eCache: A Sample-Inference-Based Intelligent Cache Scheme for High-Performance SSDs

Hui Sun, Yinan Fu, Xin Wang, Yi Zhou, and Xiao Qin, *Senior Member, IEEE*

***Abstract*—DRAM-based cache is a practical approach to enhancing the performance of large-capacity SSDs. Due to DRAM's limited capacity, cache sizes are significantly smaller than the data scale of workloads – and cache replacement schemes determine the cache hit ratio and SSD performance when a cache reaches full capacity. Prior work embarked on leveraging machine learning models (ML) to predict future access patterns in workloads, aiding cache replacement decisions for intelligent cache schemes inside SSDs. Unfortunately, existing ML-based cache schemes often overlook the impacts of data granularity – including page, request, and coarse granularities – of training datasets, model inference time overhead, and computational overhead on SSD performance. To address these challenges, we are motivated to propose an sample-inference-based intelligent *cache* scheme – eCache. eCache accurately predicts the future reuse distance of each sampled requested pages, which can enhance the accuracy of decision-making for the cache replacement. In particular, we design a parallel framework to curb the time overhead caused by model inference. We advocate for a random-group sampling inference method that utilizes the most accurate model while reducing computational overhead. Moreover, we implement eCache on the state-of-the-art SSD simulator, MQSim, and compare it against alternative cache schemes (*i.e.,* CCache, NCache, LAC, and VS-batch). The experimental results unveil that compared with the other cache schemes, eCache significantly reduces the average response time by up to 79.68% with an average reduction of 44.08%. When compared with the page-granularity-ML-empowered cache schemes, eCache greatly curtails computational overhead by up to 85.23% with an average reduction of 66.00%.**

***Index Terms*—Cache management schemes, machine learning, extreme gradient boosting, sample-inference, NAND flash memory, solid state disks.**

Received 20 December 2024; revised 21 July 2025; accepted 29 October 2025. Date of publication 5 November 2025; date of current version 15 January 2026. This work was supported in part by the National Natural Science Foundation of China under Grant 62472002 and Grant 62072001. The work of Xiao Qin work was supported by U.S. National Science Foundation under Grant IIS-1618669 and Grant OAC-1642133, in part by the National Aeronautics and Space Administration under Grant 80NSSC20M0044, in part by the National Highway Traffic Safety Administration under Grant 451861-19158, and in part by the Wright Media, LLC under Grant 240250 and under Grant 240311. Recommended for acceptance by C. Li. *(Corresponding author: Hui Sun.)*

Hui Sun, Yinan Fu, and Xin Wang are with the Department of Computer Science and Technology, Anhui University, Hefei, Anhui 230601, China (e-mail: sunhuiworking@gmail.com).

Yi Zhou is with the TSYS School of Computer Science, Columbus State University, Columbus, GA 31907 USA.

Xiao Qin is with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA.

Digital Object Identifier 10.1109/TC.2025.3628340

I. Introduction

NAND flash-based SSDs are widely adopted in high-performance storage systems thanks to their low power consumption and excellent random read/write performance [1]. Due to the physical limitations of NAND flash, its program/erase (P/E) cycles are finite [2]. To address this concern, high-performance and high-capacity SSDs utilize built-in DRAM as a cache to serve user requests [3], thereby reducing writes to back-end NAND flash to prolong SSD lifetime. In addition, the DRAM cache bridges the performance gap between the host and NAND flash, thereby shortening response time and increasing throughput. However, the cache size is significantly smaller than the data scale in workloads due to the limited space of expensive DRAMs. When a DRAM cache becomes full, cache schemes remarkably affect the cache hit ratio and SSD performance.

Existing cache replacement schemes can be broadly classified into two camps, namely, traditional heuristics and machine learning (ML)-based cache schemes. Traditional heuristic schemes [4], [5], [6], [7], [8], [9] rely on the spatiotemporal locality of workloads to improve the cache hit ratio. Unfortunately, a workload's locality amid complicated access patterns often changes frequently in multiple running phases – and it is challenging for these traditional schemes to evict data during the cache replacement [10]. Specifically, traditional heuristic cache schemes maintain a high cache hit ratio during high-spatiotemporal phases, but their performance drops sharply in low-spatiotemporal phases that result in significant fluctuations in SSD performance.

ML-based cache schemes [11], [12], [13], [14], [15], [16], [17], [18], [19] leverage ML models to learn the access patterns of workloads. Prior studies have quantitatively analyzed workload locality patterns – and demonstrated the performance advantages of ML-based cache schemes. [12] revealed that when processing workloads with poor spatiotemporal locality, the ML-based cache schemes achieve up to 32.7% higher read hit rates compared to the conventional heuristic caching schemes. Although these models adapt continuously to variable access patterns of workloads for improved performance, our findings reveal the following three issues in the existing ML-based cache schemes.

First, most existing ML-based cache schemes focus on model accuracy while overlooking the impact of data granularity on model accuracy. Notably, the data granularity in this paper refers to the size of the unit of data in the training dataset.

0018-9340 © 2025 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

The data granularity impacts training time, model accuracy, and computational overhead. In this paper, the computational overhead is referred to as the number of ML model inference operations for ML-based cache schemes.

Second, inference time as a key factor is neglected by the existing ML-based cache schemes. When the interval between two consecutive requests is short, the inference time of models enlarges the waiting time of requests to exacerbate I/O blocking. Although ML-based cache schemes strive to mitigate the impact of inference time by reducing it below the average request interval, the I/O blocking issue remains a major hurdle. Third, the existing ML-based cache schemes rarely consider the trade-off between the computational overhead and model accuracy. Specifically, ML modules implemented in resource-limited SSDs tend to introduce extra computational overhead. Moreover, the finer data granularity in training datasets causes higher computational overhead.

To address the aforementioned issues, we propose an intelligent sample-inference-based intelligent cache scheme named eCache in this paper. In particular, we first construct an ML-empowered cache replacement model anchored on page-granularity training datasets. This model is slated to accurately predict the future reuse distance of each requested page, providing accurate decision-making for the cache replacement. Additionally, unlike the existing ML-based cache schemes [11] [12], eCache allows requested pages to enter the cache from the host directly rather than await the ML model to produce labels for these pages. The ML module continuously samples pages for inference (*inference candidates*) from workloads and labels for inference candidates during the cache management inside SSDs. Therefore, eCache conducts the ML model inference and cache management in parallel, which addresses the impact of ML model inference time on the SSD performance. Furthermore, we put forth an inference candidate sampling strategy – random-group sampling – to cut back computational overhead for the ML-based cache scheme without degrading SSD performance.

The contribution of this paper is summarized as follows.

▲ We propose a sample-inference-based intelligent cache scheme named eCache, which parallelizes cache management and model inference to curtail the adverse impact of model inference time overhead on response time.

▲ We design an XGBoost-based ML model in eCache and create an arrival linked list for model-sampling candidates. In addition, we devise three linked lists and an ML-aware active writeback module to optimize inference results.

▲ We leverage sampling inference to significantly reduce the computational overhead of the ML model, achieving a balance between accuracy and overhead. We advocate for an inference candidate sampling strategy – random-group sampling – to generate sufficient cold pages and scale back the number of inference operations.

▲ We implement eCache in MQSim and compare it with the state-of-the-art cache schemes. The experimental results demonstrate that eCache can shorten the average response time of alternative cache schemes by up to 60%. eCache achieves higher performance with a lower computational overhead than the other ML-based cache schemes. eCache's inference overhead is 10%-30% that of page-granularity ones.

The rest of this paper is organized as follows: §II provides background. In §III, we study workloads and present the motivation of eCache. §IV demonstrates the design of eCache. §V shows the experimental setups and performance of eCache compared with state-of-the-art cache schemes. §VI lists the related work, and we draw the conclusion in §VII.

II. Background

An SSD consists of host interface logic (HIL), flash translation layer (FTL), and a back-end flash memory array [20]. HIL is the interface between the host and the SSD, and it slices user requests into page-sized transactions named page. The FTL handles address translation, garbage collection (GC), and wear leveling. The back-end channels, chips, dies, planes, blocks, and pages offer four-level parallelism to boost SSD performance. A block is the basic unit for erases, and a page is the unit for reads/writes.

