Revisiting Data Prefetching for Database Systems with Machine Learning Techniques

Yu Chen[1], Yong Zhang[1], Jiacheng Wu[1], Jin Wang[2], Chunxiao Xing[1]
[1] RIIT, TNList, Dept. of Computer Science and Technology, Tsinghua University, Beijing, China.
[2] Computer Science Department, University of California, Los Angeles.
{zhangyong05,xingcx}@tsinghua.edu.cn; {y-c19,wu-jc18}@mails.tsinghua.edu.cn; jinwang@cs.ucla.edu;

2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00218

***Abstract*—Among diverse parts in database systems, database prefetching, which aims at predicting future page access patterns and fetching pages to be accessed ahead of time to mitigate blocked I/O operations, plays a crucial role in the overall performance tuning. Existing approaches just use simple heuristic-based methods and suffer from the low hit rate and extra I/O overhead. Recently, with the emerging success of machine learning in different applications, attempts using learning-based models to augment or improve components for database systems have shed some light on this tough problem. Impressed by the enormous potential of machine learning in data management, we present an end-to-end deep learning-based framework to predict page access patterns. We model the prediction of page access as a classification problem and evaluate several variants of neural networks on the accuracy of prediction. On the basis of it, we propose a new Multi-Model framework to construct an accurate model for prefetching. On a suite of real-world database benchmarks, our neural network based prefetching model consistently outperforms existing widely used solutions in real-world database systems.**

I. Introduction

Buffer management [1] plays a significant role in the storage management of relational DBMS where data is accessed in the unit of *page*. Since data access from disk is much slower than in-memory data access, a major goal of the database system is to minimize the number of page transfers between the disk and memory. To this end, two categories of techniques are proposed, namely prefetching and replacement. When the buffer pool is full, it requires a replacement policy [2] to select the pages to be evicted; while prefetching [3] aims at predicting future page access and fetching the pages that are most likely to be visited into the buffer pool before query execution. In this paper, we aim at studying the problem of prefetching in relational database systems.

In database systems, there are three types of access patterns [4]: (1) locality within a transaction, (2) random accesses by transactions, and (3) sequential accesses by long queries. The state-of-the-art prefetching methods such as One Block Lookahead (OBL) [3] can cope with the first and third access patterns very well. They use a heuristic-based approach for page prefetching. In other words, they just fetch the next pages locally adjacent to the currently referenced page. However, such heuristic-based methods are not able to effectively deal with more complicated query workloads with the second pattern. For example, the DBMSs nowadays always use index structures to speed up complicated queries when faced with the huge amounts of data. While indexes can speed up the query performance, they also break the sequentially access in schemes. As the massive unordered patterns are nearly unpredictable by traditional ways, earlier heuristic-based approaches would fail to capture such patterns.

Over the last few years, many database related research fields have greatly benefited from the rapid growth of machine learning techniques [5]. Specifically, neural network models have demonstrated great power in automatically learning patterns from large scale datasets. In this paper, we follow this route and utilize neural network models to improve the performance of database prefetching. We formalize database prefetching as a classification problem with the idea of discretization [6]. Then we investigate different neural network models in the task of database prefetching. On the basis of it, we devise a Multi-Model framework which can better learn the page access pattern for database systems.

The contributions of this paper are summarized as follows:

- To the best of our knowledge, we are the first to adopt learning-based models to touch the problem of database prefetching and propose a learning-based framework.
- We formalize the database prefetching problem and investigate several mainstream neural network models. We also propose a Multi-Model framework to further improve the effectiveness.
- We conduct extensive experiments on three real-world database benchmarks. Experimental results show that our methods significantly outperform the traditional heuristic-based approach.

The rest of this paper is organized as following: Section II formally describes the problem setting. Section III introduces the overall framework. Section IV proposes the learning-based models for prefetching. Section V reports the experimental results. Finally, Section VI concludes the paper.

II. Problem Formulation

The core of database prefetching is how to predict future access pages that are not resident in the buffer pool. After that, it fetches these on-disk pages ahead of time before query execution. In order to formulate database prefetching into a prediction problem with learning-based models, we first figure out the input and output of it.

