Computers, Materials & Continua

Tech Science Press

DOI: 10.32604/cmc.2023.039236
Article

Characterization of Memory Access in Deep Learning and Its Implications in Memory Management

Jeongha Lee[1] and Hyokyung Bahn[2,*]

[1]Institute of Artificial Intelligence and Software, Ewha University, Seoul, 03760, Korea
[2]Department of Computer Science and Engineering, Ewha University, Seoul, 03760, Korea
*Corresponding Author: Hyokyung Bahn. Email: bahn@ewha.ac.kr
Received: 17 January 2023; Accepted: 11 April 2023; Published: 09 June 2023

Abstract: Due to the recent trend of software intelligence in the Fourth Industrial Revolution, deep learning has become a mainstream workload for modern computer systems. Since the data size of deep learning increasingly grows, managing the limited memory capacity efficiently for deep learning workloads becomes important. In this paper, we analyze memory accesses in deep learning workloads and find out some unique characteristics differentiated from traditional workloads. First, when comparing instruction and data accesses, data access accounts for 96%–99% of total memory accesses in deep learning workloads, which is quite different from traditional workloads. Second, when comparing read and write accesses, write access dominates, accounting for 64%–80% of total memory accesses. Third, although write access makes up the majority of memory accesses, it shows a low access bias of 0.3 in the Zipf parameter. Fourth, in predicting re-access, recency is important in read access, but frequency provides more accurate information in write access. Based on these observations, we introduce a Non-Volatile Random Access Memory (NVRAM)-accelerated memory architecture for deep learning workloads, and present a new memory management policy for this architecture. By considering the memory access characteristics of deep learning workloads, the proposed policy improves memory performance by 64.3% on average compared to the CLOCK policy.

Keywords: Memory access; deep learning; machine learning; memory access; memory management; CLOCK

1 Introduction

With the rapid advances in artificial intelligence (AI) technologies of the Fourth Industrial Revolution, deep learning is increasingly being adopted in modern software design. As a result, deep learning has become an indispensable part of our smart living infrastructure [1–3]. Various modern applications internally perform image processing and text analysis with deep learning frameworks such as TensorFlow [4,5]. Mobile applications also make use of learning techniques for intelligent services [6].

As the data size of deep learning increasingly grows, analyzing the memory access characteristics of AI workloads becomes important. Although the memory size of the system continues to grow, it is not easy to accommodate the entire memory footprint of ever-growing AI workloads because of the density limitation in the manufacturing process of Dynamic Random Access Memory (DRAM) and the large energy consumption of DRAM media [7,8]. In particular, the manufacturing process of DRAM cannot scale down the density below 5 nanometers, and the energy consumption of DRAM increases largely following the memory capacity. Since DRAM is volatile memory, consistent recharge of each cell is needed to maintain the stored data even though the data is not read or written [7,8]. Note that this recharge operation of each cell accounts for a dominant portion of energy consumption in memory systems [9].

For this reason, analyzing memory accesses is important to design efficient memory management policies for deep learning workloads. In this paper, we analyze the memory access characteristics of deep learning workloads consisting of text and image data. To the best of our knowledge, this is the first attempt to quantify the memory access behavior of deep learning workloads. There are several steps in deep learning workloads: loading the dataset, preprocessing, building the model, training the model, and testing the model. Of these, we collect memory access traces while building and training the model as it is a major step in deep learning that requires most of the system resources. Specifically, we analyze read and write operations separately for instruction and data memory accesses. Based on our analysis, we find out some special characteristics of deep learning memory accesses, which are quite different from conventional desktop workloads such as games, office software, document viewers, and photo browsers. First, when comparing instruction and data accesses, data access accounts for a dominant portion of memory accesses in deep learning workloads, which is quite different from traditional workloads. Specifically, data access accounts for 96%–99% of deep learning workloads while 70%–85% of traditional workloads. Second, when comparing read and write accesses, write access accounts for 64%–80%, which is also different from traditional workloads where write accounts for 6%–55%. Third, although write access accounts for the majority of memory accesses, it exhibits low access bias. Specifically, the Zipf parameter of write access is about 0.3, and the parameter is smaller in text workloads than in image workloads. Fourth, in predicting memory re-access, recency is important in read access, but the frequency is necessary for accurate estimation in write access.

Based on these observations, we present a new memory architecture that makes use of Non-Volatile Random Access Memory (NVRAM) to accelerate the memory system for deep learning workloads. As a large amount of data is generated during the training phase of deep learning workloads, performance degradation is inevitable unless the DRAM size is increased to accommodate it. Instead of increasing the DRAM size of the system, we observe that NVRAM can provide an alternative solution to compensate for the limited DRAM size in deep learning workloads. This will provide significant implications to address rapidly growing memory demands in deep learning workloads. In particular, while deep learning workloads generate explosive memory demands for data accesses during training, our architecture defends against memory thrashing by absorbing data pages evicted from DRAM. In the case of instruction pages, our policy simply discards them without flushing to NVRAM or storage like conventional workloads as they are read-only pages so the same versions already exist in secondary storage. In our architecture, page tables always reside in DRAM, so they never have a chance to enter NVRAM. By so doing, the proposed architecture eliminates most of the storage I/Os caused by deep learning data access considering the memory access characteristics we observe. That is, a small size of NVRAM in our architecture absorbs a large portion of data access although deep learning workloads become larger than the given DRAM capacity. As the retrieval latency from storage is not uniform depending on the backup location (i.e., NVRAM and secondary

storage), we need a new memory management policy for NVRAM-added architectures. In this paper, we present a new memory management policy that considers the different I/O costs and memory access characteristics of deep learning workloads.

Memory management is a traditional topic, so there have been many attempts to improve memory performance. Specifically, the page eviction policy plays a central role in memory management, which determines victim pages to be discarded from memory when there is no free memory space to accommodate new page requests. Although extensive studies have been performed on eviction policies, most of them have focused on the consideration of specific storage characteristics (e.g., flash memory) [10] or workload characteristics (e.g., mobile applications) [11]. Unlike previous works, our study focuses on the analysis of memory access characteristics originating from deep learning workloads and accelerating performance by adopting NVRAM-added architectures. As the CLOCK policy [12] is the most popular one that is being adopted in the current operating systems like Linux, we use CLOCK as the baseline eviction policy and validate the improvement of our policy against the existing architecture and new architecture with CLOCK. Simulations based on the memory access traces of four popular deep learning workloads consisting of Internet Movie DataBase (IMDB), Spam detection, Modified National Institute of Standards and Technology database (MNIST), and Fashion MNIST show that the proposed policy improves the memory performance by 64.3% on average compared to the CLOCK policy.

The rest of this paper is organized as follows. Section 2 describes the method of collecting memory access traces for deep learning workloads, and presents the characterization results of memory accesses focusing on access types and operations. In Section 3, we analyze the memory access characteristics of deep learning workloads with respect to access bias. Section 4 describes the prediction of memory re-access based on recency and frequency characteristics in deep learning workloads. Section 5 explains a new memory management policy suggested in this paper. Section 6 validates the proposed policy based on simulation experiments. Section 7 briefly summarizes studies related to this paper focusing on NVRAM technologies. Finally, Section 8 concludes this paper.

2 Analysis of Deep Learning Memory Accesses

TensorFlow [13] and Pytorch [14] are well-known deep learning frameworks for generating learning models with Convolution and Long Short-Term Memory (LSTM) layers. In this paper, we extract memory access traces while executing TensorFlow with LSTM and Convolution layers. For collecting memory access traces of deep learning workloads, we make use of the Callgrind module of the Valgrind toolset [15]. There are several tools that can capture memory access traces such as Pintools [16] and Valgrind [15]. Of these, we use Valgrind as it has been used in previous studies and we need to analyze our traces in comparison with the desktop traces collected by Valgrind [17]. In our trace collection, memory accesses were captured as the result of read/write requests from the last level cache to the main memory. Our trace collection is based on whether memory access was actually performed regardless of how address translation was done. That is, whether address translation is performed via the TLB (Translation Lookaside Buffer) or the page table is irrelevant to the point of view we traced. We extract memory access traces of four popular deep learning workloads: IMDB, Spam detection, Fashion MNIST, and MNIST. The brief descriptions of these workloads are as follows.

- IMDB [18]: identifying positive or negative ratings from 50,000 movie reviews by utilizing 1-dimensional Convolution layers.
- Spam detection [19]: determining whether an email is a spam or not based on the content of the email by utilizing LSTM layers.
- FashionMNIST [20]: classifying 10 kinds of clothing images including shoes, bags, and pants by utilizing 2-dimensional Convolution layers.
- MNIST [21]: classifying text images of digits 0 to 9 by utilizing LSTM layers.

Fig. 1a shows the distributions of memory accesses for the four deep learning workloads we experiment with. We also show the memory access distributions of some traditional workloads consisting of game, office, PDF (Portable Document Format), and photo for comparison purposes as shown in Fig. 1b. The game trace was extracted while playing the traditional card game application called Freecell; the office trace was collected while editing a document file by the text editor software Gedit; the photo trace was captured while executing the image browser software called Geeqie; the PDF trace was collected while viewing a PDF file by the document viewer application KGhostview [17].