An out-of-place update mechanism in NAND flash can trigger GC and wear out NAND flash. To mitigate this issue, SSDs employ built-in DRAM as a cache. The cache stores hot data to reduce the number of writes in flash memory, extending SSD lifetime. In addition, the cache bridges the performance gap between the host and NAND flash, reducing response time. However, due to its high cost, cache capacity is smaller than the size of NAND flash and the data size of workloads. Consequently, a cache replacement scheme determines which data to be evicted when the cache is full. Several studies employed read-write cache schemes in SSDs [12], [21]. This approach is suitable for the complex and dynamic modern I/O workloads, where access patterns (e.g., read/write intensity) may shift dynamically. Read or write cache schemes struggle to balance performance—pure read caches significantly increase latency under write-intensive workloads due to forced write-through to flash, while pure write caches suffer performance limitations under read-intensive workloads as data must be fetched from flash memory. Thus, a read-write cache scheme can adapt to mixed and dynamically changing workloads, an approach validated in commercial products (e.g., Seagate Nytro series [22]).

Existing heuristic cache schemes largely depend on the spatiotemporal locality of requests in a workload, facing performance bottlenecks due to the complexity of workloads [10], [23], [24], [25]. For example, the traditional heuristic schemes experience performance fluctuations in low-locality phases of a workload. To this end, ML models that learn the I/O access patterns of workloads become a promising approach for managing the data in SSDs. In an ML-based cache scheme, its ML model collects historical information for each sample as a feature according to the data granularity of the training dataset. It then labels each requested page based on these features, and the cache evicts pages according to the labels assigned by the ML model. Thereafter, *"page"* denotes *"user requested page"* in the following sections. However, existing ML-based cache schemes still face several challenges.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

TABLE I
DETAILS OF TRACES USED IN THIS PAPER

Workload[1]	TotalReq (10^6)	RD^2_{req} (10^6)	UniRatio[3]	WriteRatio	TotalPage (10^6)	Req^4_{as} (KB)	Req^5_{ai} (ms)	Req^6_{st} (10^3)	$Req^7_{1\mu s}$	I^8_{avg} (MB/s)	I_{max} (MB/s)
LUN1-1	1.38	1.35	82.35%	20.40%	4.35	17.70	2.61	1.78	0.21%	6.63	242.93
LUN2-1	2.86	2.80	78.94%	20.51%	14.60	33.31	1.26	5.18	0.27%	25.85	297.56
LUN3-1	2.06	2.01	79.68%	25.77%	10.77	34.34	1.75	2.57	0.20%	19.21	307.01
LUN2-2	5.23	5.20	26.97%	73.09%	13.19	12.70	0.69	3.69	0.13%	18.01	249.11
LUN2-3	4.39	4.36	39.70%	59.53%	12.42	15.13	0.82	4.28	0.17%	18.03	173.60
LUN1-2	2.20	2.17	38.97%	61.81%	9.26	26.12	1.63	2.78	0.21%	15.63	151.08
LUN0-1	2.21	2.17	39.80%	61.67%	5.67	12.95	1.62	1.82	0.13%	7.79	139.88
hm_0	3.99	3.41	11.00%	64.50%	6.56	7.99	151.45	105.37	18.73%	0.05	37.86
prn_1	11.23	10.95	19.35%	24.66%	33.08	19.80	53.84	3408.08	53.95%	0.36	79.90
proj_0	4.22	4.16	6.78%	87.52%	22.38	38.04	143.16	35.04	4.54%	0.26	37.03

Note: [1]LUN1-1, LUN2-1, LUN3-1, LUN2-2, LUN2-3, LUN1-2, and LUN0-1 represent traces 2016022008-LUN1, 2016022009-LUN2, 2016022009-LUN3, 2016022315-LUN2, 2016022417-LUN2, 2016030416-LUN1, and 2016030914-LUN0, respectively. [2]RD_{req}: the random request in workloads. [3]UniRatio: the proportion of unique requests in workloads. UniRatio is the degree of locality in workloads. [4]Req_{as}: the average size of requests in workloads. [5]Req_{ai}: the average interval of requests in workloads. [6]Req_{st}: the requests accessed simultaneously in workloads. [7]$Req_{1\mu s}$: the proportion of requests in workloads with an inter-arrival time of less than 1 μs. [8]I_{avg} and I_{max}: the average and maximum number of requests per second.

III. MOTIVATION

In this section, we present the challenges in existing cache schemes, which motivates us to design eCache. Table I lists features of traces. Our experiments reveal that the cache performance fluctuates under traces with various access patterns (see §IV-C1). ML-based cache schemes can predict future I/O access patterns of workloads [11], [12], [14]. However, these schemes face the following challenges. First, the data granularity in ML model training affects the model accuracy. Second, the inference time exacerbates I/O blocking during short request intervals. Third, the fine-granularity ML-based cache schemes introduce high computational overhead.

A. Finding 1. The Data Granularity of Training Datasets Affects the Model Accuracy of ML-Based Cache Schemes

For an ML model-based cache scheme, existing studies capture the features of data in the workload and corresponding labels to construct the training dataset, where the label represents the data's access patterns. The dataset is used to train models using ML algorithms (*e.g.,* random forest, decision tree). In this paper, we define the size of the data unit being processed in the training dataset as the data granularity, which can be page, request, and coarse (the size of a coarse unit is larger than that of a request unit). The trained models are accordingly classified as page-, request-, and coarse-granularity models, the abbreviations of which are the P-ML model, R-ML model, and C-ML model. The cache schemes based on these models are referred to as P-ML, R-ML, and C-ML cache schemes.

C-ML cache schemes have the largest data granularity and the fewest inferences, leading to lower computational overhead but the lowest model accuracy. For example, GL-cache [26] uses a group-granularity model, and the actual access pattern of each page within the group may not match the predicted one of the group, reducing the model accuracy. P-ML cache schemes achieve the most precise cache replacement decisions due to their superior accuracy in predicting page access patterns. However, this precision comes at the highest computational overhead. As shown in columns 2 and 6 of Table I, the number of pages in a workload is larger than the number of requests. Although the inference costs per request or page are small, the computational overhead of P-ML cache schemes is higher than that of R-ML ones (*e.g.,* 5× higher in LUN2-1). R-ML cache schemes have a lower computational overhead than P-ML ones and a higher accuracy than the C-ML ones. In R-ML cache schemes, all pages within a request have the same access pattern. Thus, R-ML cache schemes, similar to C-ML schemes, may not consistently achieve accurate decisions. In addition, we find that workloads exhibit request overlap and request partial reuse, which reduces R-ML models' accuracy. Moreover, page sharing takes place when requests are sliced into pages inside SSDs.

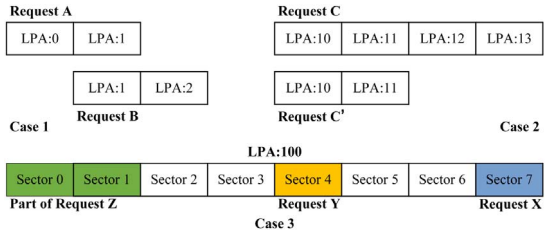


Fig. 1. The request overlap, partial reuse, and page sharing issues among pages inside an SSD.

In this paper, the term **request overlap** refers specifically to **partial overlap**. This occurs when requests with different starting LSAs are partitioned, resulting in sub-requests that will share pages in common. In case 1 of Fig. 1, both requests A and B contain LPA1. Then, LPA1 reflects the access patterns of the two requests simultaneously. LPA0 and LPA2 can be cold pages, while LPA1 is hot. **Request partial reuse** represents partial pages of a request that are re-accessed when the request hits again, see case 2 in Fig. 1. Requests C and C' have the same starting logic sector address (LSA), but LPA10 and LPA11 are re-accessed after C is accessed. Then, the access patterns are different between the first and last two pages. In **page sharing**, requests (x, y, and a portion of z) occupy different sectors in a page. The page's access frequency accumulates these requests.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

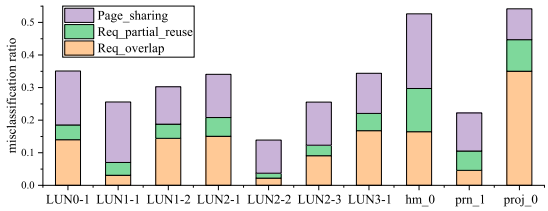


Fig. 2. Misclassification ratio in different workloads.

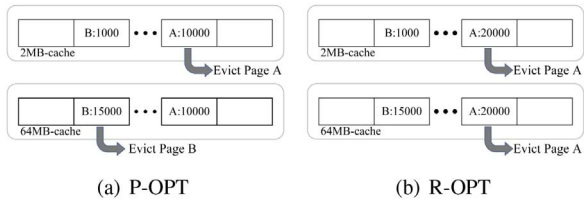


Fig. 3. Decision-making of P- and R-OPTs.

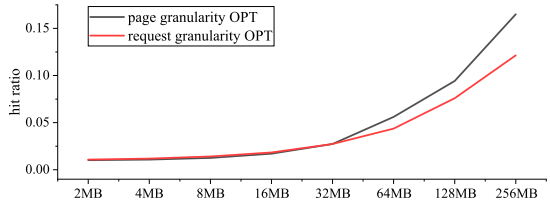


Fig. 4. The cache hit ratios between P- and R-OPTs.

