electronics

MDPI

Article

ML-CLOCK: Efficient Page Cache Algorithm Based on Perceptron-Based Neural Network

Minseon Cho and Donghyun Kang *

Department of Computer Engineering, Changwon National University, Changwon 51140, Korea; seonjm@changwon.ac.kr

* Correspondence: donghyun@changwon.ac.kr

Abstract: Today, research trends clearly confirm the fact that machine learning technologies open up new opportunities in various computing environments, such as Internet of Things, mobile, and enterprise. Unfortunately, the prior efforts rarely focused on designing system-level input/output stacks (e.g., page cache, file system, block input/output, and storage devices). In this paper, we propose a new page replacement algorithm, called *ML-CLOCK*, that embeds single-layer perceptron neural network algorithms to enable an *intelligent eviction policy*. In addition, *ML-CLOCK* employs *preference rules* that consider the features of the underlying storage media (e.g., asymmetric read and write costs and efficient write patterns). For evaluation, we implemented a prototype of *ML-CLOCK* based on trace-driven simulation and compared it with the traditional four replacement algorithms and one flash-friendly algorithm. Our experimental results on the trace-driven environments clearly confirm that *ML-CLOCK* can improve the hit ratio by up to 72% and reduces the elapsed time by up to 2.16x compared with least frequently used replacement algorithms.

Keywords: clean-first eviction; learning and prediction; page replacement algorithm; single-layer perceptron neural network; sequential write pattern

Citation: Cho, M.; Kang, D. ML-CLOCK: Efficient Page Cache Algorithm Based on Perceptron-Based Neural Network. *Electronics* **2021**, *10*, 2503. https://doi.org/10.3390/electronics10202503

Academic Editor: George A. Tsihrintzis

Received: 24 September 2021
Accepted: 11 October 2021
Published: 14 October 2021

1. Introduction

Many algorithms on the page cache layer rely on the workload's localities (i.e., temporal and spatial locality) because locality is a key driving factor behind providing opportunities for better hit ratios [1–5]. For example, CLOCK-PRO was designed to take the temporal locality by classifying the type of pages on CLOCK into hot and cold based on reuse distance (i.e., recency). Moreover, in the page cache layer, there were efforts to generate I/O patterns in a sequential order that is one of the flash-friendly approaches [6,7]. Fortunately, the hit ratio of some proposed algorithms had shown good results compared with traditional algorithms (e.g., LRU and LFU). However, it is a well-known fact that file systems have a large amount of data and the number of re-accesses is very low and unpredictable; real-world workloads are quite diverse.

Meanwhile, it is strongly desirable for software I/O stacks (e.g., page cache, file system, block IO, and storage devices) to adequately adopt machine learning (ML), such as single-layer perceptron (SLP), multi-layer perceptron (MLP), convolutional neural networks (CNN), and long short-term memory networks (LSTM) [8–13]. Today, ML technologies have been gradually applied to I/O stacks because of three reasons. First, the I/O stacks include many parameters and configurations across layers, and they should be tuned for the performance and efficiency of system. Second, ML techniques can easily help to discover what parameters and configurations highly affect time and space saving. For example, to cope with the configurable parameters in I/O stacks, some researchers focused on file systems relative to the underlying storage layer with ML technologies [14]. They found and described crucial parameters and configurations for some file systems by varying workloads. Third, the overhead of ML technologies can be hidden or pipelined behind the storage time when data are carried from I/O stacks to the underlying storage media.

Recently, some researchers believed ML technologies provide us with hints to predict I/O patterns and open new opportunities to address challenges in cache replacement policy. For example, CACHEUS dynamically determines which replacement algorithm is used to evict a page by learning the patterns from workloads whenever the eviction operation is triggered [15]. Some researchers studied ML-aware cache algorithms along with state-of-the-art storage devices[16]. They tried to take potential benefits of multiple streams by learning I/O patterns issued to the underlying NVMe devices. Other studies focused on the mechanisms (e.g., write buffer and the SLC/MLC/TLC write mode [17,18]) inside the flash-based storage devices, and it achieved improvement in performance using ML technologies [19]. Unfortunately, most prior efforts have not focused on how to orchestrate ML technologies and I/O patterns in designing page replacement algorithms.

In this paper, we present a novel page replacement algorithm called machine learning-based CLOCK (*ML-CLOCK*). *ML-CLOCK* was designed to answer the following questions:

- Is it possible to discover a page, which will not be accessed in the future, on unexpected I/O patterns by taking the benefit of ML technology?
- Given a set of available information, such as the page's recency and frequency, which one should we consider more important?
- Is it always useful to evict a page without considerations of asymmetric read and write costs on the underlying storage devices?

To answer the above questions, we carefully designed *ML-CLOCK* based on the traditional CLOCK algorithm along with its basic rules. However, *ML-CLOCK* provides a different eviction policy from the traditional CLOCK in that it applies an intelligent approach. To select a victim page in the circular list, *ML-CLOCK* employs the perceptron neural network algorithm that is one of the most famous algorithms in ML technologies [8,20]. The perceptron algorithm in *ML-CLOCK* learns the I/O pattern of workloads on-the-fly and then performs the prediction to decide whether it is a victim page or not. In addition, *ML-CLOCK* considers the characteristics of flash-based storage devices; generally, the eviction for a clean page is more lightweight than a dirty one, and the sequential pattern is better than random ones. For evaluation, we have implemented *ML-CLOCK* to run on trace-driven experiments and compared it with the five replacement algorithms, such as FIFO, LRU, LFU, CLOCK, and Sp.Clock. Our evaluation results verified that the effect of applying the intelligent approach is very promising in the page replacement layer. In the best case, *ML-CLOCK* improves the cache hit ratio by up to 72% over the traditional replacement algorithm.

In the rest of this paper, we first describe a brief overview of the perceptron algorithm to understand *ML-CLOCK* (Section 2). Next, we will present the details of *ML-CLOCK* design with a pseudo-code (Section 3) Then, we will show our evaluation results performed on trace-driven environments (Section 4) and discuss related work (Section 5). Finally, we conclude the paper (Section 6).

2. Background

In this section, we discuss ML technologies, especially the neural network algorithms, in detail.

2.1. Neural Network-Based Learning Algorithm

Today, ML is a new software paradigm that is becoming increasingly popular; ML is a subfield of AI that can simulate human objects and intelligence using software technologies [21]. A lot of efforts for applying ML technologies further accelerated such changes [8–11]. One of the efforts is to design a learning algorithm that enhances the performance and functionality of the neural networks [9–11]. Over the years, diverse learning algorithms have been designed for the enhancement of neural network models.

Table 1 describes the characteristics of the popular neural network-based algorithms. The algorithms listed in Table 1 share the same goal in that the algorithms rapidly address the existing challenge that takes up to several days or times based on given information.

However, they have slightly different characteristics according to their given information and their modeling approach. As shown in Table 1, SLP shows lower overhead in terms of time and space compared with the other three learning algorithms; we will discuss SLP in the next paragraph [22]. To take the advantages of SLP, many researchers and engineers utilize SLP to control online learning [23–25] where learning takes place when desired data are received in a sequence one at a time. In other words, the online learning performs training and prediction for the data based on real-time events [26].

Table 1. Characteristics of the different neural network-based algorithms.