To this end, we first describe the organization of database storage. As introduced before, the data is organized in the unit

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00218

of *page*. There is a challenge to track the page access history, i.e. how to uniquely identify different pages. This goal can be reached via the *page offset*, which is recorded in the file header of database pages. The page offset is the start position of the page in the table space by unit of page size. Given the value of page offset, we can uniquely determine a page. Consequently, the access pattern can be recognized from the *sequence of page offsets*. To describe the sequence, we adopt the terminology of "timestep" to denote the position of page offset in a given sequence. Therefore, we define $Prefetching$ as follows:

Definition 1 (Prefetching)**.** *Predict and fetch what pages might be needed soon based on pages being previously accessed. The input of database prefetching would be a sequence of $N-1$ page offsets, which are database pages previously accessed. And the output would be the page offset at timestep N that needs to be fetched into the buffer pool.*

III. Learning-based Framework

In this section, we introduce overall architecture of our machine learning framework for database prefetching. We first give an overview in Section III-A. Next we discuss how to prepare the data for above proposed model in Section III-B. Finally we provide the evidence to formulate the problem as a classification problem in Section III-C.

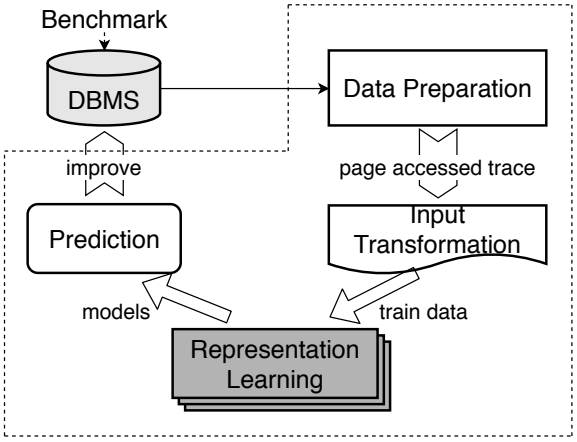


Fig. 1: Overall Architecture

A. Architecture Overview

The overall framework is shown in Figure 1. It consists of four stages: data preparation, input transformation, representation learning, and prediction.

In the data preparation stage, we acquire training data in the following way: We first run some queries on the targeting database. Then we collect the traces of page access in the transaction level: for each transaction, it extracts the page accessed trace by submitting the transaction to the database. A training instance consists of two parts: the transaction number and the page accessed trace.

During the input transformation stage, we transform the page offsets(d_i) into the *extent* [1] related representations. This transformation aims to alleviate the sparsity problem of page offsets, which will be detailed in Section III-B later. Then input representations of pages are fed to neural network models for further encoding.

In the representation learning stage, we learn the representation of the page offsets with deep neural networks. In the training processing, our model is trained on trace generated by different transactions to learn various access patterns. We investigate several popular deep learning models and propose a new multi-model to improve the performance in Section IV.

During the prediction stage, when a new transaction begins, we reset the state of the model and then perform prediction. The representation generated by the prediction model will be transformed into the page offsets and notice the database to fetch the corresponding pages from disk.

B. Data Preparation

To better utilize the learning-based models for database prefetching, we need to overcome the challenge brought by the sparsity of target space. As the size of a database could be rather large, there would be a broad range of page offsets. For example, a typical size of database files of the TPC-H benchmark could be $10GB$, which corresponds to approximately 10^7 possible target pages. Obviously, these offsets are pervasive throughout the whole range, making it rather difficult to train the model.

Following some techniques in MySQL, we take advantage of the notion extent proposed in databases. By definition, one extent is a group of consecutive pages within a data file that holds data of tables. DBMS usually manages table space by extents. For example, the pages inside in the same extent of MySQL usually have the same page type. As the number of extents in a table is significantly fewer than that of pages, it is easier for a learning-based model to predict the position of extent a page belongs to. In order to locate at a specific page, we also need to predict the offset of a page within the extent.

Thus we divide the space formed by page offsets into multiple extents. To simplify the calculation, we require the number of pages which an extent contains should be the power of 2. And every extent has the same number of pages.

Till now, we transform the page offset d_i into two components: the extent offset e_i and the in-extent offset f_i in the following ways.

$$e_i = \lfloor \frac{d_i}{\text{\# pages in a extent}} \rfloor \tag{1}$$

$$f_i = d_i - e_i \times \text{\# pages in a extent} \tag{2}$$

where the extent offset e_i represents which extent a page offset d_i belongs to; the in-extent offset f_i represents the page offset inside the extent. In Innodb's default setting environment of MySQL, each extent contains 64 pages, for a page offset $d_i = 76$, the extent offset is $e_i = 1$, and in-extent offset is $f_i = 12$.