TABLE II
DESCRIPTION OF TP, FP, TN, AND FN

TP (True Positive)	The number of pages correctly labeled as hot under the actual access pattern in the workload.
FP (False Positive)	The number of pages incorrectly labeled as hot under the actual access pattern in the workload.
TN (True Negative)	The number of pages correctly labeled as cold under the actual access pattern in the workload.
FN (False Negative)	The number of pages incorrectly labeled as cold under the actual access pattern in the workload.

Notably, larger pages with more sectors increase the potential for page sharing.

According to request overlap, request partial reuse, and page sharing in workloads, we define a misclassification ratio (MR) as the ratio of pages whose access patterns are misclassified to the total number of pages in the case of request granularity. MR unveils the differences in the prediction and actual access patterns of pages in the request-granularity training dataset.

We count the number of true positives, false positives, true negatives, and false negatives in the training dataset. The binary classification trade-off [27] is used to measure the accuracy of labeling approaches. We classify the access patterns of a workload into cold pages and hot pages. Table II lists our label categories and classification results. In the request-granularity training dataset, the request overlap and page sharing increase FNs, while the request partial reuse enlarges FPs. Thus, we have MR $= \frac{\text{FP+FN}}{\text{TP+FP+TN+FN}}$, where *MR* is prevalent in workloads using the request-granularity training dataset. In Fig. 2, the MR of hm_0 is above 50%. The highest ratio of request overlap, request partial reuse, and page sharing is 35.0%, 13.3%, and 22.9%, respectively.

To study the impacts of P- and R-ML models on the decision-making of cache schemes, we use P-ML and R-ML models to construct page- and request-granularity optimal cache replacement schemes (P- and R-OPT) [28]. For a small-sized cache, the difference between the two OPTs is negligible, and the misclassification of pages' access patterns does not impact the hot/cold data separation, see Fig. 3. Pages P_a and P_b have the largest future reuse distance (*Frd*), which means the number of pages that need to be accessed before a page is reused next time. For a 2-MB cache and an 8-KB page, the cache stores 256 pages with the smallest Frd, and the Frd of P_b is 1,000. The two OPTs evict P_a when a cache replacement occurs. Therefore, the misclassification of pages' access patterns in the small-sized cache cannot affect the hit ratio. However, as the cache size increases, the two OPTs differently process P_a. For a 64-MB cache and an 8-KB page, the cache stores 8,192 pages with the smallest Frd. As the number of cache pages increases, P_b's Frd rises to 15,000. The P- and R-OPTs recognize P_a's Frd as 10,000 and 20,000, respectively, and the P- and R-OPTs evict P_b and P_a, respectively. In Fig. 4, with a 256-MB cache, the hit ratio of the P-OPT is 30% higher than the other one.

In summary, an R-ML model assigns the same labels to all pages within a request, which overlooks request overlap, partial reuse, and page sharing issues. Then, the predicted access patterns of the ML model are inconsistent with the actual access patterns inside an SSD. Otherwise, the P-ML model accurately predicts the access pattern inside the SSD. Thus, we build a page-granularity training dataset, capturing each page's Frd, and assigning labels based on its Frd value. However, the P-ML cache schemes cause high computational overhead in the inference, see §III-B and §III-C.

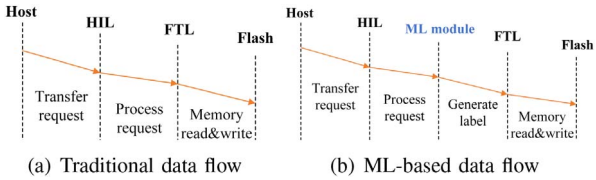
B. Finding 2: The Time Overhead of the ML Module Affects the System Performance

Prior work [11], [12], [29] focuses on the accuracy of ML models instead of the overhead of ML modules inside SSDs. The overhead includes storage space required for features and models, inference time, and computational overhead. The computational overhead means the number of ML model inference operations.

Existing ML-based cache schemes focus on model accuracy and space overhead. They exploit small-sized models and few features to balance inference accuracy and space overhead. In Fig. 5, ML-based cache schemes introduce extra inference time into the data flow in an SSD. Two methods reduce the time overhead. First, the inference time is smaller than the average request interval in a workload. Second, the ML module is pipelined into the data flow, see Fig. 5(b). When the request interval is larger than the inference time, the pipeline can lessen the time overhead, avoiding the requests' additional waiting time from the host to SSD.

Unlucky, this approach still suffers from the inference time overhead. The pipelining can offset the time overhead when

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.



(a) Traditional data flow (b) ML-based data flow

Fig. 5. Data flow of the traditional and ML-empowered SSDs.

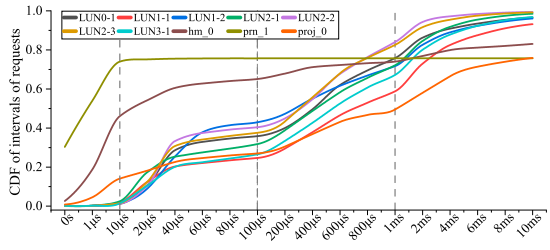


Fig. 6. The cumulative distribution function (CDF) of the request interval. The y-axis represents the proportion of request intervals no larger than the threshold (x-axis) to the total request intervals in the workloads. We set several points (1μs, 10μs, 100μs, and 1ms) to study the inference time of various ML models.

the request interval exceeds the inference time. However, the inference time exacerbates I/O blocking when the request interval is shorter. Although the average request interval varies, multiple requests arrive simultaneously, and the request interval is often small. In addition, we study the distribution of request intervals in workloads, see columns 8 to 10 in Table I. Although the proportion of small request intervals is usually low, the pipelining cannot eliminate the inference time overhead of an ML model.

Furthermore, we compare the request intervals with the model inference time. In Fig. 6, MSR traces [24] have a high proportion of small request intervals, with up to 30% of requests arriving simultaneously in prn_1. In VDI traces [10], the percentage of intervals less than 10μs is about 3%, and most inference time is greater than 10μs. The percentage of intervals less than 100μs ranges between 20% and 40%, whereas the inference time of most deep learning models (*e.g.,* DNN) exceeds 100μs.

We find the following phenomena: (1) the interval between two requests is usually smaller than the inference time; (2) the pipelining cannot offset the inference overhead; and (3) the request waiting time increases due to the inference time, exacerbating I/O blocking and degrading system performance. These findings motivate us to propose a cache scheme to address the inference time overhead. We design a method to parallelize the model inference and cache management to eliminate the time overhead (see §IV-C1).

C. Finding 3: An Inference Module in the SSD Can Cause Additional Computational Overhead

According to §III-A and §III-B, the inference for all data in workloads causes remarkable overhead because of a large number of requests. Table I shows that inference for all pages in a workload aggravates computational overhead of P-ML cache schemes on limited-resource SSDs. Thus, we aim to design an intelligent cache scheme to reduce the model's computational overhead while maintaining high accuracy.

Both the coarse-granularity and sampling inference can reduce the number of inferences to lessen the computational overhead of ML models. In §III-A, the former reduces model accuracy, whereas the latter depends on the proportion of cold pages (unaccessed or re-accessed for a long period pages) in a workload. The model is less likely to predict cold pages when the proportion is low, leading to insufficient cold pages for cache replacement. Conversely, if the proportion is high, reducing the number of inferences can yield enough cold pages. As a result, they are prioritized to be evicted during cache replacement.

The key to sampling inference is to reduce inference operations while predicting enough cold pages for replacement. Therefore, these cold pages are crucial for reducing computational overhead. To reduce the model inference time and computational overhead, it is essential to analyze the workload's distribution of cold and hot pages.

In Fig. 7, we find a large number of pages only accessed once (the coldest pages) in the workloads. The maximum percentage of these pages reaches 69%. Thus, evicting such pages is optimal. If we define pages with a Frd greater than 3 $\times$ n as cold pages, at least 42.5% of the pages in VDI workloads and at least 15.5% of the pages in MSR workloads are cold in the case of a 256-MB cache. Notably, we explore the number of cold pages in the workload at different cache sizes. When the cache size becomes smaller, our proposed eCache can recognize more pages as cold. These findings motivate us to reduce the number of inference operations.

In the ML-based cache scheme, a high percentage of cold pages increases the likelihood of labeling cold pages. Although the model inference on all pages causes more pages to be labeled as cold, it introduces high computational overhead. Therefore, we should design a method to sample a portion of cache pages for the ML model inference, which aims to achieve as many cold pages as possible in case of low computational costs in the cache replacement, see §IV-C2.