	SLP [8]	**MLP [9]**	**CNN [10]**	**LSTM [11]**
Time/Space Overhead	Low	High	High	High
Overfitting Probability	Low	High	High	High
Online Learning	✓	✗	✗	✗
Linear Binary Classification	✓	✗	✗	✗
Non-linear Classification/Regression	✗	✓	✓	✓
Image Classification/Regression	✗	✗	✓	✗
Sequence Classification/Regression	✗	✗	✗	✓

On the other hand, other algorithms mainly handle data with offline learning where data are learned once in batches; they require training data in advance before the prediction. Therefore, the prediction time of the offline learning method is faster than that of online learning because it never requires a real-time training step. Note that the offline-based algorithms are in a better position to train and predict for high computational complexity (e.g., non-linear, image, or sequence classification/regression) compared with SLP. The reason behind this is that the algorithms have more layers for training and inference compared with SLP. Therefore, some researchers focused on offline learning to take its different benefits from online ones [16]. Unfortunately, the algorithms suffer from overhead that is accumulated as increasing the number of layers [27,28]. For example, CNN model-based ResNet reported that it commonly has 26 million weights, and it takes time for one prediction at about 0.554 s; it is never a short period of time because time is accumulated every epoch run [28]. In addition, large scale layers can result in the possibility of an overfitting problem [28].

2.2. Perceptron-Based Algorithm

Now, we will describe the perceptron-based neural network algorithm in detail in order to evaluate the algorithm *ML-CLOCK*. To understand the perceptron-based algorithm, we first describe artificial neural networks (ANNs) that are inspired by the biological neural cells of humans [9]. ANN provides the abstraction for each biological neural cell to process given information, and we call it a *node*. Each node is connected to each other for communication across layers. A layer is another abstraction that receives one or more input values and transforms them for passing the output value to the next layer. The perceptron-based learning models can be composed of two default layers for one input and output and one or more hidden layers. According to the number of layers, the perceptron-based algorithms are classified into a single-layer perceptron (SLP) and a multi-layer perceptron (MLP). It is well known that SLP is one of the simple neural networks, and it is enough to predict simple classification. In SLP, for prediction, the input layer feeds a set of weighted input values and forwards the values directly to the output layer (i.e., SLP only consists of two default layers). In the rest of the paper, we refer to the output layer as an *activation function*. The activation function can be expressed as Equation (1) [8]:

$$f(x_1, \cdots, x_N) = \begin{cases} 0, & b + \sum_{i=1}^{N} x_i \cdot w_i < 0 \\ 1, & b + \sum_{i=1}^{N} x_i \cdot w_i \geq 0 \end{cases} \quad (1)$$

where b represents a bias, and x_i and w_i denote an input value and the corresponding weight value, respectively. As observed in Equation (1), the activation function renders the output value in a binary classification manner that indicates 0 or 1 by calculating the sum of the weighted input values and a bias value. The role of the bias value is to shift the decision boundary so that it does not to depend on one specific input value. On the other hand, MLP performs training in the same manner as SLP, but it has one or more extra hidden layers on two default layers [9]. MLP also has the features of a fully connected layer where perceptrons are connected to each other. Therefore, it can be used for solving more complex problems (i.e., non-linear classification or regression) based on offline learning. However, as mentioned before, MLP also reveals that performance and space overheads increase the number of layers because the total number of parameters that has to be handled grows significantly. Meanwhile, The CNN and LSTM models can be considered as subfields of MLP in that they are composed of multiple layers. In general, the CNN model is used for image classification by using a convolution filter that transforms features to feature maps [10]. On the other hand, since the LSTM model can handle the ordering characteristics of incoming data, it is widely utilized for sequence classification [11,29,30]. However, in this paper, we focus on SLP in that it can perform training and prediction on-the-fly with lightweight overhead.

3. ML-CLOCK

In this section, we describe our solution, called machine learning-based CLOCK (*ML-CLOCK*), in detail. As mentioned before, *ML-CLOCK* is not the first work on applying the concept of ML technologies into the page cache layer. However, to the best of our knowledge, it is the first work that performs online training and prediction based on the information of both recency and frequency in the page cache algorithm. We mainly designed *ML-CLOCK* to satisfy two design requirements:

- Requirement 1: It reduces the number of access to the underlying storage media.
- Requirement 2: It orchestrates functionalities of machine learning (ML) to enable the *intelligent eviction policy*.

To achieve the first requirement, *ML-CLOCK* follows the main rules of the traditional CLOCK algorithm that is well known for maintaining a high hit ratio; CLOCK scans the circular list according to the `hand` of CLOCK either to make a victim page for free space or to provide a *second chance* to a page for temporal locality. Unlike the traditional CLOCK, *ML-CLOCK* employs two hands; one is the Clean-hand (`C-hand`), and the other is the Dirty-hand (`D-hand`). *ML-CLOCK* simultaneously moves two hands to find victim candidates according to a sequence of the corresponding hand. In order to meet the second requirement, *ML-CLOCK* does not allow some rules that are responsible for the eviction policy of the CLOCK algorithm. For the second requirement, *ML-CLOCK* intelligently embeds a single-layer perceptron (SLP) neural network algorithm.

3.1. Learning and Prediction Model

When determining a victim page to reclaim free space on the page cache layer, temporal locality-based replacement algorithms commonly utilize either the characteristic of the page's recency or frequency; recency indicates the distance between two consecutive accesses at the same page, and frequency denotes the number of accesses to the same page. According to the recency or frequency principle, the position of a page on the replacement algorithm can be moved up to the high level (i.e., priority) whenever access is taking place on the same page. In other words, pages placed on higher levels imply that they might be re-accessed in the near future more than pages at lower levels. Unfortunately, it is very difficult to make a replacement algorithm that embraces both recency and frequency in that they have different priorities for the eviction candidates in each other under the same situation. In order to solve this trouble, we designed an *ML-CLOCK* that utilizes the mechanism of SLP for determining which one has more meaningful effects on the eviction

policy. In *ML-CLOCK*, learning and prediction are regarded as complex steps. The steps consist of a couple of extra properties and operations, such as those stated below:

- For prediction, there are three of input values: *reuse-distance* for recency, *reference count* for frequency, and *bias* as mentioned earlier.
- *Prediction operation* is triggered either to find a victim page or to make decisions for the learning operation.
- *Learning operation* is responsible for calculating and updating weight values for the input values.

First of all, let us describe a prediction operation. For learning and prediction, *ML-CLOCK* calculates a *predicted value* with Equation (2):

$$f_{predict}(x_d, x_c) = \begin{cases} 0, & x_d \times w_d + x_c \times w_c + 1 \times w_b < 0 \\ 1, & x_d \times w_d + x_c \times w_c + 1 \times w_b \geq 0 \end{cases} \tag{2}$$

where w_d, w_c, and w_b represent the weight value of a reuse-distance, reference count, and bias of the page, respectively. Note that the weight values of w_d and w_c represent which one has more meaningful effects on the eviction policy. Moreover, x_d and x_c indicate the inputs value of reuse-distance and reference count, respectively. Unfortunately, since x_d is orders of magnitude higher than x_c, it can result in mis-prediction. Therefore, we recalculate x_d with Equation (3) for scaling down.

$$x_d = \frac{current\ timestmap - timestamp\ of\ the\ last\ access}{size\ of\ circular\ list} \tag{3}$$

As mentioned before, SLP is mainly used for binary classification; thus, the *predicted value* calculated by Equation (2) can take either 0 or 1. In *ML-CLOCK*, we define the predicted value as follows:

- Predicted value 0: It has a low possibility that access occurs in a short period of time.
- Predicted value 1: It provides a second chance because of an opportunity for access in the near future.