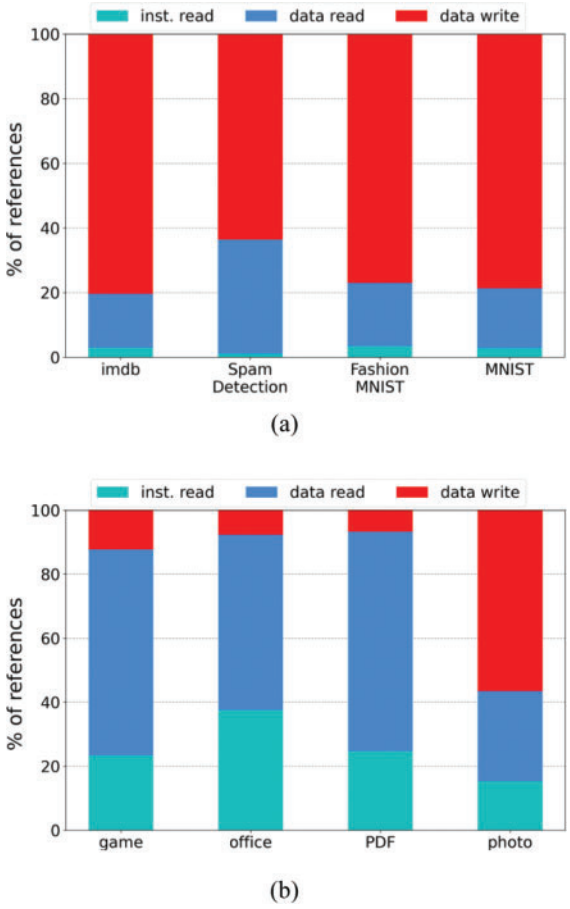


Figure 1: Memory access distributions of (a) deep learning workloads and (b) conventional workloads; IMDB = internet movie database; MNIST = modified national institute of standards and technology database

There are two memory access types (i.e., instruction and data), and two operations (i.e., read and write). As instructions are fetched from the code region of memory that has read-only permissions, write operations are not allowed on instruction-type memory accesses. Thus, we can classify memory accesses into three types, i.e., the instruction read, data read, and data write. As shown in Fig. 1a, when comparing instruction and data accesses, data access accounts for a dominant portion of memory accesses in deep learning workloads. Specifically, instruction access accounts only for 1%–3.3% of all deep learning workloads we consider. Note that this is not the case for traditional workloads where 15%–30% are instruction accesses as shown in Fig. 1b.

Another important observation is that write operations account for a large portion of total memory accesses in deep learning workloads. Specifically, data write is responsible for 64%–80% of total memory accesses regardless of workload types as shown in Fig. 1a. However, in traditional workloads shown in Fig. 1b, read access accounts for a majority of memory accesses in most cases though the writing is dominant in some workloads like the photo. Specifically, the ratio of write access in traditional workloads is 12%, 7%, 6%, and 55%, respectively, for game, office, PDF, and photo.

Fig. 2 shows the distributions of distinct memory accesses for the same workloads in Fig. 1. That is, we count only once for the same memory address although the location is accessed multiple times in Fig. 2. By observing the results in Figs. 1 and 2, we can conclude that the size and the number of data to be accessed in deep learning workloads are excessively larger than traditional workloads for executing a similar ratio of instructions.

Meanwhile, in order to see how these access characteristics affect the performance of the actual system, we measured LLC (last-level cache) misses and page faults in deep learning workloads and traditional workloads. In our experiment, the measurement result of the LLC miss rate ranged from 10% to 20%, and there was no significant difference between the traditional workloads and the deep learning workloads. This is because LLC focuses on maintaining the micro-level working set of the currently running process, so a certain ratio of cache misses inevitably occurs during working-set transitions. Nevertheless, DRAM accesses caused by LLC misses do not significantly affect overall system performance as DRAM access latency is not very large compared to storage access latency. On the other hand, if the allocated memory size is very small to accommodate the footprint of the workload, frequent page faults will occur, resulting in significant performance degradation due to slow storage access. In this paper, we measured the major faults caused by storage access and observed that the traditional workloads showed 40–100 faults/sec, while the deep learning workloads showed 3,000 to 10,000 faults/sec, indicating memory thrashing situations. In order to resolve this issue, the memory capacity should be increased during the training phase of the deep learning workloads. Our preliminary experiments showed that the memory size required for deep learning workloads is 2 to 8x to achieve performance comparable to traditional workloads. However, when we add system memory for this purpose, resources will be wasted during idle time and power consumption will also keep increasing. Thus, an effective solution to deal with this problem should be considered.

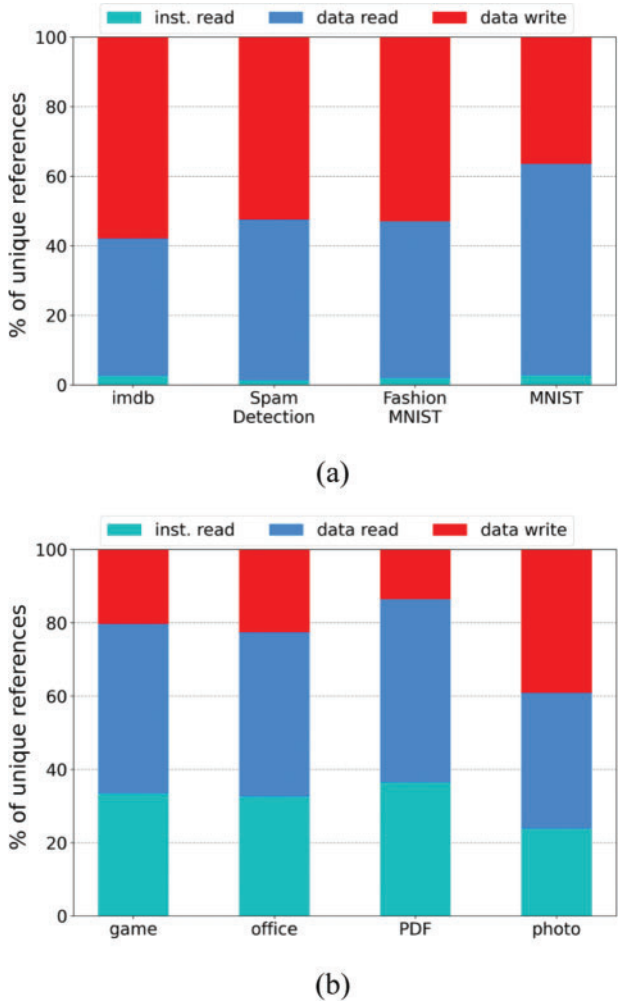


Figure 2: : Distinct memory access distributions of (a) deep learning workloads and (b) conventional workloads; IMDB = internet movie database; MNIST = modified national institute of standards and technology database

3 Memory Access Bias in Deep Learning Workloads

In this section, we analyze the characteristics of memory accesses in deep learning workloads focusing on access bias. This is important for determining the hot data of deep learning workloads that reside in memory and setting an appropriate size of memory for the system running the workload. Skewed popularity distributions are usually modeled by the Zipf distribution, where the access frequency of the i-th most popular page is proportional to $1/i$. The Zipf distribution comes from quantitative linguistics [22], where the frequency of a word's usage is inversely proportional to its rank in an ordered frequency list, and can also be applied to model biases in web pages accessed [23] or TV

channels watched [24]. In our problem, the access probability P_i of the i-th popular memory page is expressed as

$$P_i = \frac{(1/i)^{\theta}}{\sum_{k=1}^{n}(1/k)^{\theta}} \tag{1}$$

where n is the total number of memory pages accessed and θ $(0 \leq \theta \leq 1)$ is the Zipf parameter that determines the degree of popularity bias. When θ is 0, all memory pages are evenly accessed. As the value of θ increases, the popularity of memory pages is increasingly biased, and finally, when it becomes 1, the popularity is most biased [24]. If we plot the popularity rank of a page *vs.* the number of page accesses on a log-log scale, the Zipf distribution can be fitted with a straight line. Thus, to find the Zipf parameter θ of each workload, we perform a linear regression and extract the slope of the line.

Table 1 lists the Zipf parameter for the four deep-learning workloads that we analyze. As we see from this table, the Zipf parameter of write access is smaller than that of read, implying that the access bias is weaker in write operations. Specifically, the parameter of read is about 0.5 for most cases, but in writes, it is 0.29 to 0.44, which means that memory accesses in deep learning workloads are not excessively concentrated on some hot pages in writes. When comparing the four workloads, the Zipf parameter of text workloads (i.e., IMDB and Spam Detection) is smaller than that of image workloads (i.e., Fashion MNIST and MNIST). For comparison purposes, we also analyze some Zipf parameters of conventional workloads and list them in Table 2. As we see from this table, the Zipf parameter of write access is significantly larger than that of deep learning workloads in all cases.