IV. Design of eCache

A. System Overview

Motivated by the above three findings, we propose a sample-inference-based intelligent cache scheme named eCache in Fig. 8. In eCache, we employ an XGBoost to train an ML model offline based on the page-granularity dataset collected by the feature collection module. eCache consists of the arrival linked list, ML module, user page pool (UP), priority eviction linked list (PL), eviction candidates linked list (CL), and active write-back module. HIL slices a user's I/O request from the host into page-sized transactions in the cache (*i.e.,* page or cache page), which are inserted into UP in an LRU manner. The ML module collects the features of each page. eCache divides the arrival linked list into equal-sized groups. When the cache management module undergoes updates and replacements, the ML

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

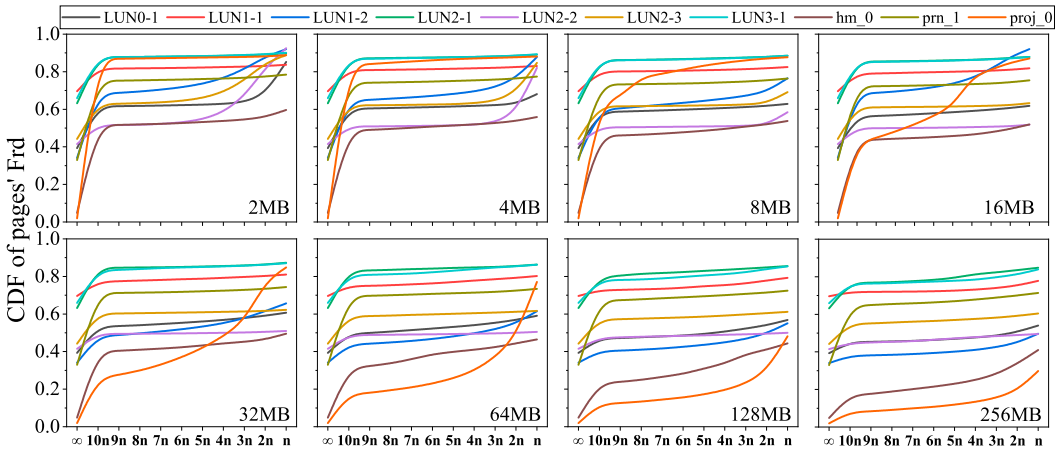


Fig. 7. CDF of pages' Frd. The y-axis denotes the proportion of pages with frd no larger than the threshold (x-axis) to the total pages' Frd in the workloads. *n* represents the cache size in terms of pages.

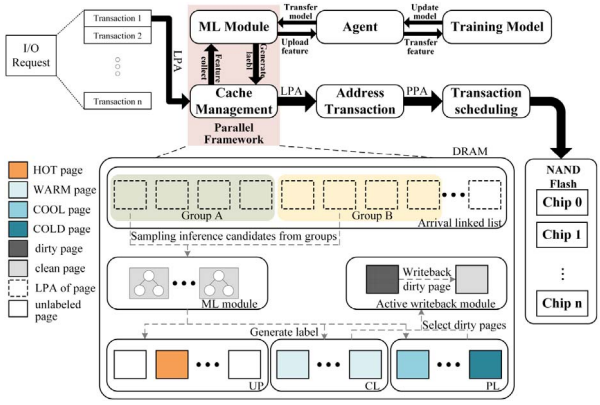


Fig. 8. Overview system of eCache.

module randomly samples pages in each group and labels them according to the features (see §IV-C1). These sampled pages are referred to as inference candidates. In addition, eCache parallelizes the ML model inference with cache management.

eCache separates hot and cold pages based on labels from the ML model. Pages labeled as COLD and COOL are migrated from the UP to PL, and pages labeled as WARM move to CL. In the cache replacement, eCache preferentially evicts pages in the PL, followed by the CL, and finally the UP. When the PL and CL are not empty, the active write-back module prioritizes actively writing back dirty pages in the PL to flash memory, followed by the CL, when the target chip is idle. In addition, the host loads the updated trained model onto the SSD via the agent module.

B. Machine-Learning Model in eCache

1) Features Collection: In the following, we present the features collected in this study. **Recency array** is the number of pages accessed between two consecutive accesses to the same LPA. R[0] is the number of pages in the most recent interval, while R[1] denotes the number of pages in the previous interval. R[2] follows the same pattern. **Frequency** is the number of accesses to each page. It reflects the popularity of the page in the past. **Request size** is related to its popularity [29]. Large requests may be accessed less frequently, while small ones can be updated and accessed frequently. **Page size** is the number of valid sectors per page. In §III-A, page sharing exists in SSDs, where the page can be accessed frequently.

These above features are page-granular, which provide the highest accuracy, see §III-A.

2) Training Dataset: eCache constructs the page-granularity training dataset from a workload that is composed of features and labels. These features include the recency array, frequency, page size, and request size of the page. The label represents the future reuse distance of each page (*Frd*), which is the number of pages accessed between the current access to page P_i and the subsequent access to the same page.

The overarching goal of eCache is to protect hot data. The cache eviction threshold of eCache determines whether a page is hot or cold. eCache preferentially evicts pages whose *Frd* exceeds the eviction threshold. If the threshold is too high, there are few cold pages, reducing the likelihood of predicting cold pages as the cache size increases. When the threshold is too low, pages to be accessed soon are evicted, thereby reducing the hit ratio. We associate the labels with the cache size to accommodate various cache sizes. To obtain the label, we define the eviction threshold in eCache as $i \times n$, where n is the cache size in terms of pages. We run the eCache under workloads with different thresholds and collect the number of evicted pages from PL, CL, and UP. We define the effective eviction ratio (P) as $P = (E_{PL} + E_{CL})/(E_{PL} + E_{CL} + E_{UP})$, where E_{PL}, E_{CL}, and E_{UP} denote the number of pages evicted from the PL, CL, and UP, respectively.

In Fig. 9, we study the evicted cold pages under thresholds. When the threshold is set to $4\times$, and $5\times$ n, P of evicted pages reduces sharply in the case of 64 MB. This result unveils that eCache evicts pages from the UP, including HOT and NONE pages. It is difficult to distinguish hot and cold pages in this scenario. Notably, the labels of all newly cached pages are initialized as NONE. When a cached page is accessed – known as

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

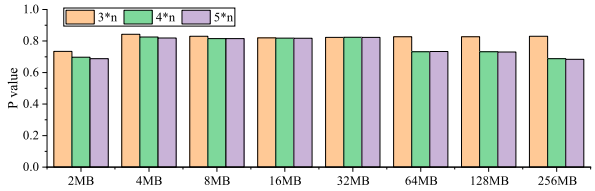


Fig. 9. P values (y-axis) of different eviction thresholds at eight cache sizes (x-axis) under LUN2-2. Symbol 3 × n represents the eviction threshold.

a cache hit, its label is reset to NONE. In addition, we find that P of 3× n almost has no change, indicating that pages whose Frd are no larger than 3× n are labeled as HOT. In addition, if the Frd of a page is greater than 3 × m, (m denotes the cached mapping table (CMT) [30] size in terms of pages), the Frd of this page is considered too large and may not be retained in CMT. Consequently, the page is labeled as COLD. Furthermore, we conduct experiments to derive empirical values for the labels WARM and COOL to achieve a more fine-grained classification of page hotness. Specifically, pages with Frd between 3 × n and 10 × n are labeled as WARM, and those with Frd between 10 × n and 3 × m are labeled as COOL. Then, we have

$$\text{Label} = \begin{cases} \text{HOT} & \text{if Frd} \leq 3 \times \text{n} \\ \text{WARM} & \text{if } 3 \times \text{n} < \text{Frd} \leq 10 \times \text{n} \\ \text{COOL} & \text{if } 10 \times \text{n} < \text{Frd} \leq 3 \times \text{m} \\ \text{COLD} & \text{if Frd} > 3 \times \text{m} \end{cases} \quad (1)$$

which is used by eCache to define labels. Notably, the pages that have never been performed inference or whose labels have been reset due to data updates are labeled as NONE. Finally, we construct eCache's training dataset using the features in §IV-B1 and the labels defined in Equ. 1.

3) ML Model Selection: It is challenging to deploy ML models on SSDs, as it must balance accuracy and computational overhead. Simple ML models (*e.g.,* support vector machine and Bayesian) have small overhead but low accuracy when processing multi-classification tasks. Neural networks (NN) perform excellently in many applications and are highly accurate. However, the model training requires expensive computing resources, and the inference can cause high time and computational overhead inside SSDs (see §III-B and §III-C). Since the accuracy and overhead of each model vary, it is crucial to select the appropriate model for the cache design. The extreme gradient boosting (XGBoost) is a flexible and prevalent gradient-boosting algorithm in ML [31]. XGBoost uses multi-threading and parallel computing techniques to accelerate the training, especially for large-scale datasets.