Note that the predicted value is also used to make a decision for calculating weight values for the input values, respectively. As mentioned in Section 2.2, each *weighted input value* in SLP is calculated by multiplying the input value by the corresponding weight value. The calculation for a learning operation can be expressed in Equation (4):

$$\begin{aligned} w_d &\leftarrow w_d + lr \times x_d \times (v_{expect} - v_{predict}) \\ w_c &\leftarrow w_c + lr \times x_c \times (v_{expect} - v_{predict}) \\ w_b &\leftarrow w_b + lr \times 1 \times (v_{expect} - v_{predict}) \end{aligned} \tag{4}$$

The equation contains the same variables (i.e., x_d, x_c, w_d, w_c, and w_b) as in the previous equation. The learning rate (i.e., lr) controls how much to change this model so as to respond to the estimated error each time when the weights are updated. $v_{predict}$ indicates the result value after calculating Equation (2) (i.e., *predicted value*), and v_{expect} means the correct answer for this prediction. Therefore, $(v_{expect} - v_{predict})$ determines whether there is a need for the update of weight values in order to render it up to date. If $v_{predict}$ matches v_{expect}, the function does nothing because it results in 0 except for the existing weight values. Otherwise, each weight value is updated for the next prediction operation based on Equation (4).

3.2. ML-CLOCK Algorithm

Now, let us describe the page replacement procedure of *ML-CLOCK* along with the learning and prediction operations. *ML-CLOCK* adopts different three policies compared with the traditional CLOCK algorithm in order to decide a victim page.

First, *ML-CLOCK* takes multiple hands, `C-hand` and `D-hand`, and it allows independent scanning of each hand to find victim candidates. When there is no room in the circular list for a new page, `C-hand` scans just clean pages and stops the scanning when it meets a clean page with zero reference bits. `D-hand` is performed in the same manner except for two differences: `D-hand` targets only dirty pages in the circular list, and it scans the pages in a sequential order (e.g., logical block address) for generating the pattern of sequential writes. We call the pages by pointing either `C-hand` or `D-hand` after scanning as the **C-candidate** and **D-candidate**, respectively. Second, *ML-CLOCK* conducts learning and prediction operations in order to correctly decide which candidate is evicted. To minimize wrong predictions, *ML-CLOCK* additionally uses a *ghost queue* that keeps information of pages evicted from the circular list; we will explain later the usage of the ghost queue in detail. Third, *ML-CLOCK* adopts a *clean-first eviction* policy where a clean page takes precedence in terms of page eviction. The reason behind this is that issuing a read operation to the underlying storage devices is much more efficient than a write operation due to asymmetric reads and write cost.

In order achieve better understanding, we present the pseudo-code for the *ML-CLOCK* algorithm (see Algorithm 1). As shown in Algorithm 1, a learning operation takes place at three points: Line 3, Line 16, and Line 21. A prediction operation is triggered after finding victim candidates in order to decide which candidate is evicted (Line 13). When the access occurs on the page placed in either the circular list (Line 2) or the ghost queue (Line 16), *ML-CLOCK* triggers a learning operation to update weight values for the next prediction. At this time, *ML-CLOCK* sets the correct answer value to the parameter that is connected to v_{expect} of Equation (4) and transfers the accessed page in order to calculate its predicted value. If $v_{predict}$ calculated by Equation (2) is matched to the transferred answer value, we determine that the prediction operation produces a correct answer. Otherwise, the weight values are recalibrated for the next prediction operation according to Equation (4). Meanwhile, even if there is no room in the ghost queue, *ML-CLOCK* triggers a learning operation by setting the parameter of v_{expect} as zero (Line 21). This is because it helps to produce a learning hint that states that a page will not be reused.

In contrast, in the case of a cache miss where there is no page for I/O requests on the circular list, we insert a new page in the list (Line 28); *ML-CLOCK* repeats this operation for handling a cache miss until there is no free space. When the circular list is full, *ML-CLOCK* starts to scan pages at the location of the `C-hand` to find a victim candidate (i.e., C-candidate) among clean pages (Line 8). At the same time, *ML-CLOCK* also moves `D-hand` to find D-candidate in a sequential order (Line 11); in order to efficiently handle ordering among dirty pages, *ML-CLOCK* keeps an auxiliary list where dirty pages on the circular list are mapped in a sorted order (i.e., logical block address). After finding two candidates, *ML-CLOCK* decides which candidate is evicted by triggering prediction operations (Line 13). The key operation in this step is to predict whether each candidate will be reused or not in near future. Finally, a victim page would be selected according to the preference rules listed in Table 2. As mentioned before, since *ML-CLOCK* allows a clean-first eviction, it has preference rules to decide a victim page (see Table 2). According to the rules, *ML-CLOCK* first evicts the C-candidate page in the case where the results of the prediction about two candidates are the same. On the other hand, *ML-CLOCK* gives a second chance to the D-candidate by setting its reference bit as one. The reason behind this is that *ML-CLOCK* respects the potential possibility coming from its prediction.

Algorithm 1 ML-CLOCK

```
1:  Input: page: page on I/O request, Clk: Clock,
           GhostQ: Ghost Queue.
2:  if Clk.findPage(page) == True then
3:     Learning(page, 1)
4:  else
5:     // cache miss
6:     if Clk.isFull() == TRUE then
7:        /* scan clean pages at the C-hand's location */
8:        C-candi = Clk.findCleanCandidate()
9:        /* scan dirty pages at the D-hand's location */
10:       /* in a sequential order (i.e., LBA) */
11:       D-candi = Clk.findDirtyCandidate(D-hand)
12:       /* decide which page is evicted using prediction */
13:       victim = Prediction(C − candi, D − candi)
14:       if GhostQ.findPage(victim) == True then
15:          /* ready to promote a page to Clk */
16:          Learning(page, 1)
17:          GhostQ.deletePage(victim)
18:       else
19:          if GhostQ.isFull() == TRUE then
20:             /* evict a page on GhostQ with learning */
21:             Learning(GhostQ.tail, 0)
22:             GhostQ.deletePage(GhostQ.tail)
23:          end if
24:          GhostQ.insertPage(victim)
25:       end if
26:       Clk.replacePage(victim, page)
27:    else
28:       Clk.insertPage(page)
29:    end if
30: end if
```

Table 2. The preference rules to decide a victim page. We denote ✗ and ✓ as the predicted value 0 and 1, respectively.

Predicted Value (C-Candidate)	Predicted Value (D-Candidate)	Victim Page
✗	✗	C-candidate
✗	✓	C-candidate
✓	✗	D-candidate
✓	✓	C-candidate

Note that, unlike the traditional CLOCK, *ML-CLOCK* additionally maintains a ghost queue for which its size is the same as the circular list. This queue is used to enhance prediction because it provides opportunities to calibrate the prediction failure of a page. In other words, pages on the ghost queue had been predicted as a victim page by Equation (2) or pushed out by obeying the rules of Table 2 (Line 24). Therefore, in the case of a cache hit, this queue provides opportunities to calibrate the previous prediction failure of a page on the queue by performing a learning operation with the same page (Line 14–Line 16). When making free space on the ghost queue, *ML-CLOCK* also triggers a learning operation to reflect the recency and frequency of the page pointed by the tail of the queue (Line 19–Line 21). This is significantly meaningful as it can help in finding patterns that will not be accessed in the future. Finally, the victim page determined by prediction and the preference rules is replaced by a new one (Line 26).

