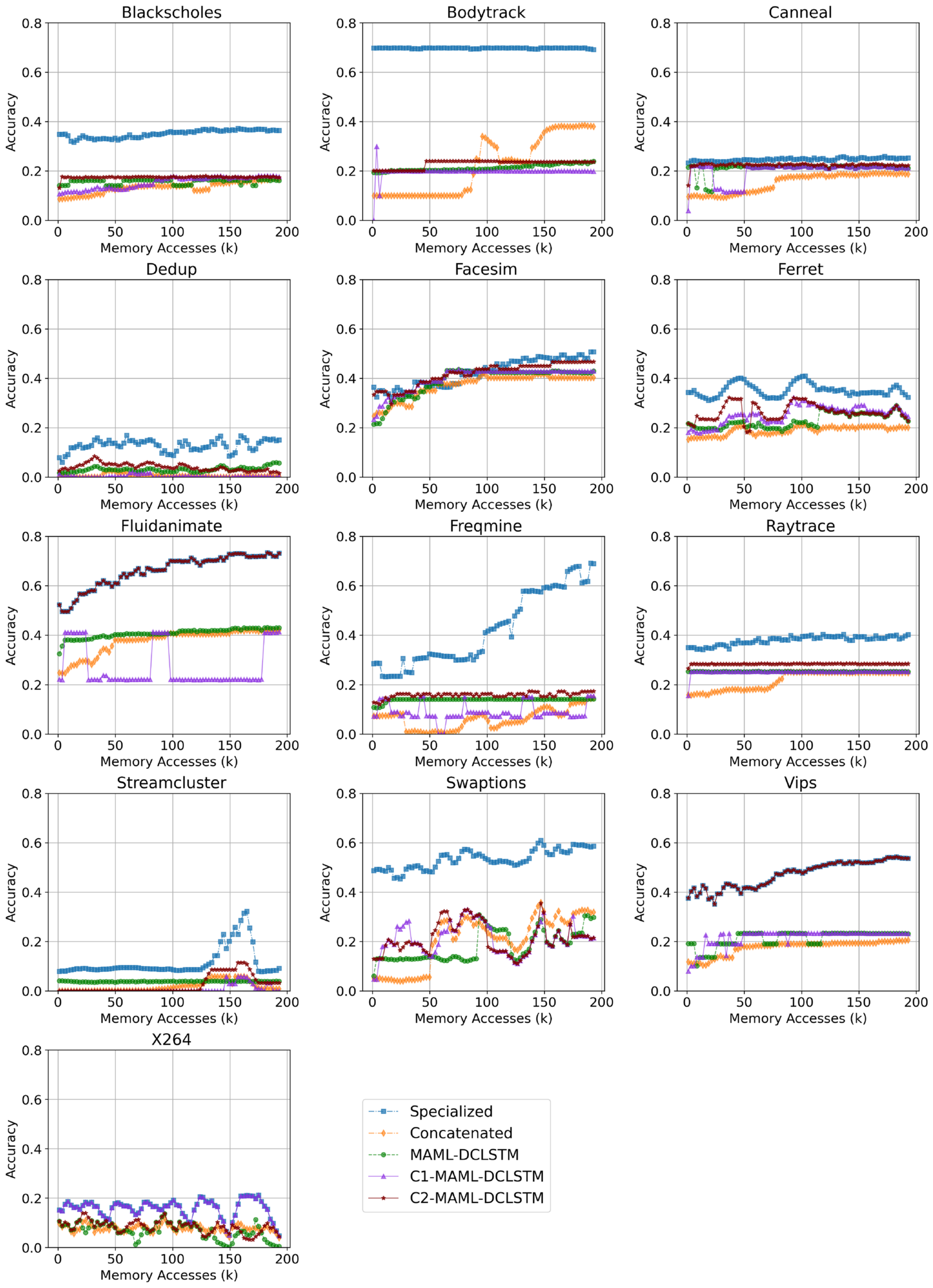


Fig. 9 Adaptability results for our meta-models under inconsistent configuration

of the concatenated model is much slower from the curves of Canneal, Fluidanimate, Raytrace, and Vips. After 50–100 batches of retraining, the curves of different models converge to similar patterns, which shows that the advantage of C-MAML-DCLSTM is more significant at the beginning stage of retraining due to its high adaptability and generalizability.

4.4 Discussion

4.4.1 Sensitivity of hidden dimension

In the experiments above, we set the hyper-parameter of the hidden dimension of LSTM as 50. This configuration is inherited from work [17] because the LSTM layer with 50 units is enough for the LSTM models to achieve high performance while keeping a compact size. This configuration is suitable for the updated models because the retrained C-MemMAP will converge to the specialized model after retraining from our observation from Fig. 6. However, the sensitivity of pre-updated C-MemMAP models influences more to the adaptability process since it determines the initial state of retraining.