We first train the XGBoost model through Python Scikit-learn on an Intel Core i7-12700 CPU (12 cores, 2.0 GHz) to measure its training time and average inference time. Subsequently, to simulate various processors within SSDs, we converted both training time and average inference time to that for the Cortex-A53 platform using the FLOPs ratio between the embedded ARM Cortex-A53 processor (4 cores, 1.2 GHz) and the i7-12700, see Table III. Table III shows that the inference and training times of XGBoost are close to those of the fastest

TABLE III
COMPARISON OF ML MODELS

Model	Inference time(μs)	Training time (s)	Accuracy	AUC	Size (KB)
XGBoost	2.99	28	90.55%	0.9305	337.2
DT	1.12	11	77.45%	0.8365	7.1
RF	78.04	305	87.56%	0.8892	644.1
DNN	236.06	2486	93.38%	0.9598	79.1

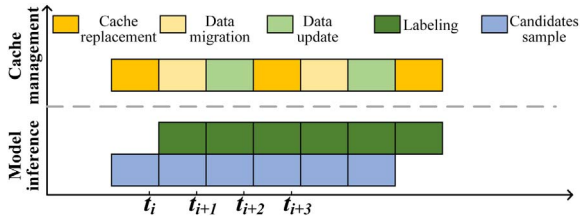


Fig. 10. The parallel framework of eCache.

decision tree (DT), and its model accuracy is 90. 55%. Thus, XGBoost has low overhead and is suitable for resource-limited SSDs. Compared to XGBoost, DT has smaller overhead and model size but lower accuracy due to its simple structure. Random forest (RF) has similar accuracy as XGBoost but delivers higher inference and training time. When RF's subtrees are close to that of XGBoost, RF's accuracy decreases. The DNN model is featured with the highest accuracy and the smaller size; however, its inference and training time is the longest. Furthermore, we added a new metric area under the ROC curve (AUC) in Table III, and it measures the model's ability to distinguish between positive and negative classes across all classification thresholds. XGBoost exhibits a significant advantage in AUC over DT and RF, approaching that of DNN. XGBoost is the best one to balance the accuracy and overhead. In addition, a workload can contain many new and cold requests, features of which can be missed. XGBoost helps eCache process these missing features, improving the accuracy of eCache. Thus, XGBoost is the optimal choice for the cache replacement.

4) Inference Procedure in eCache: The P-ML model receives the features of each sampled page and then labels them accordingly. eCache employs a pipelining technique (see Fig. 10) between the sampling and labeling of the inference candidates to accelerate the inference. Labeling operations are triggered after the ML module samples a candidate in the arrival linked list based on the sampling rule (see §IV-C2). Meanwhile, the ML module searches for the next candidate. Notably, eCache does not perform inference on pages that have been labeled. When their labels are reset, these pages can be involved in the sampling. The model performs inference during candidate sampling without waiting for all candidates to be extracted. Thus, eCache reduces the impacts of sampling inference candidates on cache performance.

5) Overheads of eCache: Now, we measure the time overhead of eCache. We configure the XGBoost model with a max_depth of 5 for inference. The inference and training time

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

for a page are within 3μs and 30s, respectively. Existing ML-based cache schemes [11] collect features and inference for labels before pages enter the cache; however, eCache parallelizes cache replacement with inference. Consequently, eCache's inference time does not impact cache performance. We also investigate the space overhead of eCache. eCache's model size is below 340 KB, and the size of features in each page is smaller than 28 B, which is acceptable for SSDs [32].

C. Cache Management in eCache

In this section, we delineate each module and the decision-making process in eCache.

1) Parallel Framework: Existing ML-based caching schemes exhibit strict sequential dependency: user I/Os must await model inference and cold-hot labeling before cache admission (Fig. 5(b)), exacerbating I/O blocking under high concurrency due to inference latency. The inter-page time interval is smaller than the inter-request time interval due to the large requests. Therefore, the P-ML cache schemes blocks more I/Os than the R-ML cache schemes.

To solve this issue, eCache eliminates this bottleneck via a parallel framework (Fig. 10): I/O requests gain immediate cache admission through an LRU-like mechanism, while model inference and cache management(e.g., data update) execute concurrently. This design eliminates waiting through parallelization, maintaining low latency without introducing additional blocking under high-concurrency workloads.

In Fig. 10, we divide the cache management into three stages: cache replacement, data migration, and data update, while the model inference was divided into candidate sampling and labeling. Between time t_i and t_{i+1}, data migration, candidate sampling and labeling can be performed in parallel. Data update, candidate sampling, and labeling are carried out in parallel between time t_{i+1} and t_{i+2}. Similarly, the cache replacement, candidate sampling, and labeling can be parallelized between time t_{i+2} and t_{i+3}. Thus, the ML module can perform multiple model inferences between two consecutive cache replacements, providing labeled pages (HOT, WARM, COOL and COLD) for the cache replacement.

Notably, when a cache replacement occurs, eCache preferentially evicts a page labeled as COLD, followed by COOL and WARM. If there are no these pages, the cache replacement module evicts a page labeled as HOT or NONE in UP rather than wait for the above three types of pages produced from the ML inference. As a result, the inference does not interfere with the cache replacement while the pages are sampled from the cache and labeled. In practice, the interval between two consecutive cache replacements is unequal; thus, we configure an inference hard threshold (*Iht*) to simulate a parallel framework for the cache management and model inference in eCache. Then, we have

$$\text{Iht} = \frac{\text{Average_replace_interval}}{\text{Average_model_inference_time}}, \tag{2}$$

where *Average_replace_interval* denotes the average time interval between two consecutive cache replacements in the workload in the LRU manner. *Average_model_inference_time* is the average inference time of the ML model. *Iht* is the maximum number of inferences on average between two consecutive cache replacements. eCache can perform the *Iht* inferences between two consecutive cache replacements, which simulates the parallel framework.

Table IV lists the Iht of workloads. Iht is high due to the small time overhead of the model. However, to reduce computational overhead, we set a soft threshold for inference between two consecutive cache replacements according to the cache size. We also set the maximum value of Iht to 215.

2) Sampling Rules: In §III-B, the inference time can be longer than the inter-request time interval in a workload. It is challenging to perform inferences for all pages in real time. Thus, we sample candidates for the inference to reduce the computational and time overhead in eCache. It is crucial to build a sampling rule to select the inference candidates. When extracting the candidates one by one, eCache faces the challenge of pages waiting before inference, when processing large-sized requests. Notably, large-sized requests are prevalent in workloads [10], [33].

The access patterns of pages in a request are similar. If these pages are re-accessed shortly, their inference can consume computational resources but cannot obtain labels COLD, COOL, and WARM. Therefore, these sampled inference candidates should be featured with diverse access patterns. In prior work [26], pages with close arrival times have similar access patterns. To achieve the sampled candidates with diverse access patterns, we create a directory arrival linked list to store the LPA of pages in the cache, ordered by the arrival time. The arrival linked list divides the directory into *n* groups. We select an inference candidate per group. We select one inference candidate per group. The resulting n candidates are labeled by eCache through per-round inference, ensuring no consecutive candidates originate from the same group. We randomly sample pages rather than select a fixed portion of pages in a group. We refer to this sampling rule as random-group sampling. This sampling rule first avoids selecting inference candidates with an access pattern in a one-by-one manner. Second, random sampling in a group selects all pages with an equal probability, which prevents eCache from capturing a fixed portion of pages in the workload.

To reduce the computational overhead, we set an inference soft threshold (*Ist*) for a round inference as

$$\text{Ist} = \begin{cases} 16 & \text{if } x < 16 \\ x & \text{if } 16 \le x \le 64 \\ 64 & \text{if } x > 64 \end{cases} \tag{3}$$

which describes the random-group sampling rule in eCache. We use *Ist* and *Iht* to construct the sampling rule which takes 128 pages as a group and randomly selects one per group. The small-sized cache yields few groups, leading to few inference candidates, which prevents the ML module from performing reference on pages labeled as COLD, COOL, and WARM; therefore, the PL and CL become empty during cache replacement. Thus, we set a minimum number of inference operations, where the group size is adjusted to x/16 MB. When the cache size is large, many inferences introduce high computational

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

TABLE IV
HARD INFERENCE THRESHOLD OF ECACHE

Workloads	LUN0-1	LUN1-1	LUN1-2	LUN2-1	LUN2-2	LUN2-3	LUN3-1	hm_0	prn_1	proj_0
Average replace interval (μs)	1421	5196	679	707	40524	696	845	226400	92992	32457
Iht	451	1649	215	224	12864	220	268	71873	29521	10303

overhead. We set *Ist* for a round inference. When there are no pages with the label COLD after a round inference, eCache continually performs random-group sampling until it performs the *Iht* number of inferences. When PL is not empty, eCache halts inference to reduce overhead.