3.3. Example

Now, we describe the process of finding a victim page in *ML-CLOCK* with a very simple example (see Figure 1). Let us suppose that a cache miss occurs during the 40th new access in the case where there is no free space in the circular list. In this example, since `C-hand` points to the page with the page number of 99, *ML-CLOCK* starts to scan clean pages by following the traditional CLOCK algorithm. Finally, `C-hand` stops at page 73 in which its reference bit is zero (❶). *ML-CLOCK* simultaneously sweeps `D-hand`, which points to page 9, to find D-candidate, and `D-hand` is stopped on page 15 due to the same reason (❷). Note that, as shown in the figure, `D-hand` is moved among dirty pages in a sequential order, unlike `C-hand`. After finding both C-candidate and D-candidate, *ML-CLOCK* conducts prediction operations based on the reuse distance and the reference count of each page, respectively (❸–❹) In Figure 1, the timestamp of the last access is used to calculate a reuse-distance; it is subtracted from the current timestamp. It can be orders of magnitudes higher than the reference count; thus, we scale it down by using Equation (3). For example, since the timestamp of the last access regarding page 73 is 10, it is 10 orders of magnitudes higher than the reference count of the page (i.e., one); the raw value of the calculated reuse-distance may negatively affect the prediction operation. Next, *ML-CLOCK* uses Equation (2) to predict whether the page will be reused or not. In this example, the weight values for w_d, w_c, and w_b respectively used -1.0, 1.5, and 1.0, and those were calculated with Equation (4). In this case, the predicted values of each page were calculated as one; thus, *ML-CLOCK* selects page 73 as the victim page according to the preference rules listed in Table 2 (❺). Meanwhile, *ML-CLOCK* provides a second chance to page 15 by setting its reference bit as one because we believe the competitiveness of the page (❻).

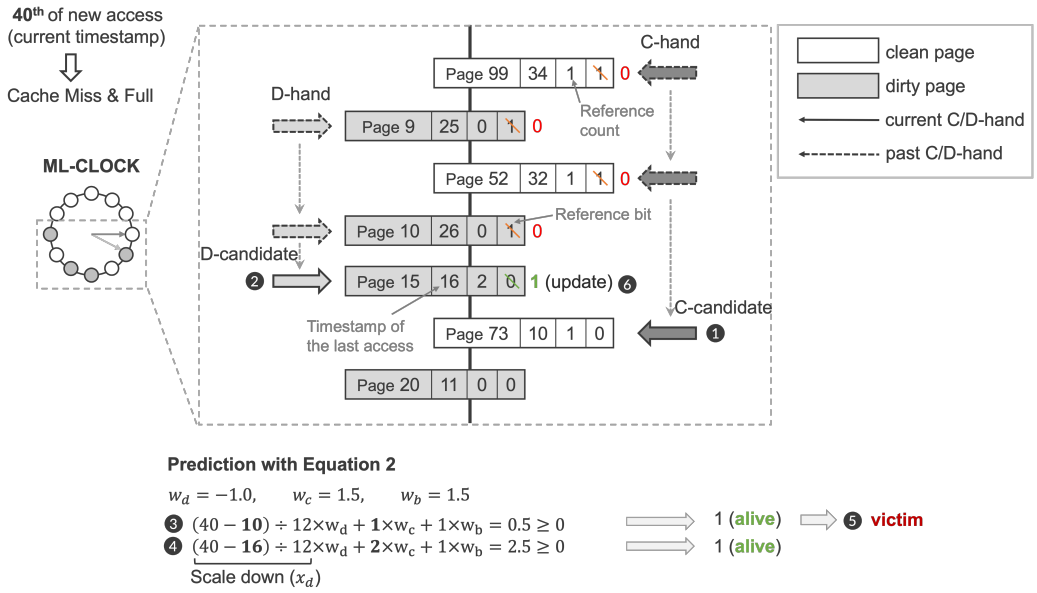


Figure 1. An example of *ML-CLOCK*.

4. Evaluation

We implemented *ML-CLOCK* to run under a trace-driven simulation and compared it with five replacement algorithms by using an I/O trace file as the input. The trace-driven simulation is a well known methodology in the evaluation of a replacement algorithm because it can be configured with various parameters values (e.g., cache size and performance metrics). In this section, we specifically answer the following questions:

- How much does *ML-CLOCK* improve the cache hit ratio under different workloads (Section 4.1)?
- How well does the prediction of *ML-CLOCK* apply corrections (Section 4.2)?

- How much does *ML-CLOCK* include performance overhead to apply the mechanism of machine learning (Section 4.3)?

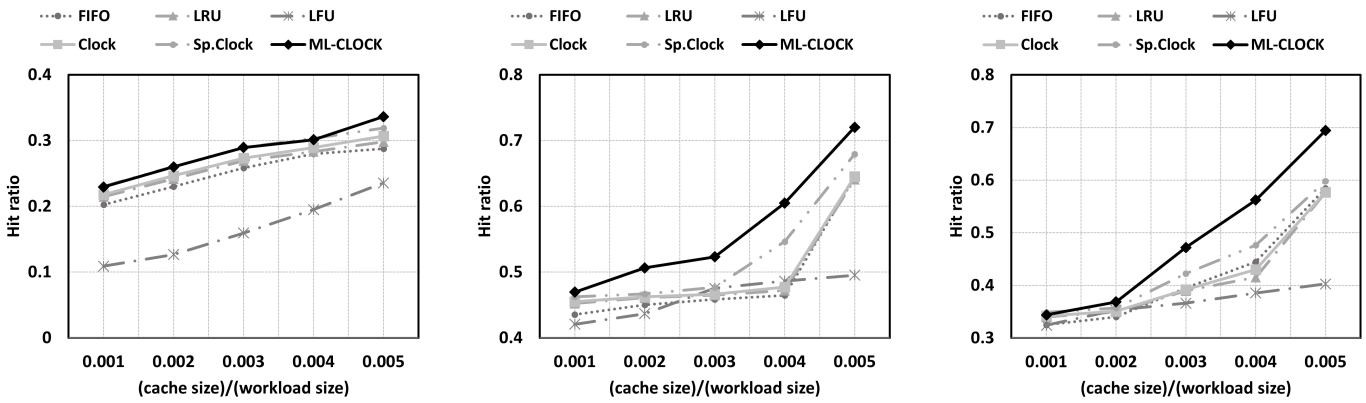
Experiment setup: All experiments described in this section are performed on a machine with the following specifications: an 8-core Intel(R) Core(TM) i7-9700 CPU @ 3.00 GHz, 32 GB memory, Samsung SSD 860 Pro, and Linux kernel 4.19 on Ubuntu 20.04 LTS. For comparison, we also implemented five replacement algorithms (i.e., FIFO, LRU, LFU, CLOCK, and Sp.Clock), and they are carefully selected according to the following steps. (1) We wished to compare *ML-CLOCK* with the FIFO algorithm that helps to understand replacement behaviors. (2) We also wanted to explore how recency and frequency information affects the replacement algorithm in detail. The reason behind this is that *ML-CLOCK* employs both recency and frequency information in order to select a victim page when the page cache is full. As a result, we added LRU and LFU algorithms to our evaluation so as to find the answer to our question. (3) Moreover, since *ML-CLOCK* follows the basic rules of CLOCK replacement algorithm, we added CLOCK to our evaluation set. (4) Finally, recent advances in storage markets have shifted to consider the SSD as the main storage device. Based on such trends, *ML-CLOCK* was designed to issue write operations in a sequential manner. Moreover, we added Sp.Clock to our evaluation set because it also shares the same design goal (i.e., flash-friendly replacement algorithm) [6]. In order to evaluate *ML-CLOCK* over a wide range of applications, three different workloads from the FIU repository were used as the input of our simulator: ikki, webmail, and web-vm [31]. Table 3 summarizes the characteristics of each workload in detail.