[1] https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_extent

Then we map the page offset d_i to its representation triplet $\boldsymbol{t}_i = \langle d_i, e_i, f_i \rangle$ and use it as the input of neural network models. The reason we corporate d_i into the representation is that we want to keep the associated information of extent offset e_i and in-extent offset f_i with d_i.

C. Classification vs. Regression

Based on above discussion, we observe that it would be more proper to treat the prefetching problem as classification rather than regression. The reason is that the output is a page offset to be fetched, which belongs to a large, discrete vocabulary, i.e. all pages in the table space managed by the DBMS. In this case, a classification model could gain better flexibility by being able to produce multi-modal outputs.

Regarding the output of classification, we utilize softmax function to obtain the page offset distribution. However, the number of softmax targets, i.e. the DBMS table space, is still too large. To improve the accuracy of prediction at the same time, we adopt the similar idea in dealing with the input. We use the (in-)extent offset of page offset as the target of softmax. That is, our model outputs the extent offset and in-extent offset distributions rather than the original page offset distribution.

IV. Methodology

Here we investigate how different neural network models to be used as the prediction model. Besides, we propose a prefetch decision module to further imporve the overall performance of prefetcher. Then we further propose a Multi-Model framework that is more adaptable to the variety of page access patterns. Finally, we put more attention on how to select the suitable page offsets from the output of our models.

A. Single Prediction Models

We first introduce the choice of prediction model for page access patterns. Here we investigate three mainstream neural network models. We are fully aware that there could be other models that can serve as the component for prediction. But the goal here is not designing the "best" methods for this task, but rather finding the capabilities of the mainstream models. We leave utilizing other advanced models in database prefetching as future work.

Deep Neural Network (DNN) belongs to the category of feed forward network. It is a very primitive depth model, and it has made a lot of contributions to machine learning. Meanwhile, **Convolutional Neural Network (CNN)** has been widely used for image classification [7] as well as other image-related problem. Previous studies apply 1-d CNN into tasks of sequence prediction [8], [9], [10]. **Recurrent Neural Network (RNN)** is also effective at predicting access patterns. Both simple RNN and Long short-term memory (LSTM) model have been used in machine translation [11] and text analysis [12]. We use DNN, 1-d CNN, simple RNN and LSTM as our prediction models.

Next we present how a prediction model interacts with other components. The input of the model would be a series of the representation triplet $\boldsymbol{t}_i$ of page offsets with a specified sliding time window T after input transformation stage (for RNN and LSTM models, the window size T is as large as the whole historical accessed page offsets inside one transaction). The model contains an embedding layer connected with the model specific layers (DNN, CNN or RNN layers), and two fully connected layers both directly connecting the model specific layers as input, finally each fully connected layer is connected with a softmax layer. Here the output vector of one softmax layer represents the distribution of extent offset $p(e_i)$ and that of another softmax layer represents the distribution of in-extent offset $p(f_i)$. We then acquire the K output representations $\boldsymbol{o}_i$ from these two distributions. Therefore, the models, mathematically, need to learn the mapping that $f(\boldsymbol{t}_{n-T}, \boldsymbol{t}_{n-T+1}, \cdots, \boldsymbol{t}_{n-1}) = \boldsymbol{o}_n$ for each timestep n where $\boldsymbol{t}_i$ and $\boldsymbol{o}_i$ represent the series input and output representations generated by the sequence of page offsets d_i.

Now, to learn the prefetching model $\mathcal{M}$, the remaining question is the loss function. Notice that our output of model is two seperate distributions of extent and in-extent offset, i.e. $p(\boldsymbol{o}_n) = \{p(e_n), p(f_n)\}$, and the label used for training is a pair of offsets $\hat{\boldsymbol{o}}_n = \{\hat{e}_n, \hat{f}_n\}$. We first use one-hot encoding to transform the label $\hat{\boldsymbol{o}}$ into two 0-1 distributions $p(\hat{\boldsymbol{o}}_n) = \{p(\hat{e}_n), p(\hat{f}_n)\}$. Then the loss $\mathcal{L}$ between the output and label is defined as follows :

$$\mathcal{L}(p(\boldsymbol{o}_n), p(\hat{\boldsymbol{o}}_n)) = -\sum_{e_n} p(\hat{e}_n) \log p(e_n) - \sum_{f_n} p(\hat{f}_n) \log p(f_n) \tag{3}$$

In fact, we just calculate the sum of two cross-entropies for extent and in-extent distributions.