3) Design of UP, PL, and CL: In eCache, we design linked lists PL, CL, and PL for the ML module. The PL stores the pages migrated from the UP because they are labeled as COLD and COOL. The CL stores the pages labeled as WARM. In a cache replacement, eCache evicts pages from the PL, CL, and UP in an LRU manner. eCache does not terminate inference even though it predicts a COLD page before the cache replacement. The ML module continues inferring to enrich the PL and CL. This method has two advantages. First, the PL and CL store COLD, COOL, and WARM pages which are preferentially evicted in the cache replacement. This prevents pages with the label HOT or NONE in UP from being evicted. Second, PL and CL collaborate with the active write-back module to proactively write dirty cache pages to flash memory. The dirty pages in PL identified as COLD or COOL pages are the optimal candidates for active write-back. When the PL is empty, eCache prioritizes dirty pages from the CL for active write-back to reduce response time.

4) ML-Aware Active Write-Back Module: In a cache replacement, an evicted clean page is removed from the cache, whereas evicting a dirty page requires writing it back to flash memory. The time overhead for a program (write) is much higher than that of a read. The high time overhead of evicting a dirty page worsens I/O blocking and response time. eCache combines the active write-back module with an ML module, and the former writes the dirty pages of the PL back to flash memory during idle periods. When the PL is empty, eCache actively writes back the dirty pages in the CL. Since the active write-back module only involves COLD, COOL, and WARM pages, eCache lessens the negative impact of active write-backs while reducing the response time.

D. Decision-Making of eCache

We present the decision-making process in eCache. Before entering the cache, a request is sliced into pages, which are inserted into the head of the UP with the label NONE. LPAs of pages in the arrival linked list are sequentially updated based on the pages' arrival time. Once a page enters the cache, the ML module collects features. The LRU-based cache replacement in the UP does not interfere with the sampling and labeling. According to Equ. 3, we sample up to 64 inference candidates between two consecutive cache replacements named a round inference according to the cache size. If the PL is still empty after a round inference, eCache performs inferences until it approaches *Iht*. The sampling and labeling are performed in the pipeline to avoid stalls in the labeling. If PL is not empty, eCache terminates inference to reduce computational overhead.

WARM pages are migrated to the CL. COOL pages are inserted into the head of the PL. COLD pages are placed at the tail of the PL. The label of each page can be reset upon a cache hit, enabling it to be selected as a candidate for inference. In a cache replacement, eCache first evicts pages from PL's tail. When the PL is empty, eCache evicts pages from the CL in an LRU manner. If both PL and CL are empty, eCache evicts pages from the UP following the LRU. In addition, when the flash memory is idle, the active write-back module writes back the dirty pages in the PL and CL, producing a large number of clean and cold pages. The replacement costs of clean pages are lower than those of dirty pages, thereby reducing the average response time of eCache.

TABLE V
CONFIGURATION PARAMETERS OF MQSIM

SSD Organization	Capacity: 512 GB, 8 channels, 4 chips/channel
Flash Micro-architecture	8-KB page, 256 pages/block 2048 blocks/plane, 2 planes/die, 2 dies/chip
Flash latency	Read: 50μs, Program: 750μs, Erase: 3ms.
FTL	GC policy: greedy [34], Mapping scheme: DFTL [30], TSU: Sprinkler [35]

V. EXPERIMENTAL RESULTS

Experimental setups. We implement eCache in MQSim and compare it with LRU, LAC [5], VS-batch [7], NCache [12], and CCache [11]. MQSim provides a scientific simulation of behaviors of SSDs and supports the latest NVMe protocol. Table V lists the experimental setups. To study the performance of eCache, we utilize 10 traces from SNIA [10], [24] and eight cache sizes. The model training for eCache is performed through Python Scikit-learn.

Modern SSD simulators trace system calls to quantify latency and hit rate, avoid physical media wear, and enable rapid, repeatable large-scale experiments for analyzing I/O workloads and cache policies. However, they introduce 6% to 8% error compared to real MLC SSDs due to simplified invalid/valid page-distribution models. Real SSDs offer true performance and support emerging technologies such as PCIe-DRAM and 3D XPoint but incur wear and lack repeatable initial states across runs. We adopt MQSim because: (1) It accurately models modern multi-queue SSDs. (2) Its efficient preprocessing and auto-configuration reduce setup time and eliminate hardware-specific variability. (3) It captures full end-to-end latency, quantifies ML-module cache overhead, and evaluates backend resource impacts.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

Performance Metrics. In the following, we outline the metrics used to evaluate the performance of eCache. **Average response time** represents the average time interval from the input of an I/O request into the submission queue (SQ) until the completion of SSD processing and the feedback to the host via the completion queue (CQ). This metric measures the performance of a cache scheme. **Write amplification (WA)** is the ratio of the amount of data written to the flash memory to the amount of user data written on the host. **Erase count** is a measure of the lifetime of a flash block in an SSD. **Write/Read hit ratio** indicates the proportion of read/write I/Os that are serviced by the cache. We normalize data to avoid information loss from scale differences (e.g., μs vs. thousands of μs), highlighting trends and improving readability. This suits non-negative data, emphasizes relative magnitudes, and aids multi-scale dataset overlays (e.g., 10 sub-plots in baseline experiments). Each dataset is linearly mapped to [0, 1]: we find the maximum value MM per workload and divide all values by MM. The mapping uses the global maximum across cache sizes and policies. As eCache uses LRU (straightforward to implement), normalization is intuitive. In some smaller-scale extended experiments, raw values are shown since absolute magnitudes do not compromise plot visibility.

A. Performance Evaluation

1) Average Response Time: Fig. 11 shows the normalized average response time of eCache with various cache sizes. eCache demonstrates its superior performance over most alternative cache schemes. For a 2-MB cache under LUN1-1, eCache lowers average response time by 64.31%, 60.48%, 64.21%, 64.68%, 60.97%, and 63.71% compared with LRU, LAC, VS-batch, ARC, NCache, and CCache, respectively. Similarly, with a 256-MB cache, eCache reduces response time by 66.77%, 58.07%, 59.99%, 69.24%, 43.07%, and 43.86% compared to the same alternative cache schemes. Under LUN1-2, eCache reduces the average response time by 63.65%, 63.60%, 66.36%, 66.34%, 55.67%, and 62.37% compared with LRU, LAC, VS-batch, ARC, NCache, and CCache, respectively, as the cache size is set to 8 MB.

eCache employs an XGBoost-based ML model to differentiate between hot and cold data in workloads, enhancing its adaptability to diverse access patterns. By learning from the OPT, eCache accurately predicts the future access patterns for each page, thereby lowering the average response time of LAC and VS-batch. Compared to NCache and CCache, eCache mitigates the time overhead of ML models through a parallel framework. It also incorporates an active write-back strategy to convert cold and dirty pages into clean ones, which reduces cache-miss cost and average response time.

2) SSD Lifetime: NAND flash has a limited lifespan, with WA and erase count being key parameters in evaluating SSD lifetime, see Figs. 12 and 13. Under LUN1-2, the 4-MB-cache eCache reduces WA by 22.15%, 39.97%, 30.85%, 30.50%, 6.90%, and 20.92% compared to LRU, LAC, VS-batch, ARC, NCache, and CCache, respectively. When the cache size is 8 MB, eCache reduces the erase count by 25.99%, 59.16%,

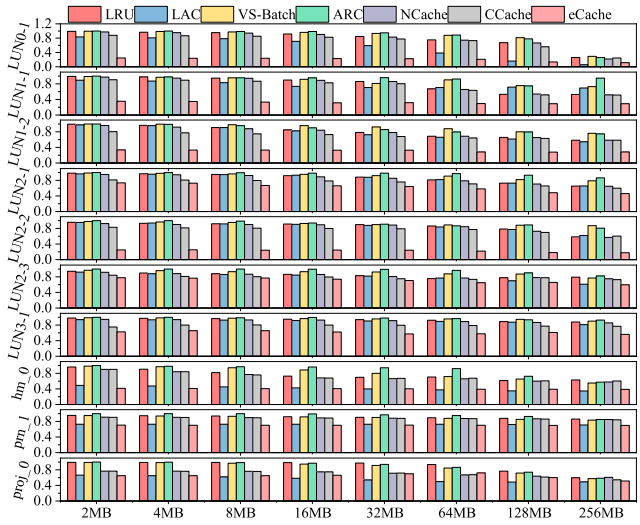


Fig. 11. Normalized average response time.