Table 1: Skewness parameter of deep learning workloads

	IMDB	Spam detection	Fashion MNIST	MNIST
Inst. read	0.50818	0.52530	0.52440	0.52989
Data read	0.48746	0.36947	0.53208	0.53624
Data write	0.36771	0.29781	0.38493	0.44271

Table 2: Skewness parameter of desktop workloads

	Game	Office	PDF	Photo
Inst. read	0.45630	0.52673	0.45341	0.46795
Data read	0.46453	0.49033	0.45977	0.39400
Data write	0.45589	0.79030	0.56293	0.43831

4 Re-Access Estimation of Memory Pages

To accelerate memory performances, it is important to predict future memory accesses well and maintain those pages likely to be re-accessed in memory as much as possible. To do this, the memory system should utilize good estimators that predict future memory accesses. Recency and frequency are two well-known estimators utilized in predicting the re-access of memory pages [25]. We compare these two estimators in deep learning workloads and analyze which one leads to better predictions.

Fig. 3 plots the effect of recency rankings of accessed pages on re-access of pages for the four deep learning workloads. In the figure, the x-axis is the page ranking based on the last access time (i.e., recency ranking), and the y-axis is the number of accesses on that page ranking. For example, ranking

1 is the most recently accessed page in a chronologically sorted order. An increase in rankings along the x-axis implies that more time has passed since the pages have been accessed. The cyan, blue, and red plots represent instruction read, data read, and data write, respectively. Note that we separately maintain page rankings based on access types and accumulate when the corresponding access type happens. As we see from this figure, the curve decreases monotonically within certain top-ranking ranges, implying that recently accessed pages are more likely to be re-accessed soon. When comparing access types, the plots of "data read" and "instruction read" are mostly located above those of "data write" in the rankings of less than 10^3, indicating that recency is a strong estimator in read accesses, but it is relatively weak in write accesses.

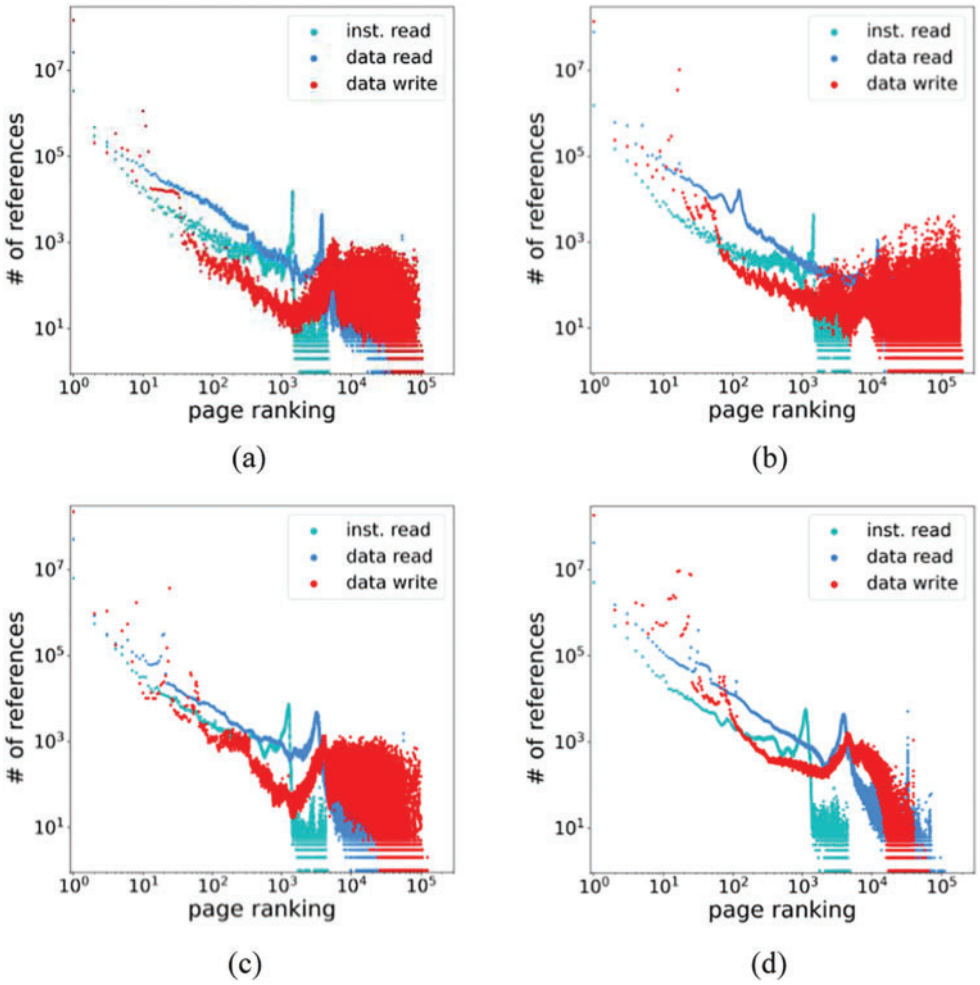


Figure 3: Number of accesses for page rankings based on recency: (a) IMDB (internet movie database); (b) spam detection; (c) fashion MNIST (modified national institute of standards and technology database); (d) MNIST

Similar to the recency estimator, the effect of the frequency estimator can be analyzed based on the corresponding page rankings. To this end, we sort pages based on their access frequency, and whenever a page in a certain ranking is re-accessed, we increase the number of accesses for that ranking by one, which possibly results in the reordering of the rankings.

In Fig. 4, the x-axis is the rankings of pages based on their past access frequency, and the y-axis is the number of re-accesses on the corresponding rankings. As can be seen in this figure, the number of accesses in top-ranking pages is not as large as that of recency ranking curves in Fig. 3. This is possible as the number of references plotted for each ranking in the graph is not the access count of a particular page, but the total count accumulated for a ranking that may represent different pages over time. However, if we add up the reference count for the overall rankings, the sum will be the same in the recency and frequency graphs. This observation indicates that recency ranking can estimate the re-access of memory pages better than frequency ranking for certain top-ranking pages. However, in the case of write access, the frequency estimator consistently exhibits a large number of re-accesses even after top rankings, but the recency estimator shows some irregular patterns such that the number of accesses drops sharply around the rankings of 10^3 and then rises again. This makes recency rankings difficult to estimate future memory write accesses when the number of memory pages to be estimated increases.

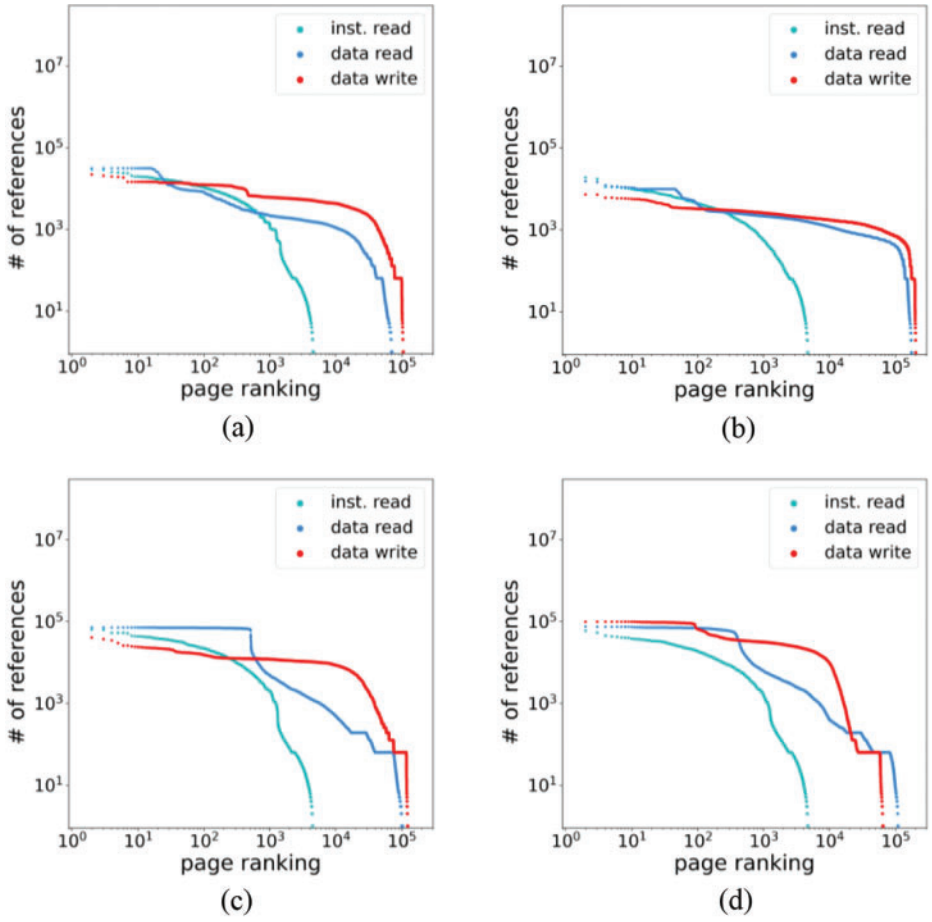


Figure 4: Number of accesses for page rankings based on frequency: (a) IMDB (internet movie database); (b) spam detection; (c) fashion MNIST (modified national institute of standards and technology database); (d) MNIST

To precisely compare the recency and frequency estimators in write access, we extract the write curves from Figs. 3 and 4, and re-plot them in Fig. 5. As clearly shown in this figure, the two curves

intersect roughly at ranking 10^1. This indicates that the recency estimator is more accurate than the frequency estimator for the highest rankings, but the frequency estimator provides consistently good information for overall ranking ranges. We can utilize this result in memory management policies as follows. If we have only a limited memory capacity of 10 pages, we can select the top-ranking pages from the recency estimator. However, more memory capacities are available, an efficient memory management policy should maintain pages suggested by the frequency estimator.