B. Prefetching Decision

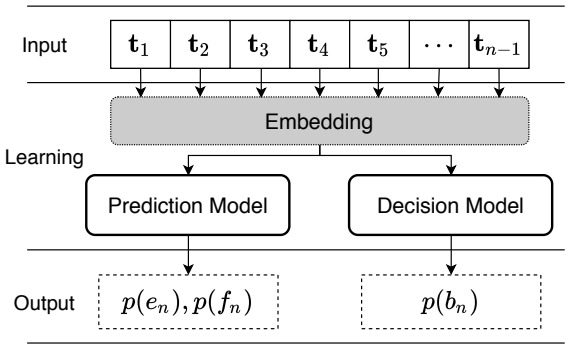


Fig. 2: Single Model with Prefetching Decision

We find that it's not necessary to prefetch pages in real DBMS. In practice, wrong prefetching, though asynchronous, will hurt the performance of the system due to the extra I/O cost. So, if the learning-based prefetcher can be told to do the prefetching operation only at the proper time, the database systems can achieve performance gain.

In reality, this "decision" problem can be considered as a binary classification problem. Thus we propose an extra decision model $\mathcal{D}$ that learns the mapping from input

representation series $\boldsymbol{t}_i$ to the decision of prefetching, i.e., $\mathcal{D}(\boldsymbol{t}_{n-T}, \boldsymbol{t}_{n-T+2}, \cdots, \boldsymbol{t}_{n-1}) = b_n$ where b_n is 1 if we need to prefetch in current timestep n or 0 otherwise.

In Figure 2, we show the data flow of our single model equipped with the decision model. The decision model $\mathcal{D}$ shares the embedding layer with prefetching model $\mathcal{M}$. And the output of the decision model is the decision probability $p(b_n)$ of prefetching pages.

Algorithm 1: Training for Decision Model

Input: Prediction Model $\mathcal{M}$, Input Representation $\boldsymbol{t}_i$.
Output: Decision Model $\mathcal{D}$.

```
1  train_set = ∅.
2  foreach timestep n do
3      Obtain the label ô_n.
4      Obtain p(o_n) by applying M on input series.
5      Select K page offsets from p(o_n) as collection S.
6      if ô_n ∈ S then
7          Mark timestep n with positive label.
8      else
9          Mark timestep n with negative label.
10     Add timestep n into train_set.
11 Train Decision Model D on train_set.
12 return D.
```

We then focus on the training and prediction process of the decision model. The training part is shown in Algorithm 1. We train the decision model $\mathcal{D}$ after the prefetching model $\mathcal{M}$ is well-trained. At each timestep n, we first obtain the current label $\hat{\boldsymbol{o}}_n$ (line 3) and then apply the prediction model $\mathcal{M}$ to obtain K possible pages (line 4-5). If we find that the candidate output $\mathcal{S}$ contains the label $\hat{\boldsymbol{o}}_n$, it means our prefetching model correctly predicts at current timestep. Then we mark this timestep with positive label (line 7). Otherwise it should be considered as a negative case (line 9). Next we add the input representations at timestep n into training dataset with its label (line 10) and train the decision model (line 11). We fix the embedding layer during the training process of the decision model.

After training the decision model, we utilize this model $\mathcal{D}$ to improve the performance of our prefetching models. At each timestep in the prediction phase, we first apply the decision model on input and obtain the decision probability $p(b_n)$. If the probability is larger than a given threshold, we run prefetching model $\mathcal{M}$ to predict the pages.

C. A Multi-Model Framework

Though its effectiveness in learning patterns, the above approach has certain limitations. In most case the transactions are generated from fixed number of SQL templates, the transactions from different SQL templates will produce dissimilar access patterns. So it is not practical for a single model to fit the different types of patterns for all the transactions. A good way to solve this problem is using multiple models to predict the various types of patterns separately, and then collect different output predictions to obtain a more precise prediction. The whole idea is somehow similar to the ensemble learning [2].