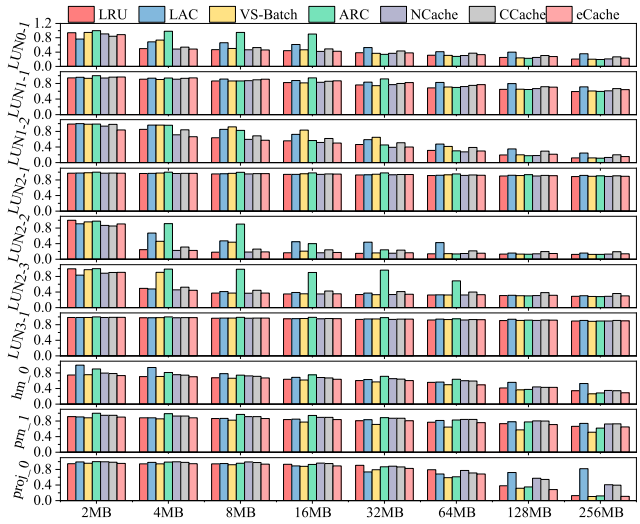


Fig. 12. Normalized write amplification.

68.33%, 75.60%, 48.11%, and 53.26% compared to these cache schemes under LUN2-2. Particularly with large-sized cache, eCache shows a lower WA and erase count than other schemes with active write-back. For example, under proj_0, with a 128-MB cache, eCache reduces WA by 25.68%, 60.83%, 10.31%, 19.80%, 50.81%, and 47.85% compared to LRU, LAC, VS-batch, ARC, NCache, and CCache, respectively. LAC exhibits the highest WA due to its tendency to actively write back more data when the NAND flash is idle, which increases WA and erase count. In contrast, eCache prioritizes writing back dirty data from the PL and adopts a conservative write-back strategy with a large-sized cache, thereby reducing WA and erase count.

3) Hit Ratio: Figs. 14–16 present the hit ratio, write hit ratio, and read hit ratio, respectively. ML-based cache schemes demonstrate a high hit ratio. NCache uses a P-ML model and a full inference approach, demonstrating high performance.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

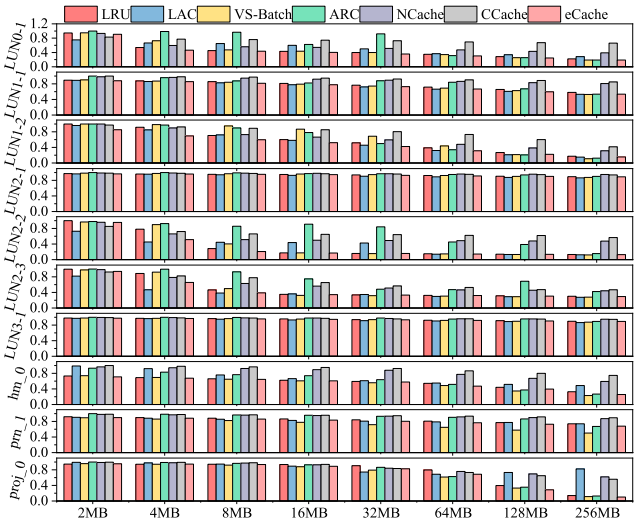


Fig. 13. Normalized erase count.

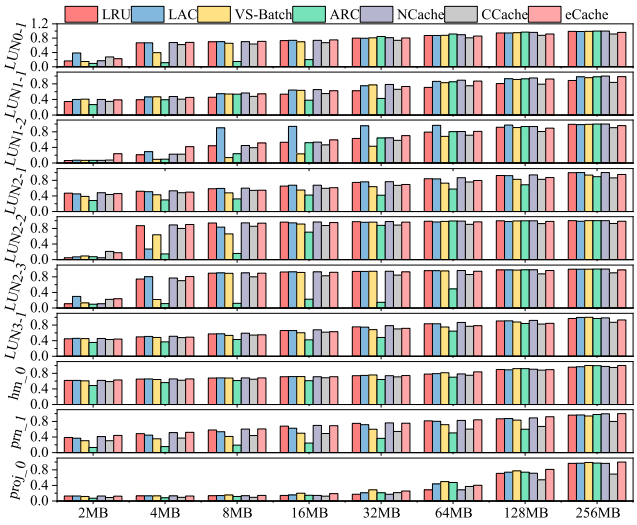


Fig. 15. Normalized write hit ratio.

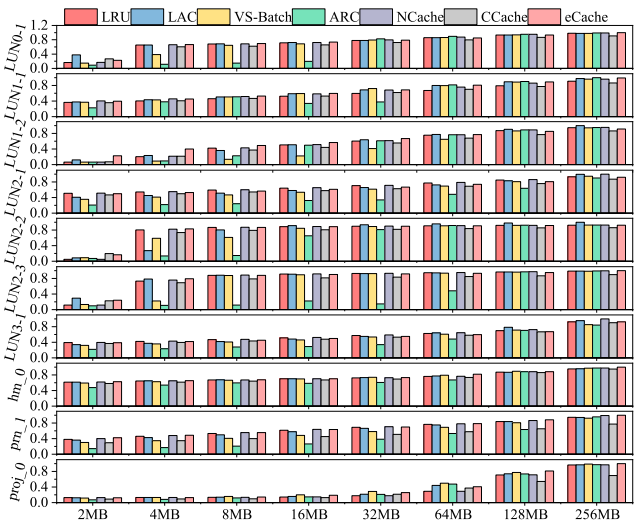


Fig. 14. Normalized hit ratio.

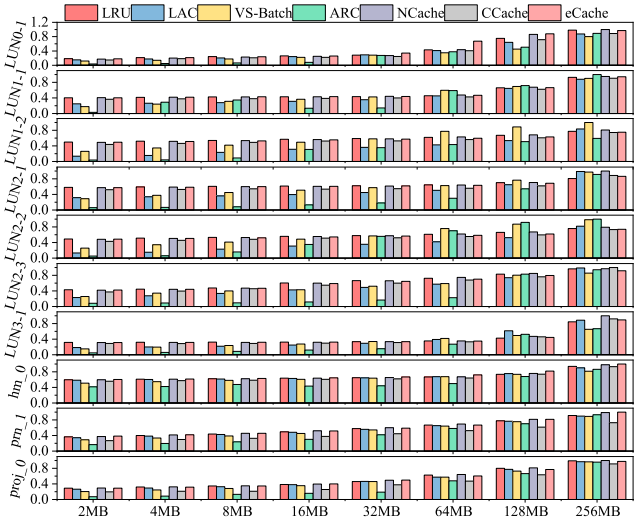


Fig. 16. Normalized read hit ratio.

In contrast, eCache reduces computational overhead through sampling inference. Additionally, eCache utilizes random-group sampling to obtain sufficient cold pages for hot-cold separation, even with fewer inferences. For the 2-MB cache under prn_1, eCache's hit ratio surpasses that of LRU, LAC, VS-batch, ARC, NCache, and CCache by 10.81%, 16.73%, 40.43%, 193.30%, 5.73%, and 44.32%, respectively. Under LUN1-2, eCache with a 32-MB cache improves the hit ratio of LRU, LAC, VS-batch, ARC, NCache, and CCache by 10.9%, 4.97%, 63.12%, 9.05%, 8.64%, and 20.63%, respectively. Under proj_0, the 128-MB-cache eCache improves the hit ratio by 14.36%, 9.95%, 4.88%, 10.19%, 14.12%, and 49.40%, respectively, over the same alternative cache schemes.

Thus, eCache outperforms other cache schemes even with sampling inference because of the effectiveness of eCache's label used in the training dataset. Prior work [36] unveiled that the access patterns of workload vary with cache sizes. This is because the large-sized cache can retain some pages that would be evicted in the case of a small-sized cache, allowing cold pages to become hot. To obtain these changes, eCache associates labels with cache sizes, enabling a page's label to adapt to the cache size. In contrast, NCache and CCache assign cache-independent labels, which are fixed regardless of cache sizes. eCache's adaptive labeling approach increases the cache hit ratio by dynamically adjusting labels to match cache sizes.