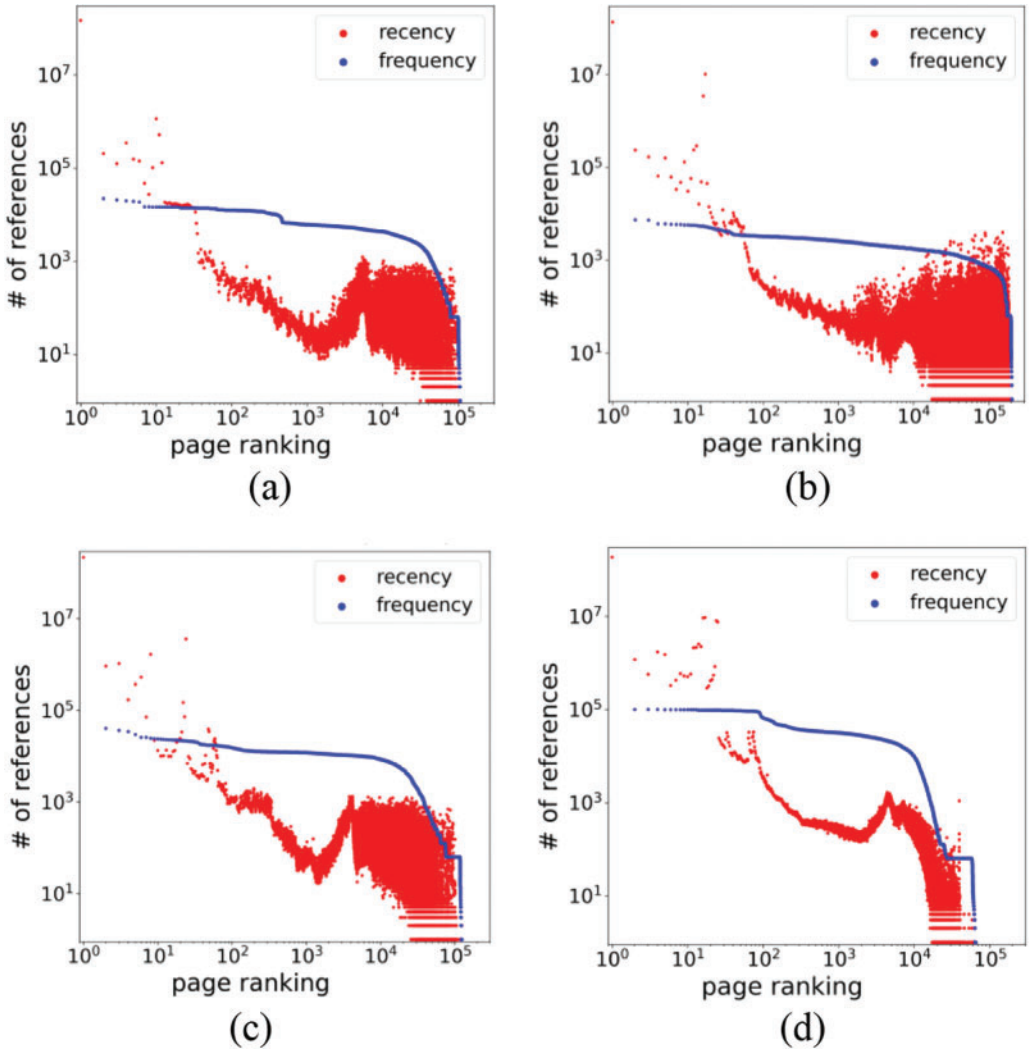


Figure 5: Comparison of recency and frequency distributions in memory write accesses: (a) IMDB (internet movie database); (b) spam detection; (c) fashion MNIST (modified national institute of standards and technology database); (d) MNIST

In summary, the recency estimator is sufficient to predict read access for deep learning workloads as shown in Fig. 3. However, it shows some reversed trends for write access as the rank increases. Therefore, the frequency estimator can be used in conjunction with the recency estimator to accurately predict write access.

5 Implications to Memory Management

In this section, we present a new memory architecture for deep learning workloads that utilize NVRAM in order to accelerate the memory performance of deep learning systems without increasing the system's DRAM size. We then propose a memory management policy designed appropriately for that architecture. The proposed architecture makes use of a small size of NVRAM residing between DRAM memory and secondary storage as shown in Fig. 6. Although some previous studies have already suggested using NVRAM as a storage accelerator or extension of main memory media [11,26], we observe the memory pressure of deep learning workloads, especially for the large number of data pages generated during the training phase. In this situation, instead of increasing the DRAM memory size of the system, we make use of NVRAM as victim memory to prevent thrashing by absorbing data pages evicted from DRAM. In the case of instruction pages, our policy simply discards them without flushing them to NVRAM. NVRAM technologies such as Intel's Optane™ [27] provide fast storage with low energy consumption [28]. However, NVRAM cannot completely replace secondary storage due to its cost per capacity, so we use it as an additional component for performance accelerators [26,29].

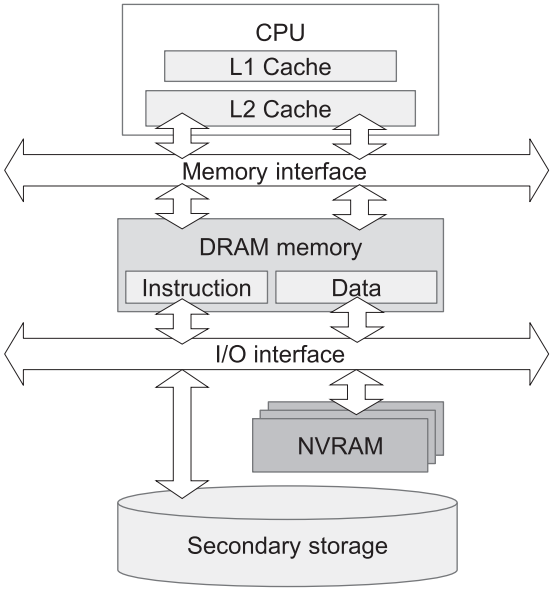


Figure 6: The system architecture that makes use of NVRAM (non-volatile random access memory) for deep learning workloads

Main memory is managed in units of pages, and when a page is requested, the memory management system searches whether the requested page resides in memory. If it is found, the application continues execution by accessing the requested page. Otherwise, the page should be loaded from secondary storage, which is called a page fault situation. Upon a page fault, if there is no available memory space, the memory management system selects a victim page and removes it from memory. However, if the same page is requested again in the future, it should be reloaded from storage. To improve memory performance, minimizing the number of page faults is important, which can be achieved by equipping with a memory size large enough to accommodate the entire data set. However, as the system requires large memory only in the training phase of deep learning workloads, we make use of NVRAM instead of increasing the DRAM memory size.

As instruction access accounts for only a small fraction of memory accesses in deep learning workloads, we do not allow them to enter NVRAM. Also, since instructions are read-only, they do not change after being loaded into memory. This implies that instruction pages in memory necessarily have the same contents as the original files in storage. So we can simply delete pages containing instructions from memory without writing back to storage. In contrast, as data access accounts for a majority of memory accesses in deep learning workloads, we accelerate them by making use of NVRAM. Specifically, data writing is responsible for 65%–80% of total memory accesses as analyzed in Section 2, and we do not need to flush them to slow storage by maintaining them in NVRAM.

By adopting NVRAM as the data access accelerator, instruction pages and data pages have different access costs when they need to be loaded into memory. That is, due to the performance gap between NVRAM and secondary storage, retrieving data pages from NVRAM has a lower cost than retrieving instruction pages from slow storage. So, we design a new memory management policy for deep learning systems, which considers different access characteristics and I/O costs.