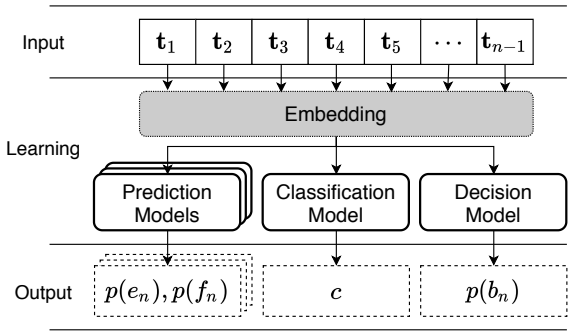


Fig. 3: Multi-Model Framework

In Figure 3, our Multi-Model contains G different prefetching models $\mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_G$ and an extra classification model $\mathcal{C}$. They share the same embedding layers in order to assure the same encoding on input series for different models. These different prefetching models are recommended to have the same neural network architecture but with different parameters. They accept the sequence of embedded input triplets and decide the next accessed page offsets, respectively.

The classification model $\mathcal{C}$ contains similar architecture with prefetching models. It just receives the embedded input series and output classification probability distribution $\boldsymbol{c}$, i.e., a vector of length G whose elements represent the probability that the corresponding prefetching model will be chosen.

Given $\boldsymbol{c}$ of $\mathcal{C}$ and the extent and in-extent distributions $p_j(\boldsymbol{o}_n) = \{p_j(e_n), p_j(f_n)\}$ of prefetching models $\mathcal{M}_j$, we can merge them to $\bar{p}(\boldsymbol{o}_n) = \{\bar{p}(e_n), \bar{p}(f_n)\}$ as follows:

$$\{\bar{p}(e_n), \bar{p}(f_n)\} = \{\sum_{j=1}^{G} \boldsymbol{c}_j \cdot p_j(e_n), \sum_{j=1}^{G} \boldsymbol{c}_j \cdot p_j(f_n)\} \quad (4)$$

And the loss function for Multi-Model can be calculated similar with Equation 3.

In the prediction stage, to select the final K page offsets, we do not simply combine the output distribution of different prefetching models. We further propose more strategies in Section IV-D.

Here, we also add an extra decision model to further improve the prefetching performance. Copy the idea from Section IV-B, we train our decision model after the classification model and prefetching models are fully trained. From our evaluation in Section V, we show that CNN single model has the best performance among all the single models. So we only use CNN model inside the Multi-Model.

Finally, we admit that the Multi-Model has a more complex structure and the number of parameters is larger than that of single models. Specifically, in our experiment, the total number of trainable parameters for DNN, CNN, RNN, LSTM, and Multi-Model are 13 million, 8 million, 16 million, 16 million and 20 million, respectively. However, the prefetching models

[2]https://en.wikipedia.org/wiki/Ensemble_learning

in the Multi-Model can train and predict in parallel, and won't incur too much overhead compared with these single models.

D. Output for Prediction

As the models introduced in the above sections output extent offset e_i and in-extent offset f_i, we propose the ways to transform (in-)extent offsets into real page offset.

Ouput Selection for Single Model: The output of the single model is the distributions of extent offset $p(e_i)$ and the in-extent offset $p(f_i)$. We directly take the most likely extent offset e_i and the most likely in-extent offset f_i to get the page offset d_i. For instance, if we need to obtain the K page offsets, we take out the most likely x extent offsets and the most likely y in-extent offsets and combine them in pairs to get $x \times y$ page offsets where $x = y = \sqrt{K}$.

Ouput Selection for Multi-Model: For a Multi-Model which contains G single models, we firstly compute the merged distribution of the extent and in-extent offset like the loss function did, then we use the approaches mentioned for the single model to select the final outputs.

V. Evaluation

In this section, we first introduce the experiment setup. Then we show the effectiveness of different prefetching methods.

A. Experimental Setup

1) Datasets: We evaluate the prefetchers using three popular benchmarks for database systems mentioned in [13].

- TPC-H It is the current industry standard for evaluating the performance of OLAP systems [3].
- TPC-DS It is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance [4].
- SSB The Star Schema Benchmark (SSB) is designed to measure performance of database products in support of classical data warehousing applications.