Table 3. Characteristics of FIU workloads [31].

	# of Lines	Read (%)	Write (%)	File Size (MB)
ikki	6337164	23	77	71.8
webmail	7795815	18	82	73.7
web-vm	14294158	22	78	146.6

4.1. Cache Hit Ratio

We first focus on the cache hit ratio because it is commonly used as one of the critical performance metrics for evaluating replacement algorithms. Figure 2 shows our evaluation results from the trace-driven simulation, and they are measured by varying the cache size from 0.001 to 0.005; the cache size is determined relative to the input workload. Figure 2 shows that the LFU replacement algorithm unfortunately reveals the lowest hit ratios in most workloads. The major reason for hit ratio collapse is that the frequency-based signal of LFU may result in a situation where a page is evicted early before accessing it again; we call this situation early eviction. Moreover, the other algorithms except for *ML-CLOCK* plot similar patterns; they were designed based on recency. On the other hand, as shown in Figure 2, *ML-CLOCK* shows similar or better cache hit ratios compared with other replacement algorithms. Specifically, *ML-CLOCK* significantly outperforms LFU by up to 72% in web-vm workloads when the cache size is 0.005. We expected these results before performing experiments because *ML-CLOCK* decides to a victim page along with intelligent approach (i.e., SLP), unlike other algorithms. The figure explains the answers for each prediction from *ML-CLOCK* frequently correct because it has more training and inference opportunities as the cache size increased. Thus, we believe such improvements come from the correct prediction of *ML-CLOCK*.



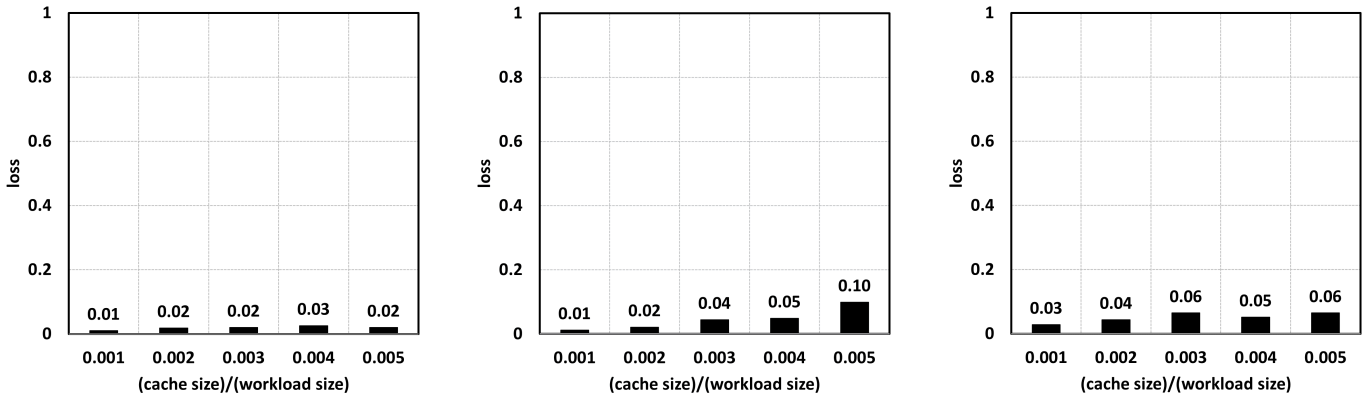
(**a**) ikki (**b**) webmail (**c**) web-vm

Figure 2. Cache hit ratio.

4.2. Loss of Accuracy

To confirm our intuition, we additionally measured the loss of accuracy in *ML-CLOCK* while running workloads. In ML, the loss of accuracy is the ratio that represents how the number of times our prediction for the questions resulted in wrong answers. Therefore, we calculated the loss of accuracy by dividing the total number of prediction counts by the total number of wrong answers.

Figure 3 plots the loss of accuracy in *ML-CLOCK* in detail. In the figure, the value '0' means that all predictions are correct; thus, a lower value points out good results. As shown in Figure 3, the prediction performed in our solution has rarely missed the opportunities for correct answers; miss prediction inevitably occurs in the initial step because of the lack of training and inference opportunities. These results also explain that we need to consider both recency and frequency metrics in designing a page replacement algorithm, and the metrics help to not only be trained for detecting the patterns of page access but also for predicting the possibility of re-access.



(**a**) ikki (**b**) webmail (**c**) web-vm

Figure 3. The loss of accuracy.

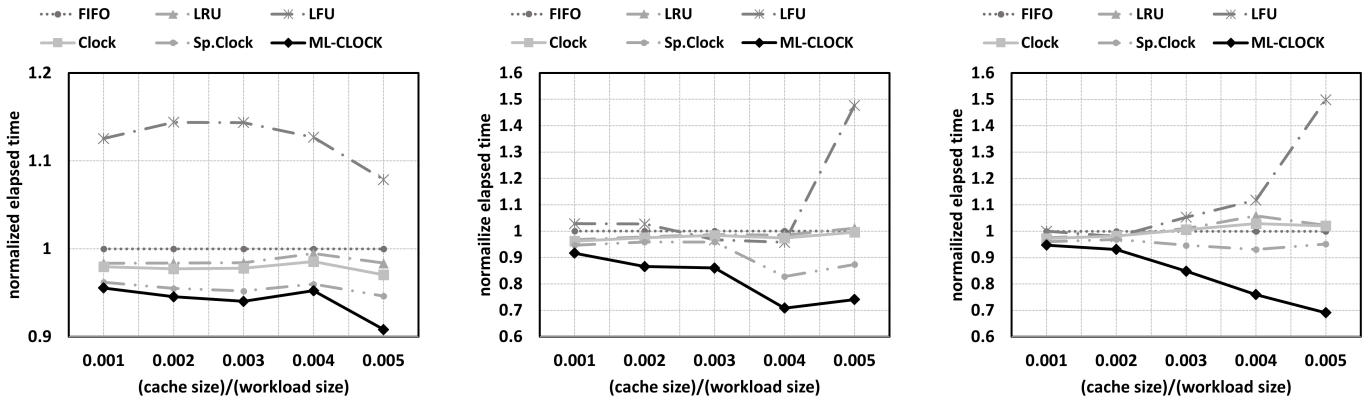
4.3. Simulated Performance

In general, a high hit ratio guarantees good performance in that the number of read/write operations issued to the underlying storage devices decreases. Unfortunately, *ML-CLOCK* inevitably includes extra time for online learning and prediction even though SLP is a very lightweight algorithm. To clearly confirm this overhead, we measured the spend time of unique operations in *ML-CLOCK* and collected the number of access operations for both DRAM and storage devices for all algorithms. We enumerated the elapsed time of each algorithm with Equation (5):

$$T_{elapsed} \leftarrow N_r \times L_r + N_w \times L_w + (N_{mem} \times L_{mem}) \tag{5}$$

where N_r and N_w represent the number of read and write operations issued to the storage device, and L_r and L_w denote the latency of storage-level read and write. In general, since the latency of DRAM read and write has the same period of time, we calculated the elapsed time of DRAM by combining read and write access and its cost; N_{mem} and L_{mem} indicate the number of DRAM access counts and the latency of DRAM, respectively.