Specifically, our policy classifies memory pages into instruction pages, data/read pages, and data/write pages. Then, the policy manages the main memory by logically separating it into three areas, namely the instruction area, data/write area, and data/read area. Each area is then resized based on the access characteristics and their costs. To do so, we add a small size of history list for the three areas to see their access characteristics [30,31]. The history list maintains the metadata of recently removed pages instead of their actual contents. By monitoring accesses to pages in the history list, we can predict the page fault ratio of each memory area. Specifically, the effectiveness of enlarging each memory area can be estimated by history lists. For example, if there are frequent accesses to pages in the history list of instruction area, we can enlarge the instruction area to improve memory performance. As the total memory size is fixed, enlarging the instruction area accompanies the data areas to shrink accordingly.

In addition to access frequencies in the history list, our policy considers different retrieving costs from NVRAM and secondary storage. That is, we give higher memory priorities to the instruction area as the cost of retrieving instruction pages from secondary storage is higher than that of retrieving data pages from NVRAM. For example, if data pages are accessed 100 times more often than instruction pages in the history list, but loading an instruction page from secondary storage is 500 times slower than a data page from NVRAM, we set the priorities of instruction and data areas 5 and 1, respectively. This implies that the instruction area is enlarged 5x faster than the data area for the same number of accesses in the history list. As a data page can be read and then also written, a data page can be included in both data/read and data/write areas at the same time. In this case, only a single copy of the page content is maintained in the physical memory and it is shared by the two areas.

When the system's free memory is exhausted, the memory management policy should select some victim pages and discard them to make available memory space. The most popular policy used for this purpose is CLOCK, which manages all memory-resident pages in a circular list. Although CLOCK was devised in the early days [12], its variants are still being adopted in the current Android and Linux systems. CLOCK sequentially checks the access bit of each page to monitor whether the page has been used recently or not. If the access bit of a page is 1, CLOCK clears the bit instead of evicting it; otherwise, CLOCK selects the page as a victim and removes it.

Similar to the CLOCK policy, we manage the access recency of memory pages based on circular linked lists, but we separately manage the three areas, i.e., instruction area, data/read area, and data/write areas, by utilizing different lists. When we need to find a victim page from the instruction area or data/read area, our policy traverses the circular linked list of the target area and investigates the read access bit of pages. In contrast, if we need to find a victim from the data/write area, our policy

investigates the write access bit of pages. When a page is evicted from one of the instruction, data/read, and data/write areas, its metadata is added to the corresponding history list. When there is no available slot in the history list, the oldest page is evicted.

As the frequency is also important to estimate the re-access likelihood of write access, our policy internally manages data/write area by two sub-areas, namely data/write_area$_{recency}$ and data/write_area$_{frequency}$ as shown in Fig. 7. History lists for these two sub-areas are also maintained separately. Note that data/write_area$_{recency}$ maintains data pages that are written once whereas data/write_area$_{frequency}$ maintains data pages that are written more than once after entering memory. Algorithm 1 shows the pseudo-code of the proposed policy upon a page-fault situation.

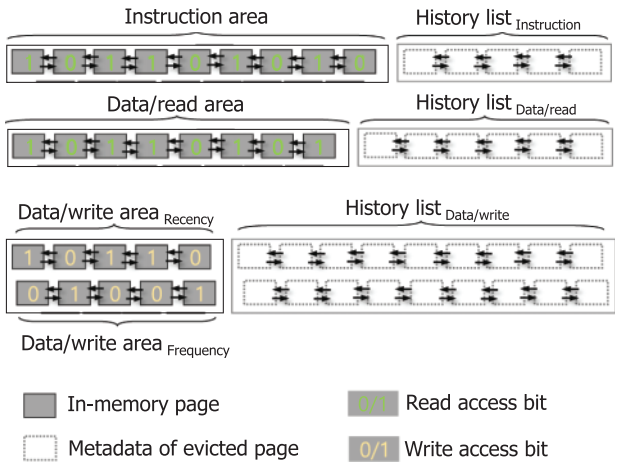


Figure 7: Memory areas and their history list in the proposed policy

Algorithm 1

procedure PAGE-FAULT (page *Pg*)
  Area ← area to add *Pg*;
  $Hist_{Area}$ ← history list of *Area*;
  if no free space in *Area* **then**
    EVICT (*Area*);
  end if
  if no free space in $Hist_{Area}$ **then**
    discard the oldest page from $Hist_{Area}$;
  end if
  if $Pg \in Hist_{Area}$ **then**
    remove *Pg* from $Hist_{Area}$;
    increase the size of *Area* based on access cost;
    adjust the size of other areas;
  end if

(Continued)

Algorithm 1 Continued

```
    add Pg to Area;
end procedure
procedure EVICT (area Ar)
    Pg ← page pointed by circular list of Ar;
    while access-bit (Pg, Ar) = 1 do
        access-bit (Pg, Ar) ← 0;
        advance the pointer of Ar;
    end while
    delete Pg from Ar and add Pg to Hist_Ar;
end procedure
```

6 Performance Evaluations

6.1 Simulation Experiments

In this section, we validate the effectiveness of the proposed memory management policy for deep learning workloads through trace-driven simulations. For our experiments, memory accesses were captured while executing four deep learning workloads, IMDB, Spam Detection, Fashion MNIST, and MNIST as introduced in Section 2, and the traces were replayed under the given architecture with different memory management policies. The hardware specifications of our system for collecting memory access traces consist of an Intel Core i7-11700 2.5 GHz 8-core processor, 8 GB DDR4 memory, and 1TB HDD storage. Tables 3 to 5 summarize the data set characteristics and the experimental parameters of our simulations.

Table 3: Memory access characteristics of deep learning workloads

		IMDB	Spam detection	Fashion MNIST	MNIST
Number of accesses captured	Total	191,485,840	245,134,853	304,472,130	312,882,815
	Inst. read	5,463,881	2,537,635	10,056,350	8,558,141
	Data read	32,131,186	86,539,004	59,803,608	58,138,678
	Data write	153,890,773	156,058,214	234,612,172	246,185,996
Access ratio	Read:write	1:4.09	1:1.75	1:3.36	1:3.69
	Inst.:data	1:34.05	1:95.60	1:29.28	1:35.56

Table 4: Memory access characteristics of desktop workloads

		Game	Office	PDF	Photo
Number of accesses captured	Total	490,175	1,733,763	1,546,135	610,685
	Inst. read	114,233	649,500	380,609	93,242
	Data read	315,902	951,441	1,061,986	172,044
	Data write	60,040	132,822	103,540	345,399

(Continued)

Table 4: Continued

		Game	Office	PDF	Photo
Access ratio	Read:write	7.16:1	12.05:1	13.93:1	1:1.30
	Inst.:data	1:3.29	1:1.67	1:3.06	1:5.55

Table 5: System configurations for our experiments

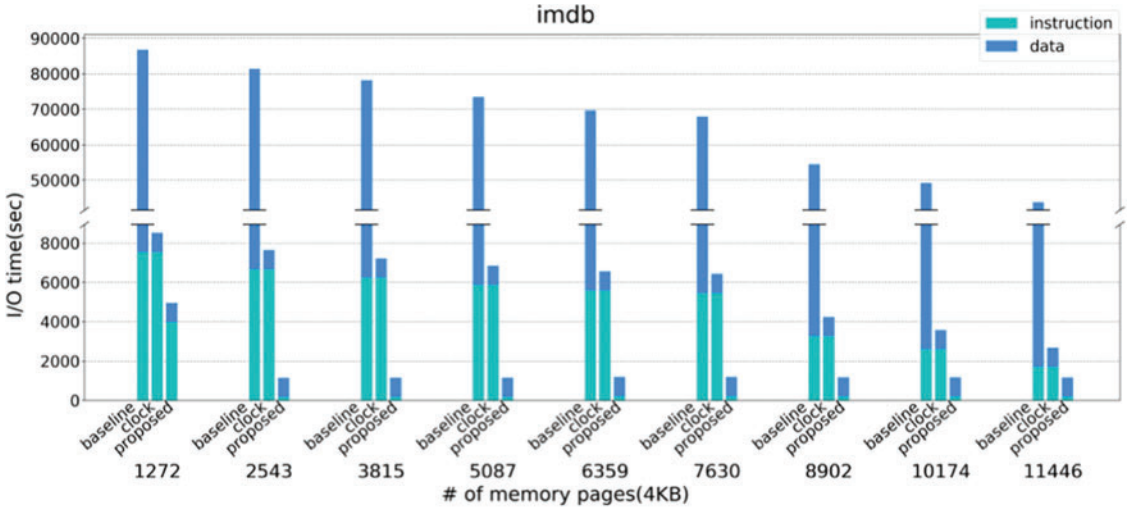
Resource type	Configurations
Processor core	Intel Core i7-11700 2.5 GHz 8-core processor
L1 instruction cache	32 KB (×8), 64-byte lines, 8-way set associative
L1 data cache	48 KB (×8), 64-byte lines, 12-way set associative
L2 cache	512 KB (×8), 64-byte lines, 8-way set associative
Last-level cache	16 MB, 64-byte lines, 16-way set associative
Main memory	8 GB DDR4 (read/write latency 50 ns)
NVRAM	8 GB PCM (read 100 ns, write 350 ns)
Secondary storage	1TB HDD (8 ms average access latency)

We developed a functional simulator that has the ability to evaluate the effectiveness of memory hierarchies when the page eviction policy and the parameters of memory/storage media are given. Specifically, our simulator replays memory access traces consisting of a series of logical page numbers and access types. During the simulations, if the requested page is not in memory, we simulate I/O activities based on the performance characteristics of each storage type. For hard disk drives (HDD), we use the parameters of Toshiba DT01ACA1, of which the read/write access latency is 8 milliseconds. For NVRAM, we use the parameters of PCM (Phase-Change Memory), of which the read and write latencies are 100 nanoseconds and 350 nanoseconds, respectively [9,32]. PCM stores data based on two phases of a material called GST (germanium-antimony-tellurium), which provides a different resistance to the cell when current flows so that data can be distinguished [32]. We use PCM as it is one of the well-known NVRAM media that can be placed in front of slow storage for performance acceleration. The size of a page is set to 4 kilobytes as it is the most common size used in modern operating systems such as Linux.