Figure 10 illustrates the predicting accuracy of the pre-updated meta-models at various hidden dimensions. Hidden dimension size at 10, 20, 30, 40, 50, 60, and 70 is tested. Results show that the predicting accuracy increases fast from dimension 10 to dimension 50, and then, the curves are significantly flattened at the dimension of 60 and 70. This result demonstrates that hidden dimension at 50 not only guarantees a satisfactory final accuracy but also provides a high initial point for model adapting from retraining.

4.4.2 LSTM versus GRU

In the proposed C-MemMAP model, we used LSTM as the recurrent layer. Gated recurrent units (GRUs) is another commonly used variation of recurrent neural networks, introduced in 2014 by Kyunghyun Cho et al. [2], which can also be applied to our approach replacing the LSTM layer. Figure 11 illustrates the effect of replacing the LSTM layer with a GRU layer. We found that the two versions produce nearly identical results.

4.4.3 Partial accuracy

The prediction output of our approach is in the format of encoded 16-bit binary values. An advantage of this format is that we are allowed to use only upper n bits of the predictions if that part provides higher accuracy. From the hardware point of view, memory fetching is executed in the unit of cache lines so the lower bits of a memory access address are naturally ignored in prefetching. Figure 12 shows a case study of how the partial accuracy and confidence decrease

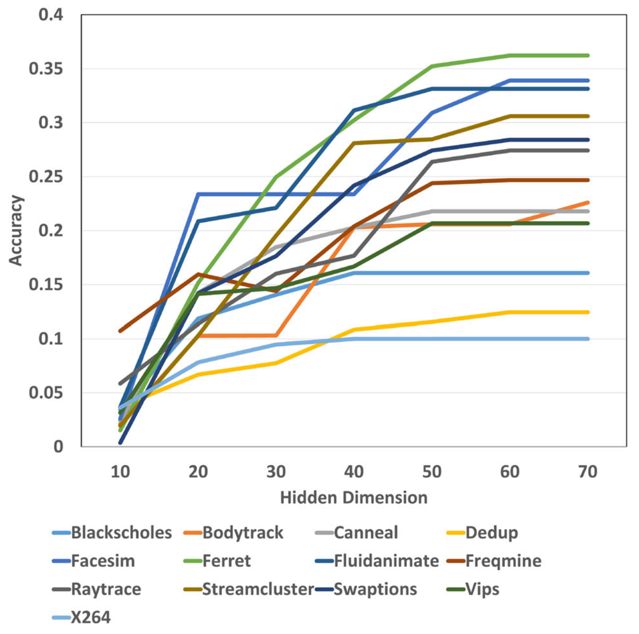


Fig. 10 Pre-updated accuracy of C-MemMAP model at various hidden dimension of LSTM

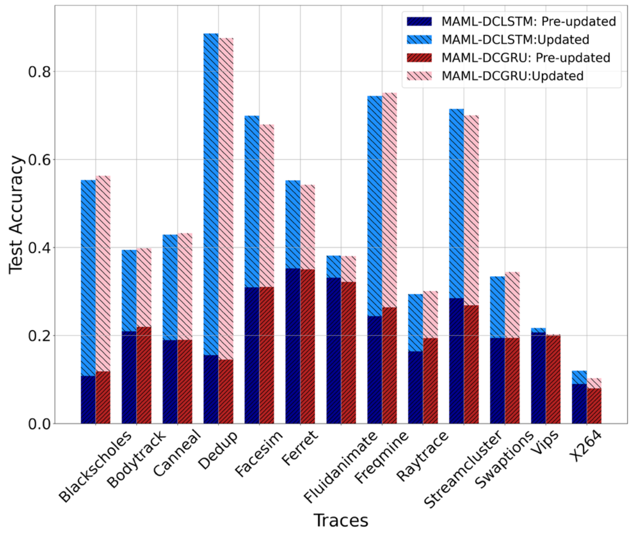


Fig. 11 GRU version of C-MemMAP models acquire similar performance compared to LSTM version

with the increase in prediction bits. The result is based on the application Blacksholes and we define the confidence of upper n bits as Eq. 5:

$$\text{Confidence}(n) = \prod_{b=1}^{n} P\left(y_{\text{pred}}(b) = y_{\text{test}}(b) \mid X, M\right) \tag{5}$$

where $P\left(y_{\text{pred}}(b) = y_{\text{test}}(b) \mid X, M\right)$ is the probability of correct prediction at bit b given the input X and model M.