Figure 4 shows the emulated elapsed time of algorithms where the latencies of DRAM and the storage device are configured with parameters from the paper [32]. As shown in the figure, *ML-CLOCK* can reduce elapsed time by up to 9%, 30%, and 30% with respect to ikki, webmail, and web-vm, respectively, compared with the FIFO algorithm. In addition, *ML-CLOCK* outperforms the LFU replacement algorithm by up to 2.16x on the web-vm workload. Note that such performance gain is very meaningful in that *ML-CLOCK* includes the spend time for online learning and prediction operations, unlike other algorithms. In summary, *ML-CLOCK* contributes much to reducing overall elapsed time even though it performs online training and prediction operations.



(**a**) ikki (**b**) webmail (**c**) web-vm

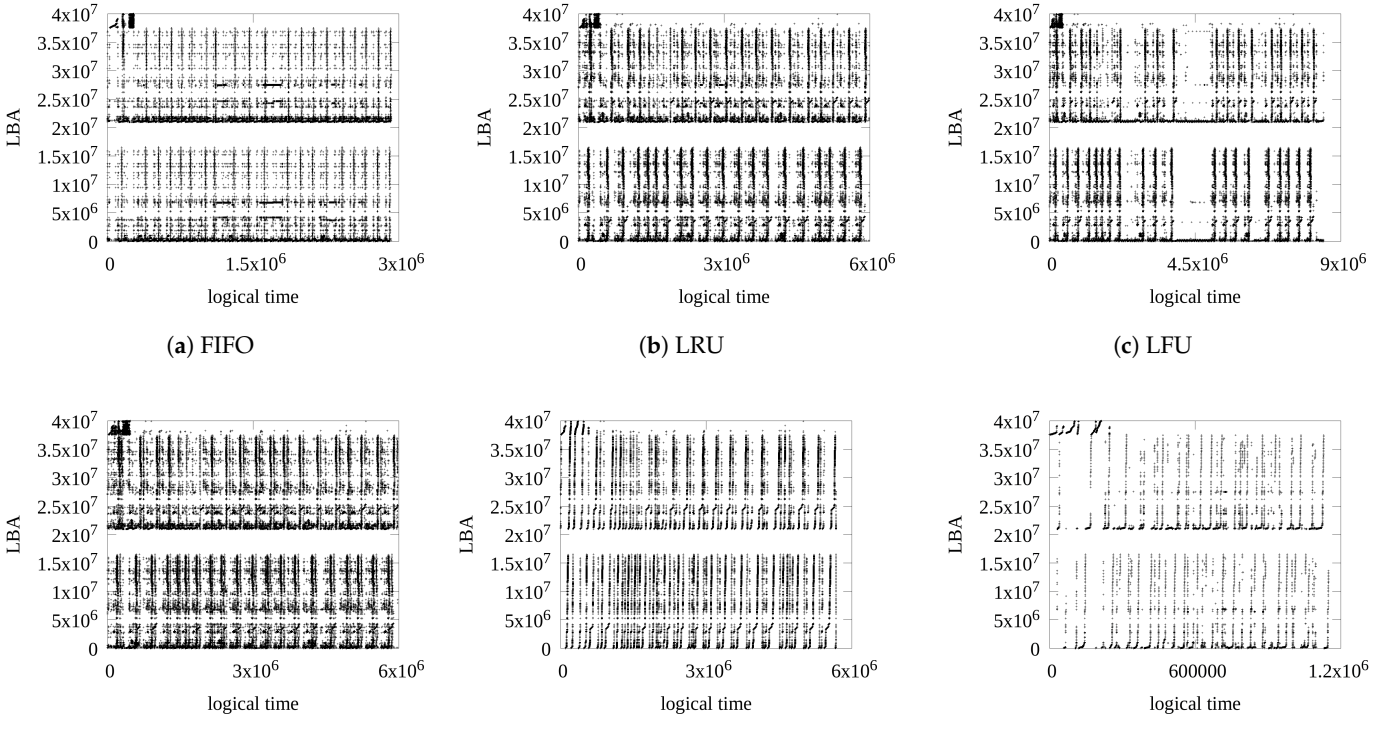
Figure 4. Emulated elapsed time of each algorithm. The results are normalized with respect to the FIFO replacement algorithm in order to clearly show the gap in elapsed time.

4.4. Analysis of Read and Write Patterns

Now, let us analyze the read and write patterns issued to the underlying storage device because the patterns strongly affect the overall performance of the underlying storage media and endurance in the case of flash-based SSD devices. To the best of our knowledge, a sequential write pattern is much faster than a random one both in HDD and SSD devices (including NVMe SSDs). Therefore, some researchers had designed flash-friendly replacement algorithms. For example, Sp.Clock was inspired by the observation that the patterns of a sequential write usually achieve higher performance and endurance in SSD devices [6]. Thus, Sp.Clock generates write operations in sequential orders by evicting pages on the sorted circular list based on logical block address (LBA). To take the

same benefits, *ML-CLOCK* also maintains `D-hand` and scans dirty pages on the circular list in a sequential order.

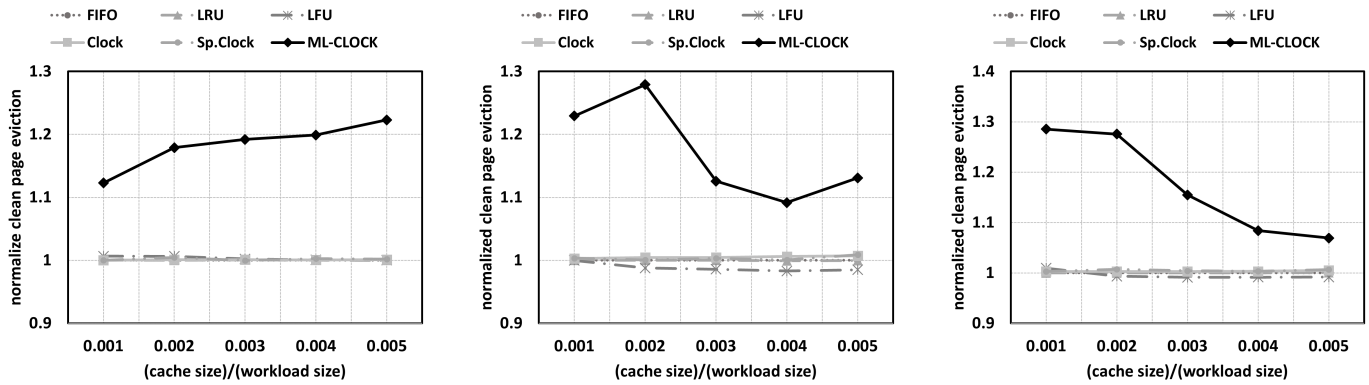
Figure 5 plots the reference of write operations to the underlying storage device. Due to space limitations, we only provide a figure on the web-vm workload with 0.005 cache size. As we expected, the algorithms except for Sp.Clock and *ML-CLOCK* show that they evict write operations in a random manner. On the other hand, Sp.Clock and *ML-CLOCK* show the efforts on making sequential writes. Interestingly, the figure shows the fact that the number of write operations on *ML-CLOCK* has much smaller density than those of Sp.Clock. The key reason behind this is that *ML-CLOCK* prefers to evict clean pages when an eviction operation is inevitably required to make free room in the circular list (i.e., clean-first eviction). To confirm the number of clean pages that are evicted from each algorithm, we additionally measured the number of read operations issued to the underlying storage device. Figure 6 compares the normalized number of clean pages. In the figure, we can observe a larger gap between *ML-CLOCK* and other algorithms, and we hypothesize that such a gap results from the clean-first eviction. In the worst case scenario, *ML-CLOCK* issues more clean pages by up to 28% compared with the FIFO replacement algorithm. However, we believe that the incremented read operations can be negligible in that the overall performance of *ML-CLOCK* in all cases outperforms other replacement algorithms (see Figure 4).