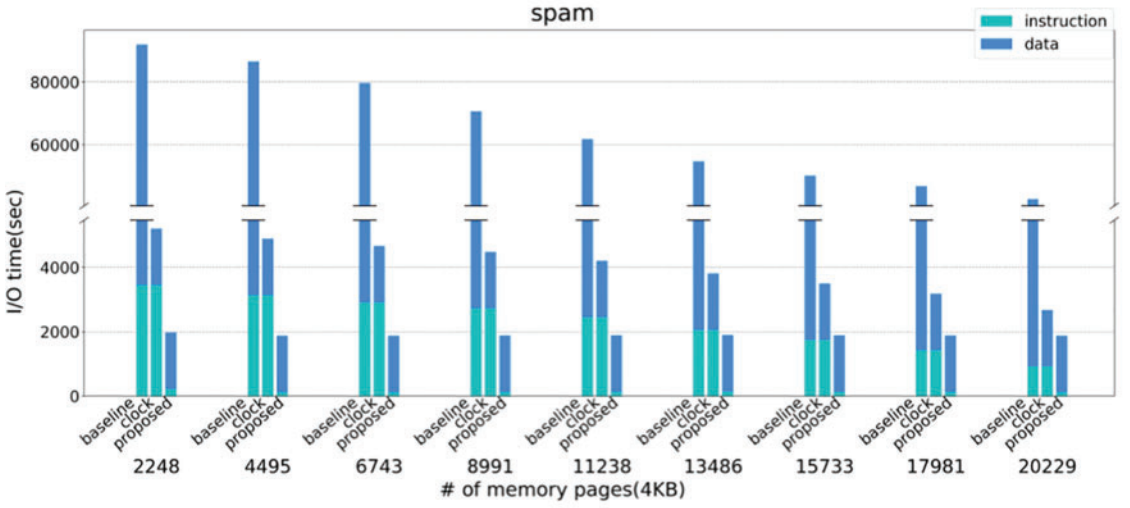
We compare our policy with the CLOCK policy [12] under the same NVRAM architecture. To see the effect of adopting NVRAM hardware itself rather than using judicious memory management policies, we also evaluate the performance of the system that does not use NVRAM, which we call Baseline.

Fig. 8 shows the total I/O time of the proposed scheme, CLOCK, and Baseline as the number of memory pages is varied. Since our memory access traces are collected on a sampling basis, the collected footprint is smaller than the actual memory footprint. Hence, the memory size used in our simulations should be much smaller than the actual system environment. Note that the x-axis in Fig. 8 is the DRAM memory size for each workload we simulate. The NVRAM size used in our experiments was set to the entire footprint size of the deep learning workload, similar to the setting of the swap memory size in desktops. As we see from this figure, the proposed policy performs consistently better than CLOCK regardless of the memory capacity for all workloads. In particular, the effectiveness apparently appears when the number of memory pages is relatively small. This is because our policy

manages the limited memory capacity more efficiently to reduce the total I/O time. That is, our policy considers the memory access characteristics of deep learning in terms of instruction and data access, and also takes into account the different I/O costs of NVRAM and secondary storage. By so doing, the proposed policy improves the total I/O time by 64.3% on average and up to 89.5% in comparison with CLOCK. When compared to Baseline, the performance improvement of the proposed policy is 97.6% on average and up to 99.2%.

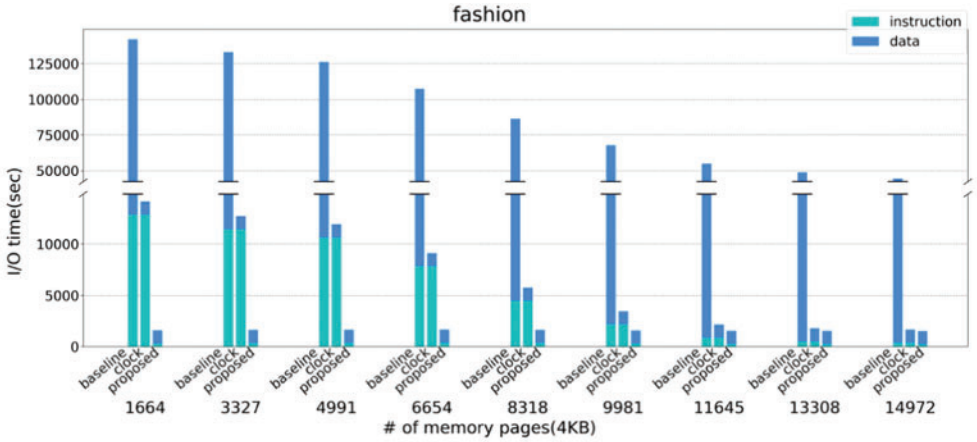


(a) IMDB (Internet Movie DataBase)

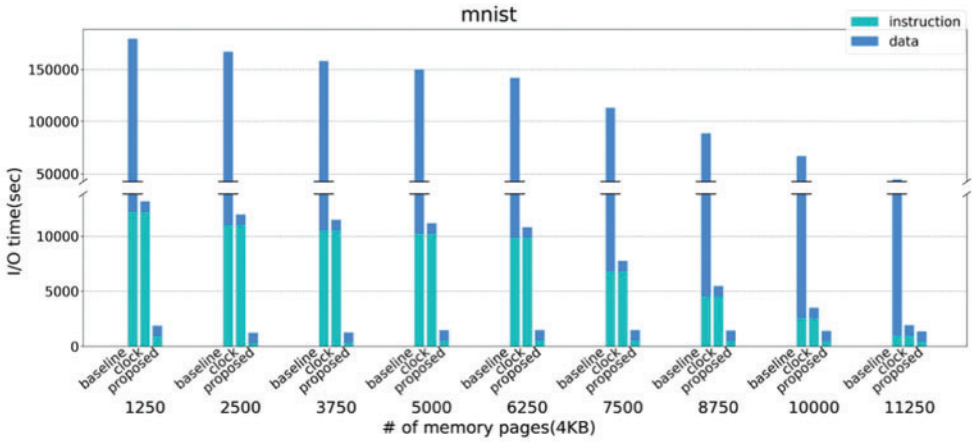


(b) Spam Detection

Figure 8: (Continued)



(c) Fashion MNIST (Modified National Institute of Standards and Technology database)



(d) MNIST

Figure 8: Comparison of baseline, CLOCK, and the proposed schemes

In Fig. 8, we separately plot the I/O time for instruction and data accesses. The two sources of performance improvement in this paper are the adoption of new hardware architecture and new software management. When comparing CLOCK with Baseline, we can see the improvement by adopting the hardware architecture, where data accesses benefit significantly. Note that the Baseline graph has been truncated because the gap between the numbers is too large. When comparing the proposed policy with CLOCK under the same architecture, we can see the improvement in software management, where instruction access gains significantly. Specifically, the reduced I/O time of instruction access is 88.1% on average compared to CLOCK. This is because our policy preserves instruction pages as much as possible by assigning high priorities in accordance with their I/O cost, whereas CLOCK does not consider the different access costs of NVRAM and secondary storage by giving the same priorities. Finally, when we compare our policy with Baseline, we can see the improvement by adopting both hardware architecture and software management. As we see, our policy improves the I/O time of data access as well as instruction access significantly. The reason is that we employ NVRAM basically for accelerating data access, but instruction access is also enhanced due

to judicious software management. The improvement of the proposed policy against the Baseline is 88.1% and 98.1% for the I/O time of instruction and data accesses, respectively.

6.2 Discussions

In this subsection, we will discuss the impact of our architecture on real system environments that execute deep learning workloads with a footprint of several gigabytes based on our preliminary simulation results. In Section 2, we showed that the size of memory required for the training phase of deep learning is 2x to 8x that of traditional workloads for executing a similar ratio of instructions. This implies that the system needs to allocate large memory size to deep learning workloads while training is performed to avoid memory-thrashing situations. Suppose that there are 8 GB of DRAM in the system that can be allocated to user processes. While the training phase of deep learning, the system temporarily requires at least an additional 8 GB in order to handle a large data set. In this situation, instead of increasing the DRAM size to 16 GB, let us consider our architecture equipped with 8 GB of NVRAM. Note that the typical size of a swap device in virtual memory systems is 1x to 2x the size of the main memory. In this paper, as the basic swap storage already exists, we set the size of NVRAM to the same as that of the main memory.