We mainly focus on the OLAP benchmarks since the disk I/O time is usually a large bottleneck for OLAP benchmarks. In each benchmark, queries are constructed based on about 15 templates. To obtain the train and test data, we run three benchmarks in MySQL and collect the page access traces with warm-start. The number of transactions in TPC-C, TPC-DS and SSB are 1317, 396, 130, accordingly. For each dataset, we first randomly divide the benchmark transactions into two collections, in the portion of 8:2. Then the traces of all transactions of each collection together form the training and testing sets, respectively. During the training process, we use 3-fold cross-validation to tune the models.

2) Metrics and Baselines: We use *Precision* and *Recall* as the metrics. At each timestep, the prefetcher predicts K pages as results. If one of the K pages is the same as the page that is accessed in the next timestep, it is considered as correct. Similarly, we count the number of the unique prefetched page offsets and calculate the ratio of these predicted page offsets to the correct page offsets as *Recall*.

We extend One Block Lookahead algorithm [3] into K Page Lookahead (LookAhead) as the baseline method, which has been implemented in most mainstream DBMSs. We also simulate Random Readahead (Random) algorithm as another baseline, which is currently deployed in MySQL's InnoDB engine. Random Readahead does not prefetch pages everytime, but has a trigger condition which is similar as our *Prefetch Decision* module described in Section IV-B.

As is discussed above, we choose the DNN, CNN, RNN, LSTM and the proposed Multi-Model as prediction models of the prefetcher. We use the Adam optimizer [14] to train the models with learning rate as 0.0001.

B. Model Comparison

We compare different learning-based models with the baseline LookAhead and Random on different benchmarks. The results are shown in Table I. For all methods, we prefetch $K = 9$ pages at every timestep. And we use $G = 5$ as the number of inside prefetching models for Multi-Model. We observe that the Multi-Model significantly outperforms all other models in both precision and recall. It is because that the Multi-Model can better adapt to different access patterns even with extremely large data volume.

Except for the Multi-Model, CNN ranks second among other learning-based models. To our surprise, RNN variants do not perform well in our experiments. The reason is that due to the inherited characteristics of DBMSs, the current page access usually correlates with recent traces. As CNN is good at learning and mastering neighborhood features, it is more suitable for capturing the patterns of database page access.

Besides, all learning-based models significantly outperform LookAhead and Random, which demonstrates the necessity of applying learning-based models for the task of database prefetching. The reason for a relatively high precision and recall is that: Since the train set covers queries from all kinds of templates, the models are not difficult to capture such patterns and thus have good results even traces in train and test sets are not overlapped with each other.

In Figure 4, we investigate the influence of K over the results of effectiveness. As the Random prefetcher can not control the number of prefetch pages. We just exclude it in investigating the best value of K.

From all sub-figures of Figure 4, we observe that the improvement of both precision and recall is limited when $K \geq 9$. We also notice that it is difficult for neural network models to achieve high accuracy with prefetching only one page. The main reason is that with more page predictions at one time, the model can cover more different targets. Besides, the overhead of prefetching 9 pages is also tolerable. Therefore, it is proper to select $K = 9$ as the default setting.

VI. Conclusion

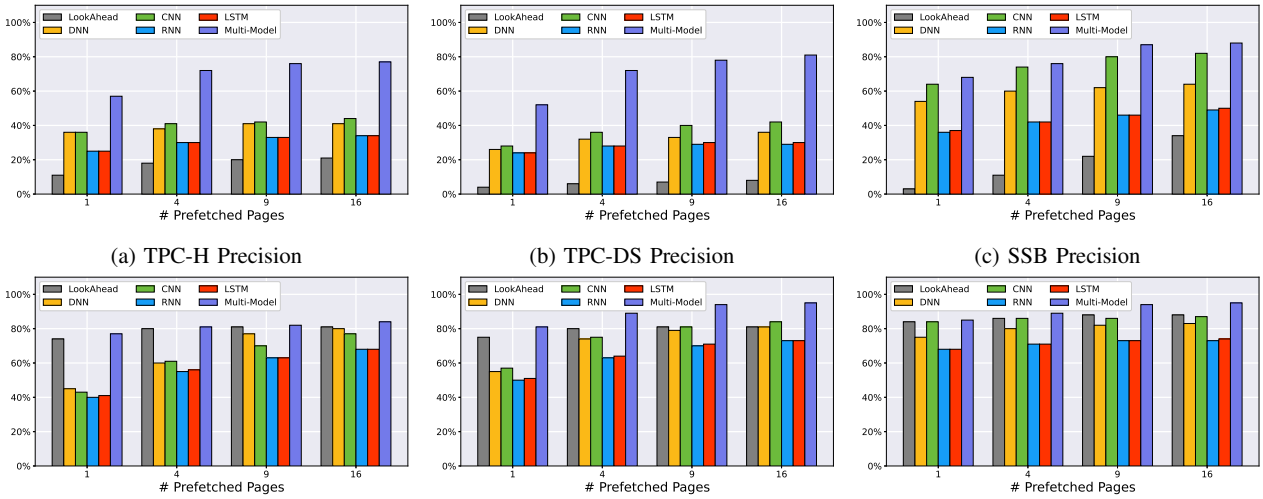
Database prefetching is an essential procedure in improving the overall performance. In this paper, we present a machine