From Fig. 12, we observe that the accuracy and prediction confidence is high in the upper 8 bits, but both drop rapidly

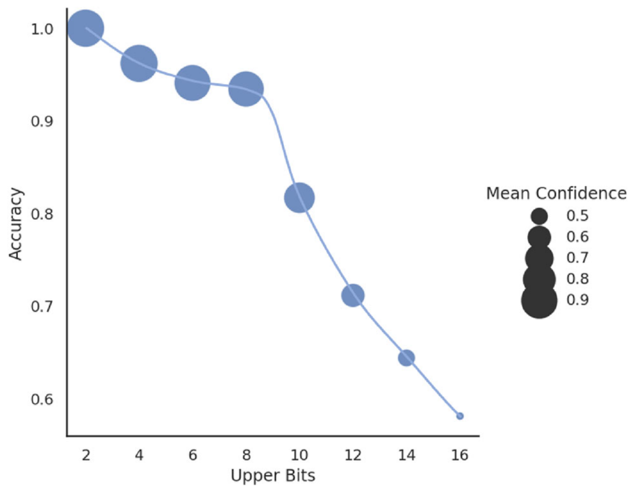


Fig. 12 Prediction accuracy and confidence decrease with the increase of prediction bits for application Blackscholes

for bits longer than 10. This feature supports our hypothesis that by using only upper bits of the prediction, the predictor can achieve higher accuracy, which will lead to more useful prefetches in a practical system.

5 Conclusions

We have proposed C-MemMAP, a clustering-driven 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. While it is possible to train one model for all applications, the accuracy was typically lower. We propose to use a clustering-driven meta-learning approach, where the applications are first clustered and then a meta-model is trained for each cluster. We introduce a novel Delegated model clustering approach that uses the weight matrices of trained LSTM models as delegates of the original traces for the further clustering step. 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. Experiments under inconsistent configuration show that Delegated model is reliable in supporting meta-models and shows 15% higher accuracy than soft-DTW k-means. In future work, we will explore the hardware implementation of our approach to support a prefetcher.

Acknowledgements This work is supported by Air Force Research Laboratory Grant No. FA8750-18-S-7001, and National Science Foundation award number 1912680. Cesar A. F. De Rose was supported by CNPq PDE Grant and Fapergs PRONEX project. The authors would like to thank Dr. Angelos Lazaris for useful discussions.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

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 '08, pp. 72–81. ACM, New York, NY, USA (2008). https://doi.org/10.1145/1454115.1454128
2. Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)
3. Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)
4. Cuturi, M., Blondel, M.: Soft-dtw: A differentiable loss function for time-series. In: Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML'17, pp. 894–903. JMLR.org (2017). http://dl.acm.org/citation.cfm?id=3305381.3305474
5. 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-Volume 70, pp. 1126–1135. JMLR. org (2017)
6. Gers, F.A., Schmidhuber, J., Cummins, F.: Learning to forget: Continual prediction with lstm (1999)
7. Hashemi, M., Swersky, K., Smith, J.A., Ayers, G., Litz, H., Chang, J., Kozyrakis, C., Ranganathan, P.: Learning memory access patterns (2018)
8. Hashemi, M., Swersky, K., Smith, J.A., Ayers, G., Litz, H., Chang, J., Kozyrakis, C., Ranganathan, P.: Learning memory access patterns. CoRR arXiv:1803.02329 (2018)
9. Lea, C., Flynn, M.D., Vidal, R., Reiter, A., Hager, G.D.: Temporal convolutional networks for action segmentation and detection. In: proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 156–165 (2017)
10. 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 (2009)
11. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: Building customized program analysis tools with dynamic instrumentation. SIGPLAN Not. **40**(6), 190–200 (2005). https://doi.org/10.1145/1064978.1065034

12. Narayanan, A., Verma, S., Ramadan, E., Babaie, P., Zhang, Z.L.: Deepcache: A deep learning based framework for content caching. pp. 48–53 (2018). https://doi.org/10.1145/3229543.3229555
13. Peled, L., Weiser, U., Etsion, Y.: A neural network memory prefetcher using semantic locality (2018)
14. 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)
15. 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 (2015)
16. Shevgoor, M., Koladiya, S., Balasubramonian, R., Wilkerson, C., Pugsley, S.H., Chishti, Z.: Efficiently prefetching complex address patterns. In: 2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), pp. 141–152. IEEE (2015)
17. 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)
18. Srivastava, A., Wang, T.Y., Zhang, P., De Rose, C.A.F., Kannan, R., Prasanna, V.K.: Memmap: Compact and generalizable meta-lstm models for memory access prediction. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 57–68. Springer (2020)
19. 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)
20. 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 (2017)
21. Zhang, P., Srivastava, A., Brooks, B., Kannan, R., Prasanna, V.K.: Raop: Recurrent neural network augmented offset prefetcher. In: The International Symposium on Memory Systems (MEMSYS 2020) (2020)