We can expect two advantages from this architecture. First, unlike the main memory system, we can use more complicated management policies in our NVRAM layer. That is, in the proposed architecture, since NVRAM is located below the main memory layer, it is possible to design management policies that utilize the characteristics of data access more precisely. In the main memory system, the reference history of each data is simply monitored by 1-bit information, allowing only simple algorithm design. For example, the working of the CLOCK algorithm is based on the binary information of the reference bit indicating whether the data has been recently used or not. In contrast, in the NVRAM layer of the proposed architecture, it is possible to design sophisticated algorithms using data request characteristics (e.g., access time, frequency) and region information (e.g., code, data). Since our purpose is to prevent memory thrashing during deep learning training, only the data area is absorbed by NVRAM. Also, we can utilize the full information of access time and frequency in managing NVRAM space. In summary, unlike the main memory system, which has many restrictions on management, it is possible to manage the NVRAM area more tailored to our purpose, so we can expect better results by adding the same size of NVRAM instead of DRAM.

Second, by adopting the proposed NVRAM-added architecture, we can expect the energy-saving of the system greatly. The energy consumption of a memory system consists of "active energy" consumed during read/write operations and "static energy" consumed regardless of any memory operation. It has been reported that static energy accounts for the majority of energy consumption in memory systems and that energy consumption increases proportionally to the size of the DRAM used [7,8]. To quantify the effect of our architecture with respect to energy-saving, we model the energy consumed in DRAM and NVRAM, respectively, as $Energy_{\mathrm{DRAM}}$ and $Energy_{\mathrm{NVRAM}}$. The DRAM energy consumption $Energy_{\mathrm{DRAM}}$ is the sum of static energy $Energy_{\mathrm{DRAM\_static}}$ and active energy $Energy_{\mathrm{DRAM\_active}}$, that is

$$Energy_{\mathrm{DRAM}} = Energy_{\mathrm{DRAM\_}static} + Energy_{\mathrm{DRAM\_}active} \tag{2}$$

Static energy $Energy_{\mathrm{DRAM\_static}}$ is the energy consumed consistently irrespective of any operations in DRAM memory, which can be calculated as

$$Energy_{\mathrm{DRAM\_}static} = Static\_Power\ \ (W/GB) * Size\_DRAM\ \ (GB) * Exec\_Time\ \ (s) \tag{3}$$

where *Static_Power* is the power consumption of DRAM per capacity regardless of read/write operations, *Size_DRAM* is the size of DRAM, and *Exec_Time* is the total running time of the system. The active energy $Energy_{\text{DRAM\_active}}$ is the energy consumed while read/write operations are performed, which can be modeled as

$$Energy_{\text{DRAM\_active}} = Energy\_DRAM_{read} * Num\_DRAM_{read} + Energy\_DRAM_{write} * Num\_DRAM_{write} \quad (4)$$

where Num_DRAM_{read} and Num_DRAM_{write} are the number of memory read and write operations, respectively, and $Energy\_DRAM_{read}$ and $Energy\_DRAM_{write}$ are the read and write energy consumption for the unit access size of DRAM, respectively. The NVRAM energy consumption $Energy_{\text{NVRAM}}$ is calculated as

$$Energy_{\text{NVRAM}} = Energy\_NVRAM_{read} * Num\_NVRAM_{read} + Energy\_NVRAM_{write} * Num\_NVRAM_{write} \quad (5)$$

where Num_NVRAM_{read} and Num_NVRAM_{write} are the number of NVRAM read and write operations, respectively, and $Energy\_NVRAM_{read}$ and $Energy\_NVRAM_{write}$ are the read and write energy consumption for the unit access size of NVRAM, respectively. Note that we do not consider the static energy consumption of NVRAM as it is non-volatile, and thus does not need refresh operations.

Based on this energy model, we simulate the energy consumption of the memory system with the increased DRAM size and our NVRAM-added architecture. In this simulation, the read/write energy of DRAM is set to 0.1 (nJ/bit) and the static power of DRAM is set to 1 (W/GB) following previous studies [7,9]. The read and write energy of NVRAM is set to 0.2 (nJ/bit) and 1.0 (nJ/bit), respectively [7,9]. The energy consumption depends on how frequently read/write operations are performed. Our simulation shows that the NVRAM-added architecture can reduce the energy consumption of the memory system by 42.4% to 46.6% depending on the frequency of memory access in each workload. This again confirms that the static energy of DRAM is significant in memory energy consumption.

7 Related Work

Using NVRAM in various memory hierarchies of computer systems has been attempted. As NVRAM is byte-addressable like DRAM or SRAM (Static Random Access Memory), some studies focus on using NVRAM for cache memory or main memory layers. Also, since NVRAM is non-volatile, there are studies to utilize NVRAM as a fast storage medium. However, NVRAM has some weak features for use at the memory tier (i.e., limited write endurance and slow writes) and storage tier (i.e., high cost per capacity). Therefore, research on NVRAM has attempted to hide these limitations of NVRAM while improving the energy and performance characteristics of memory and storage systems.

Some studies focus on the non-volatile nature of NVRAM when it is adopted for the main memory layer. Unlike existing systems that consider in-memory data to be ephemeral, these studies exploit the persistency of in-memory data. That is, when the main memory becomes non-volatile, the cache lines become the boundary layer between temporary and permanent devices. This means that write atomicity is guaranteed at the granularity of a cache line. By considering this, Cho et al. conduct studies for persistent in-memory data structures [33]. Specifically, they propose the FBR-tree (Failure-atomic Byte-addressable R-tree) data structure to guarantee consistency against crash situations.

Studies focusing on on-chip cache architectures aim to use NVRAM as a replacement for SRAM. Talebi et al. present an on-chip cache architecture that makes use of NVRAM [34]. Specifically, they

present an eviction policy for the NVRAM cache to enhance the robustness of NVRAM media against failure conditions.

Rucker et al. make use of NVRAM as an L2 or L4 CPU cache with low area and power requirements compared to SRAM [35]. Because NVRAM has endurance concerns, they assess the effectiveness of cache management policies in terms of the wear-out issue of NVRAM.

Fan et al. use NVRAM as the main memory and focus on the fact that dirty pages (i.e., updated pages) can be kept longer without the urgent need to be flushed to storage as the main memory becomes non-volatile [36]. Specifically, they present the Hierarchical Adaptive Replacement Cache (H-ARC) policy that considers the status of memory pages based on clean/dirty as well as recency/frequency characteristics.

Wang et al. suggest an NVRAM-based processing-in-memory accelerator for the training of neural networks [37]. They try to balance memory density and computation flexibility, thereby improving not only performance but also energy and area efficiency.

Jin et al. propose an eviction policy for cache memory when hybrid DRAM and NVRAM main memory architectures are used [38]. They argue that the miss penalty in memory access is more important than the hit ratio in their architecture and present Miss-penalty Aware LRU (MALRU) to improve overall performance.

Some studies make use of NVRAM as the swap device of virtual memory systems. Liu et al. adopt NVRAM as the swap partition of mobile devices to improve the performance of flash memory swaps [39]. They also consider the wear leveling of NVRAM media by evenly distributing writes.

Volos et al. present an interface for programming in NVRAM memory that provides the creation and management of data without inconsistency risks under failure situations [40]. Specifically, their interface allows programmers to define persistent data structures based on given primitives that guarantee consistency through transaction management functions.

Hadizadeh et al. constitute a storage-accelerating architecture hierarchically with NVRAM and flash memory [41]. They allocate clean and dirty pages to NVRAM and flash caches based on vulnerability aspects and generate ECCs (Error-Correction Codes) for dirty pages dynamically for improving reliability.

The CacheLib project supports hybrid storage cache architectures consisting of DRAM and NVRAM [42]. Specifically, they provide transparent caching to users by providing cache allocation interfaces. As NVRAM has limited write endurance, their cache allocator supports a pluggable eviction policy that can reject items if necessary.