[3] http://www.tpc.org/tpch/
[4] http://www.tpc.org/tpcds/

TABLE I: Model Comparison

	TPC-H		TPC-DS		SSB	
	Precision(%)[1]	Recall(%)	Precision(%)	Recall(%)	Precision(%)	Recall(%)
LookAhead	20/-	81	7/-	81	22/-	88
Random	14/28	80	7/15	78	40/55	85
DNN	41/75	77	33/78	79	62/69	82
CNN	41/85	70	40/79	81	80/74	86
RNN	33/64	63	29/62	70	46/62	73
LSTM	33/64	63	30/62	71	46/63	73
Multi-Model	**76/87**	**82**	**78/87**	**94**	**87/84**	**94**

[1] We measure 2 precisions: Overall-Precision / Decision-Precision. Overall-Precision means we measure the prefetcher's precision all the timesteps, and Decision-Precision only counts when the prefetcher decides to make a prefetching.



(a) TPC-H Precision (b) TPC-DS Precision (c) SSB Precision

(d) TPC-H Recall (e) TPC-DS Recall (f) SSB Recall

Fig. 4: The Influence of Number of Pages to Prefetch

learning based framework to improve the performance of database prefetching. Compared with traditional heuristic-based approach, the neural network models are more powerful in learning complicated patterns from the history of database page access. We formalize the database prefetching problem as a classification task and investigate the performance of different neural network models. To learn a better representation for prediction, we propose a Multi-Model framework to encode the page access patterns. Experimental results on three real world database benchmarks demonstrate that our method has significant performance gain over heuristic-based approaches in both precision and recall.

Acknowledgment This work was supported by NSFC(91646202), National Key R&D Program of China (2018YFB1404401, 2018YFB1402701).

REFERENCES

[1] W. Effelsberg and T. Härder, "Principles of database buffer management," *ACM Trans. Database Syst.*, vol. 9, no. 4, pp. 560–595, 1984.

[2] Z. Jiang, Y. Zhang, J. Wang, and C. Xing, "A cost-aware buffer management policy for flash-based storage devices," in *DASFAA*, 2015, pp. 175–190.

[3] A. J. Smith, "Sequentiality and prefetching in database systems," *ACM Trans. Database Syst.*, vol. 3, no. 3, pp. 223–247, 1978.

[4] A. Dan, P. S. Yu, and J. Chung, "Characterization of database access pattern for analytic prediction of buffer hit probability," *VLDB J.*, vol. 4, no. 1, pp. 127–154, 1995.

[5] R. Marcus and O. Papaemmanouil, "Towards a hands-free query optimizer through deep learning," in *CIDR*, 2019.

[6] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *ICML*, 2016, pp. 1747–1756.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1106–1114.

[8] D. Palaz, M. Magimai-Doss, and R. Collobert, "Learning linearly separable features for speech recognition using convolutional neural networks," in *ICLR Workshop*, 2015.

[9] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. P. Kuksa, "Natural language processing (almost) from scratch," *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, 2011.

[10] M. Binkowski, G. Marti, and P. Donnat, "Autoregressive convolutional neural networks for asynchronous time series," in *ICML*, 2018, pp. 579–588.

[11] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *ICLR*, 2015.

[12] A. Jagannatha and H. Yu, "Structured prediction models for RNN based sequence labeling in clinical text," in *EMNLP*, 2016, pp. 856–865.

[13] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux, "Oltp-bench: An extensible testbed for benchmarking relational databases," *PVLDB*, vol. 7, no. 4, pp. 277–288, 2013.

[14] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, 2015.