(**a**) FIFO (**b**) LRU (**c**) LFU

(**d**) Clock (**e**) Sp.Clock (**f**) ML-CLOCK

Figure 5. Write patterns of each algorithm issued to the underlying storage device (web-vm workload with 0.005 cache size).



(**a**) ikki (**b**) webmail (**c**) web-vm

Figure 6. The number of clean pages issued to the underlying storage device. The results are normalized with respect to the FIFO replacement algorithm.

5. Related Work

Past page replacement algorithms have focused on access patterns, such as the temporal and spatial locality [1–5], or the characteristics of the underlying storage media [6,7,33–35] in order to make dynamical decisions about a victim page. For example, when page allocation for new access is required, CLOCK-PRO [3] efficiently selects a victim page by classifying existing pages on CLOCK into hot and cold pages; a cold page is shifted forward to hot according to the number of accesses (i.e., the temporal locality). On the other hand, Sp.Clock [6] enhances spatial locality to consider the characteristics of NAND flash-based storage media (e.g., Micro SD Cards, SSDs, and NVMe SSDs). As we know, the performance of sequential read/write in the flash storage devices is much faster than the performance of write ones. Other researchers also focused on spatial locality to make a flash-friendly write pattern (i.e., sequential writes) [33–35]. TS-CLOCK [7] considered not only temporal locality for the page cache layer but also spatial locality for the storage media at once. Unfortunately, even though there are previous efforts, the page cache algorithm is still considered one of the reasons for performance bottleneck.

Meanwhile, over the few years, some researchers have revisited the page replacement algorithm in order to inject the core concept of neural network inference into them [15,16,19]. CACHEUS [15] has two different replacement algorithms: scan resistant LRU and churn resistant LFU. When there is no free page, it finds a victim page by determining which algorithm is used based on reinforcement learning that is one of the online learning mechanisms. Teran et al. introduced an attractive approach that renders the reuse distance in CPU cache level more sophisticated by using SLP [13]. Moreover, Sethumurugan et al. proposed reinforcement learned replacement (RLR) that learns a last-level cache policy without hardware implementation [36]. MLCache [16] learns an I/O pattern on NVMe SSDs where it allows the host to associate write operations with an independent stream and adjusts the cache space of each stream. For adjustments, MLCache employs MLP based on offline learning. ML-WP [37] identifies write-only data in various data traffic in order to reduce the number of write operations relative to the SSD cache in clouding computing environments.

To better understand, we summarized the differences between *ML-CLOCK* and prior studies (see Table 4) In this table, we focused on the replacement algorithms designed for the page-cache layer [3,6,7,15]. First, CACHEUS and *ML-CLOCK* were designed using ML technologies and online-learning, but CACHEUS does not consider the features of the underlying storage devices (i.e., sequential writes). Unfortunately, nowadays this consideration is very important in that most data storage devices have been replaced from HDDs to SSD devices. In SSD devices, the pattern of sequential writes can improve

endurance as well as the performance of the devices. Meanwhile, some algorithms without ML technologies selectively utilize the benefits of the recency (i.e., the spatial locality) and frequency (i.e., the temporal locality), as summarized in Table 4. However, they select a victim page without elaborate predictions running on the patterns of workloads. On the other hand, *ML-CLOCK* makes predictions for the next I/O pattern by learning and making inference steps and selects a victim page more efficiently. In summary, *ML-CLOCK* is different from previous studies in that it embraces both the benefits of online learning and the features of the underlying storage devices.

Table 4. Characteristics of the different page replacement algorithms.

	CLOCK-PRO [3]	Sp. Clock [6]	TS-CLOCK [7]	CACHEUS [15]	ML-CLOCK
ML	✗	✗	✗	RL	SLP
Recency	✓	✓	✓	✓	✓
Frequency	✓	✗	✓	✓	✓
Sequential writes	✗	✓	✓	✗	✓

6. Conclusions

Nowadays, applying ML theory is becoming a common technique in various environments. In this paper, we briefly studied how and when a neural network-based learning algorithm is used in the system layer. We also proposed a new page replacement algorithm, called *ML-CLOCK*, that provides online learning and prediction services for deciding which page is evicted. For evaluation, we implemented *ML-CLOCK* and five replacement algorithms and compared them using trace-driven simulation. More specifically, in this paper, we tried to prove whether *ML-CLOCK* efficiently orchestrates a single-layer perceptron algorithm and the features of the underlying storage devices. Our evaluation clearly confirms that *ML-CLOCK* outperforms all existing replacement algorithms for FIU workloads. We believe that the key idea behind *ML-CLOCK*, "ML-based sequential writes", can be extended to optimize a broad area of the system-layer for high-end storage devices, such as NVMe SSDs and SSDs. Therefore, our future work will focus on high-end computing systems (e.g., many-core CPU/GPU, NVMe-based storage devices, etc.) with diverse workloads. In particular, we plan the exploration of limitations in terms of two aspects. First, we will focus on the initial I/O situation where there is a lack of training and inference opportunities. Second, we consider performing extensive experiments on different environments, such as mobile devices and IoT.

Author Contributions: Conceptualization, M.C. and D.K.; methodology, M.C.; software, M.C.; validation, M.C. and D.K.; formal analysis, M.C. and D.K.; investigation, M.C. and D.K.; resources, D.K.; data curation, M.C.; writing–original draft preparation, M.C.; writing–review and editing, D.K.; visualization, M.C.; supervision, D.K.; project administration, D.K.; funding acquisition, D.K. Both authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1G1A1099932).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence;
ML	Machine Learning;
SLP	Single-Layer Perceptron;
MLP	Multi-Layer Perceptron;
ANN	Artificial Neural Network

CNN	Convolutional Neural network;
LSTM	Long Short-Term Memory;
ML-CLOCK	Maching Learning based CLOCK algorithm;
C-hand	Clean-hand;
D-hand	Dirty-hand;
C-candidate	Clean-candidate;
D-candidate	Dirty-candidate.