8 Conclusions

As the data size of deep learning continues to grow, it is difficult to accommodate the full data set of deep learning workloads in memory, resulting in serious performance degradation. To cope with this situation, this paper performed extensive characterization studies for deep learning memory accesses. Specifically, we collected memory access traces of four deep learning workloads and analyzed them with respect to access types, operations, access bias, and re-access estimation. Our observations from this analysis can be summarized as follows. First, when comparing instruction and data accesses, instruction access accounts for a little portion of memory accesses in deep learning workloads, which is quite different from traditional workloads. Specifically, instruction access accounts for only 1%–3.3% of deep learning workloads whereas 15%–30% of traditional workloads. This implies that data access

will be the bottleneck of memory systems in deep learning workloads. Second, when comparing read and write accesses, write access accounts for 64%–80% of deep learning, which is also different from traditional workloads. Third, although write access accounts for the majority of memory accesses, it exhibits low access bias. Specifically, the Zipf parameter of write access is only 0.3. Fourth, in predicting the re-access of memory pages, recency ranking is important in read access, whereas frequency ranking is necessary for the accurate estimation of write access. Based on these observations, we introduced an NVRAM-accelerated memory architecture for deep learning workloads and presented a new memory management policy for this architecture. By considering the memory access characteristics of deep learning workloads, the proposed policy improves memory performance by 64.3% on average compared to the well-acknowledged CLOCK policy.

The research conducted in this paper is an early version of a model to cope with the explosive memory demand in deep learning training. In the future, we will consider a dynamic allocation of NVRAM for deep learning workloads executed in cloud environments. When various workloads coexist on physical machines in the cloud, it is challenging to optimally allocate NVRAM resources to each virtual machine over time to handle the memory pressure of deep learning workloads.

Funding Statement: This work was supported in part by the NRF (National Research Foundation of Korea) Grant (No. 2019R1A2C1009275) and by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2021-0-02068, Artificial Intelligence Innovation Hub).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Dargan, M. Kumar, M. R. Ayyagari and G. Kumar, "A survey of deep learning and its applications: A new paradigm to machine learning," *Archives of Computational Methods in Engineering*, vol. 27, no. 1, pp. 1071–1092, 2020.

[2] J. Li, N. Mirza, B. Rahat and D. Xiong, "Machine learning and credit ratings prediction in the age of fourth industrial revolution," *Technological Forecasting and Social Change*, vol. 161, no. 1, pp. 1–13, 2020.

[3] S. Idowu, D. Strüber and T. Berger, "Asset management in machine learning: State-of-research and state-of-practice," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–35, 2022.

[4] H. Fujiyoshi, T. Hirakawa and T. Yamashita, "Deep learning-based image recognition for autonomous driving," *IATSS Research*, vol. 43, no. 4, pp. 244–252, 2019.

[5] J. Xiong, D. Yu, S. Liu, L. Shu, X. Wang *et al.,* "A review of plant phenotypic image recognition technology based on deep learning," *Electronics*, vol. 10, no. 1, pp. 1–19, 2021.

[6] I. H. Sarker, M. M. Hoque, M. K. Uddin and T. Alsanoosy, "Mobile data science and intelligent apps: Concepts, AI-based modeling and research directions," *Mobile Networks and Applications*, vol. 26, no. 1, pp. 285–303, 2021.

[7] E. Lee, H. Kang, H. Bahn and K. G. Shin, "Eliminating periodic flush overhead of file I/O with non-volatile buffer cache," *IEEE Transactions on Computers*, vol. 65, no. 4, pp. 1145–1157, 2016.

[8] D. T. Nguyen, H. Kim, H. J. Lee and I. J. Chang, "An approximate memory architecture for a reduction of refresh power consumption in deep learning applications," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, Florence, Italy, pp. 1–5, 2018.

[9] S. Yoo, Y. Jo and H. Bahn, "Integrated scheduling of real-time and interactive tasks for configurable industrial systems," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 1, pp. 631–641, 2022.

[10] H. Kim, M. Ryu and U. Ramachandran, "What is a good buffer cache replacement scheme for mobile flash storage," in *Proc. ACM SIGMETRICS Conf.*, London, UK, pp. 235–246, 2012.

[11] S. Kang, S. Park, H. Jung, H. Shim and J. Cha, "Performance trade-offs in using NVRAM write buffer for flash memory-based storage devices," *IEEE Transactions on Computers*, vol. 58, no. 6, pp. 744–758, 2009.
[12] F. J. Corbato, "A paging experiment with the multics system," in *Honor of Philip M. Morse*, 1st ed., Cambridge, MA, USA: The MIT Press, pp. 217–228, 1969.
[13] Tensorflow, https://www.tensorflow.org/
[14] Pytorch, https://pytorch.org/
[15] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," *ACM SIGPLAN Notices*, vol. 42, no. 6, pp. 89–100, 2007.
[16] C. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser *et al.,* "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. ACM SIGPLAN Notices*, Chicago, IL, USA, vol. 40, no. 6, pp. 190–200, 2005.
[17] Memory access traces, https://github.com/oslab-ewha/memtrace
[18] IMDB, https://ai.stanford.edu/~amaas/data/sentiment/
[19] Spam Collection Data Set, https://archive.ics.uci.edu/ml/datasets/sms+spam+collection
[20] Fashion MNIST, https://github.com/zalandoresearch/fashion-mnist
[21] MNIST, http://yann.lecun.com/exdb/mnist/
[22] G. K. Zipf, *Human Behavior and the Principle of Least Effort*, 1st ed., New York, NY, USA: Addison-Wesley Press, 1949.
[23] L. Breslau, P. Cao, L. Fan, G. Phillips and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. IEEE INFOCOM Conf.*, New York, NY, USA, pp. 126–134, 1999.
[24] U. Oh, S. Lim and H. Bahn, "Channel reordering and prefetching schemes for efficient IPTV channel navigation," *IEEE Transactions on Consumer Electronics*, vol. 56, no. 2, pp. 483–487, 2010.
[25] H. Bahn, S. Noh, S. Min and K. Koh, "Using full reference history for efficient document replacement in web caches," in *Proc. USENIX Symp. on Internet Technol. Systems (USITS)*, Boulder, CO, USA, pp. 187–196, 1999.
[26] D. Liu, K. Zhong, X. Zhu, Y. Li, L. Long *et al.,* "Non-volatile memory based page swapping for building high-performance mobile devices," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1918–1931, 2017.
[27] Intel Optane™ Persistent Memory, https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/overview.html
[28] E. Lee, H. Bahn, S. Yoo and S. H. Noh, "Empirical study of NVM storage: An operating system's perspective and implications," in *Proc. IEEE MASCOTS Conf.*, Paris, France, pp. 405–410, 2014.
[29] E. Cheshmikhani, H. Farbeh, S. Miremadi and H. Asadi, "TA-LRW: A replacement policy for error rate reduction in STT-MRAM caches," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 455–470, 2019.
[30] S. Bansal and D. S., Modha, "CAR: Clock with adaptive replacement," in *Proc. USENIX Conf. on File and Storage Technol. (FAST)*, San Francisco, CA, USA, pp. 1–15, 2004.
[31] T. Johnson and D. Shasha, "2Q: A low overhead high performance buffer management replacement algorithm," in *Proc. VLDB Conf.*, Santiago, Chile, pp. 439–450, 1994.
[32] V. Deep and T. Elarabi, "Write latency reduction techniques of state-of-the-art phase change memory," in *Proc. European Modelling Symp. (EMS)*, Pisa, Italy, pp. 213–217, 2016.
[33] S. Cho, W. Kim, S. Oh, C. Kim, K. Koh *et al.,* "Failure-atomic byte-addressable R-tree for persistent memory," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 601–614, 2021.
[34] M. Talebi, A. Salahvarzi, A. M. Monazzah, K. Skadron and M. Fazeli, "ROCKY: A robust hybrid on-chip memory kit for the processors with STT-MRAM cache technology," *IEEE Transactions on Computers*, vol. 70, no. 12, pp. 2198–2210, 2021.
[35] A. Rucker, A. Bartolo and C. Chute, "NVM cache with predictive allocation," 2017. [Online]. Available: https://stanford.edu/~bartolo/assets/nvm-cache.pdf
[36] Z. Fan, D. H. C. Du and D. Voigt, "H-ARC: A non-volatile memory based cache policy for solid state drives," in *Proc. IEEE MSST Conf.*, Santa Clara, CA, USA, pp. 1–11, 2014.

[37] H. Wang, Y. Zhao, C. Li, Y. Wang and Y. Lin, "A new MRAM-based process in-memory accelerator for efficient neural network training with floating point precision," in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, Seville, Spain, pp. 1–5, 2020.
[38] H. Jin, D. Chen, H. Liu, X. Liao, R. Guo *et al.,* "Miss penalty aware cache replacement for hybrid memory systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4669–4682, 2020.
[39] S. Liu, K. Pattabiraman, T. Moscibroda and B. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," *ACM SIGPLAN Notices*, vol. 46, no. 3, pp. 213–224, 2011.
[40] H. Volos, A. Tack and M. Swift, "Mnemosyne: Lightweight persistent memory," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 1, pp. 91–104, 2011.
[41] M. Hadizadeh, E. Cheshmikhani and H. Asadi, "STAIR: High reliable STT-MRAM aware multi-level I/O cache architecture by adaptive ECC allocation," in *Proc. IEEE DATE Conf.*, Grenoble, France, pp. 1484–1489, 2020.
[42] The CacheLib project, https://cachelib.org/docs/Cache_Library_Architecture_Guide/hybrid_cache/