References

1. Silberschatz, A.; Galvin, P.B.; Gagne, G. *Operating System Concepts*, 9th ed.; John Wiley & Sons: Hoboken, NJ, USA, 2003; pp. 401–412.
2. Corbato, F.J. *A Paging Experiment with the Multics System*; Technical Report, MIT Project MAC Report MAC-M-384; MIT: Cambridge, MA, USA, 1968.
3. Jiang, S.; Chen, F.; Zhang, X. CLOCK-Pro: An Effective Improvement of the CLOCK Replacement. In Proceedings of the USENIX Annual Technical Conference, General Track, Anaheim, CA, USA, 10–15 April 2005.
4. Jiang, S.; Zhang, X. LIRS: An efficient low inter-reference recency set replacement policy to improve buffer cache performance. *ACM SIGMETRICS Perform. Eval. Rev.* **2002**, *30*, 31–42. [CrossRef]
5. Bansal, S.; Modha, D.S. CAR: Clock with Adaptive Replacement. In Proceedings of the USENIX Conference on File and Storage Technologies, Santa Clara, CA, USA, 31 March–2 April 2004; pp. 187–200.
6. Kim, H.; Ryu, M.; Ramachandran, U. What is a good buffer cache replacement scheme for mobile flash storage? *ACM SIGMETRICS Perform. Eval. Rev.* **2012**, *20*, 235–246. [CrossRef]
7. Kang, D.H.; Min, C.; Eom, Y.I. An Efficient Buffer Replacement Algorithm for NAND Flash Storage Devices. In Proceedings of the International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems, Paris, France, 9–11 September 2014.
8. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386–408. [CrossRef] [PubMed]
9. McCulloch, W.S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.* **1943**, *5*, 115–133. [CrossRef]
10. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
11. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
12. Kim, M.; Lee, S. Reducing tail latency of DNN-based recommender systems using in-storage processing. In Proceedings of the ACM SIGOPS Asia-Pacific Workshop on Systems, Tsukuba, Japan, 24–25 August 2020.
13. Teran, E.; Wang, Z.; Jiménez, D.A. Perceptron learning for reuse prediction. In Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture, Taipei, Taiwan, 15–19 October 2016.
14. Cao, Z.; Tarasov, V.; Tiwari, S.; Zadok, E. Towards better understanding of black-box auto-tuning: A comparative analysis for storage systems. In Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, 11–13 July 2018.
15. Rodriguez, L.V.; Yusuf, F.; Lyons, S.; Paz, E.; Rangaswami, R.; Liu, J.; Zhao, M.; Narasimhan, G. Learning Cache Replacement with CACHEUS. In Proceedings of the USENIX Conference on File and Storage Technologies, Online, 23–25 February 2021.
16. Liu, W.; Cui, J.; Liu, J.; Yang, L.T. MLCache: A space-efficient cache scheme based on reuse distance and machine learning for NVMe SSDs. In Proceedings of the IEEE/ACM International Conference On Computer Aided Design, San Diego, CA, USA, 2–5 November 2020.
17. Schroeder, B.; Lagisetty, R.; Merchant, A. Flash reliability in production: The expected and the unexpected. In Proceedings of the USENIX Conference on File and Storage Technologies, Santa Clara, CA, USA, 22–25 February 2016.
18. Toshiba Makes Major Advances in NAND Flash Memory with 3-Bit-per-Cell 32nm Generation and with 4-Bit-per-Cell 43 nm Technology. Available online: https://www.global.toshiba/ww/news/corporate/2009/02/pr1102.html (accessed on 23 September 2021).
19. Yoo, S.; Shin, D. Reinforcement Learning-Based SLC Cache Technique for Enhancing SSD Write Performance. In Proceedings of the USENIX Workshop on Hot Topics in Storage and File Systems, Online, 13–14 July 2020.
20. Minsky, M.; Papert, S.A. *Perceptrons: An Introduction to Computational Geometry*; MIT Press: Cambridge, MA, USA, 2017; pp. 1–316.
21. What Is Artificial Intelligence (AI)? Available online: https://www.ibm.com/cloud/learn/what-is-artificial-intelligence (accessed on 9 October 2021).
22. Bengio, Y.; LeCun, Y. Scaling learning algorithms towards AI. *Large-Scale Kernel Mach.* **2007**, *34*, 1–41.
23. Sahoo, D.; Pham, Q.; Lu, J.; Hoi, S.C. Online deep learning: Learning deep neural networks on the fly. *arXiv* **2017**, arXiv:1711.03705.
24. Chu, Q.; Ouyang, W.; Li, H.; Wang, X.; Liu, B.; Yu, N. Online multi-object tracking using CNN-based single object tracker with spatial-temporal attention mechanism. In Proceedings of the IEEE International Conference on Computer Vision, Venice, Italy, 22–29 October 2017.
25. Ergen, T.; Kozat, S.S. Efficient online learning algorithms based on LSTM neural networks. *IEEE Trans. Neural Networks Learn. Syst.* **2017**, *29*, 3772–3783.

26. Hoi, S.C.; Sahoo, D.; Lu, J.; Zhao, P. Online learning: A comprehensive survey. *arXiv* **2018**, arXiv:1802.02871.
27. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the the IEEE conference on computer vision and pattern recognition, Las Vegas, NV, USA, 27–30 June 2016.
28. Tan, M.; Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019.
29. Hu, W.S.; Li, H.C.; Pan, L.; Li, W.; Tao, R.; Du, Q. Spatial–Spectral Feature Extraction via Deep ConvLSTM Neural Networks for Hyperspectral Image Classification. *IEEE Trans. Geosci. Remote. Sens.* **2020**, *58*, 4237–4250. [CrossRef]
30. Yin, J.; Qi, C.; Chen, Q.; Qu, J. Spatial-Spectral Network for Hyperspectral Image Classification: A 3-D CNN and Bi-LSTM Framework. *Remote Sens.* **2021**, *13*, 2353. [CrossRef]
31. Verma, A.; Koller, R.; Useche, L.; Rangaswami, R. SRCMap: Energy Proportional Storage Using Dynamic Consolidation. In Proceedings of the USENIX Conference on File and Storage Technologies, San Jose, CA, USA, 23–26 February 2010.
32. Sudan, K.; Badam, A.; Nellans, D. NAND-flash: Fast storage or slow memory? In Proceedings of the Non-Volatile Memory Workshop, La Jolla, CA, USA, 4–6 March 2012; pp. 1–2.
33. Jo, H.; Kang, J.U.; Park, S.Y.; Kim, J.S.; Lee, J. FAB: Flash-aware buffer management policy for portable media players. *IEEE Trans. Consum. Electron.* **2006**, *52*, 485–493.
34. Kim, H.; Ahn, S. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In Proceedings of the USENIX Conference on File and Storage Technologies, San Jose, CA, USA, 26–29 February 2008.
35. Debnath, B.; Subramanya, S.; Du, D.; Lilja, D.J. Large block CLOCK (LB-CLOCK): A write caching algorithm for solid state disks. In Proceedings of the IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems, London, UK, 21–23 September 2009.
36. Sethumurugan, S.; Yin, J.; Sartori, J. Designing a Cost-Effective Cache Replacement Policy using Machine Learning. In Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, 27 February–3 March 2021.
37. Zhang, Y.; Zhou, K.; Huang, P.; Wang, H.; Hu, J.; Wang, Y.; Ji, Y.; Cheng, B. A machine learning based write policy for SSD cache in cloud block storage. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition, Grenoble, France, 9–13 March 2020.