B. Extended Studies

1) Tail Latency: we measure the tail latency of eCache under various workloads. Fig. 17 shows that eCache exhibits lower tail latency under all workloads. With a 2-MB cache, eCache reduces the tail latency (P99) of LRU, LAC, VS-batch, NCache, and CCache by 61.21%, 55.37%, 65.50%, 61.64%, and 62.28%, respectively, under LUN1-2. Under LUN1-1, eCache with a 128-MB cache lowers P99 latency by 63.47%,

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

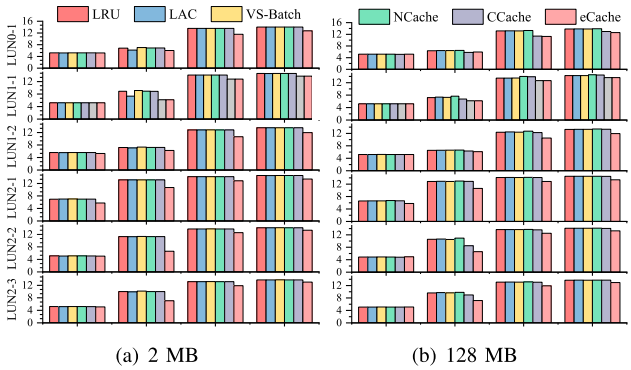


Fig. 17. Tail latency in cache schemes. The y-axis represents the logarithm of tail latency (μs).

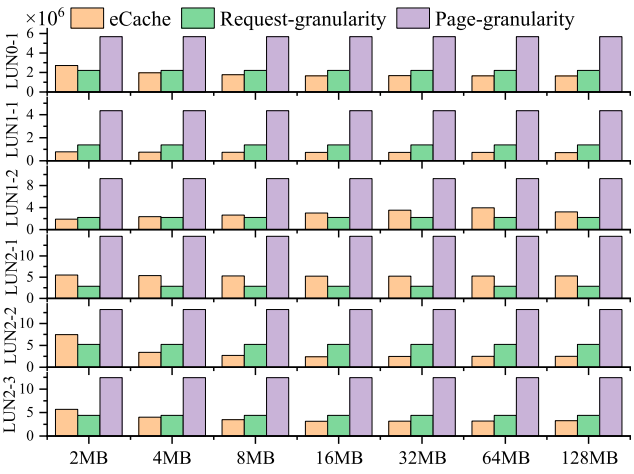


Fig. 18. The inference computational overhead of eCache with P-ML cache schemes and R-ML cache schemes.

42.16%, 66.67%, 68.90%, and 77.36%, compared to the same alternative cache schemes. This validates the effectiveness of our eCache. Through the random-group sampling for inference, eCache can identify sufficient cold pages. Subsequently, the active write-back module converts dirty pages among cold pages into clean ones, reducing cache replacement costs.

2) Computational Overhead: we refer to the number of model inference operations as the inference computational overhead of eCache. Fig. 18 shows the computational overhead of eCache, compared with the P-ML and R-ML cache schemes. Although eCache uses a P-ML model, its computational overhead is, at most, merely 59.39% of that of other P-ML cache schemes. Under LUN1-1, eCache's computational overhead is 16.67% of P-ML cache schemes on average. Compared to R-ML cache schemes, eCache reduces the inference overhead. Under LUN2-3 with a 128-MB cache, eCache's computational overhead is 74.18% of that of the R-ML cache scheme. In the worst case, eCache's computational overhead is twice that of the R-ML cache schemes but still significantly lower than that of the P-ML cache schemes.

Unlike the ML-based cache schemes leveraging full inference, eCache reduces computational overhead through sampling inference. Furthermore, its random-group sampling method ensures sufficient cold pages and few model inference operations. This approach reduces the impact of the sampling inference on the response time and hit ratio, improving its overall performance.

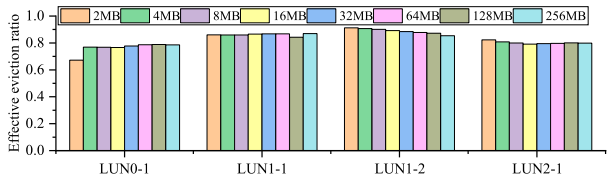


Fig. 19. Effective eviction ratio of eCache under workloads.

3) Effective Eviction Ratio of eCache: eCache uses an XGBoost-based model to classify sampled pages as label HOT, WARM, COOL, or COLD, prioritizing these pages with COLD, COOL, and WARM for eviction during cache replacement. In Equ. 1, eCache's labels are associated with cache size, meaning that the number of pages labeled as COLD by eCache varies with cache sizes under a workload. A small-sized cache causes more pages with the COLD label and frequent cache replacements. Conversely, with a large-sized cache, fewer pages can be labeled as COLD, and few frequent cache replacements are triggered. To achieve a target effective eviction ratio across various cache sizes, we adjust the group size and Ist for each cache size in eCache (see §IV-C2).

Fig. 19 shows that eCache achieves a high effective eviction ratio in workloads, and the values do not show significant fluctuations as the cache sizes change. Under LUN1-1, the effective eviction ratio of eCache is above 85%, while the ratio ranges from 67.18% to 78.61% under LUN0-1. The high effective eviction ratio validates the random-group-sampling approach in eCache. Thus, eCache reduces the computational overhead (see §V-B2), while maintaining a high effective eviction ratio. Then, a sufficient number of cold pages are available as evicted candidates in the cache replacement.

4) Comparison of eCache and OPT: eCache adopts sampling inference to reduce computational overhead. To evaluate the effectiveness of this approach, we compare eCache's hit ratio with that of OPT. Since OPT has access to the true label of each page, it represents a model with an accuracy of 100%, where all pages are correctly labeled and sorted into the responding linked lists in the cache. In Fig. 20, eCache's hit ratio closely approaches that of OPT under most workloads. Under LUN1-2, eCache's average hit ratio is 8.99% lower than OPT across all cache sizes, validating eCache's hot-cold classification. Through the random-group sampling inference, eCache reduces computational overhead while achieving a high hit ratio close to OPT, thereby justifying the trade-off.

5) The Significance of Features in eCache: we investigate the impact of features used in eCache on model accuracy.

Table VI shows that removing any feature will lead to a decrease in model accuracy. Removing R[1] largely decreases the accuracy, with a decline of 14.88%. Removing inherent features (e.g., request size) slightly decreases the model accuracy (within 10%). However, since the space overhead of these

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

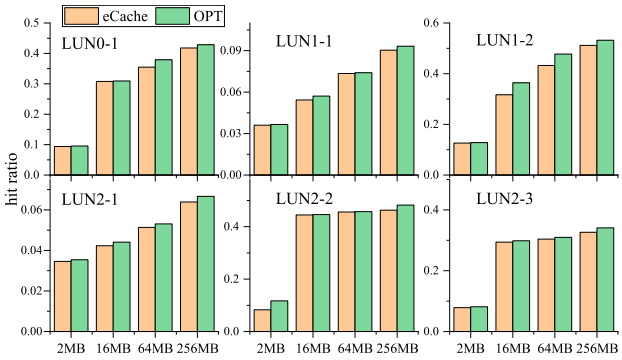


Fig. 20. The cache hit ratio of eCache and OPT.

TABLE VI
THE IMPACT OF FEATURE SETS ON ACCURACY

Feature set	F1	F2	F3	F4	F5	F6
Accuracy (%)	78.94	75.67	78.47	79.80	82.13	81.18

Note: Symbols F1, F2, F3, F4, F5, F6 represent feature sets that lacking R[0], R[1], R[2], frequency, page size and request size, respectively.

TABLE VII
THE FEATURE IMPORTANCE OF ECACHE

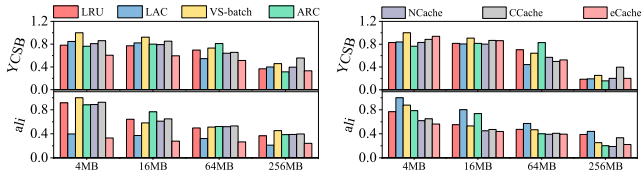
Feature	R[0]	R[1]	R[2]	Freq	Sp	Sr
Importance	0.3571	0.3264	0.0612	0.0899	0.1245	0.0408

Note: Symbols Freq, Sp, Sr represent frequency, page size and request size, respectively.

inherent features is small and the acquisition is simple, it is essential to employ them in the feature set. Therefore, we decide to use all these features in eCache. We also provided an analysis on the importance of features through calculating the Gain metric during model training to quantify the contribution of each feature to the prediction outcomes, thereby precisely evaluating a feature's independent impact on the model. In XGBoost, Gain is the default feature-importance metric, defined as the average reduction in the training loss from all tree splits that use a given feature, thereby quantifying how much that feature contributes to model performance. As shown in Table VII, using the XGBoost Gain importance evaluation framework, R[0] represents above 30% of the total importance of the feature, and it is the most important feature. These inherent features, such as request size show a limited impact (total contribution $< 10\%$).

6) Performance Under YCSB and AliCloud Traces: Based on our analysis (see Table I and Fig. 6), there is a significant proportion of requests with short intervals in the MSR and VDI traces. These short-interval phases effectively simulate the high-concurrency access scenarios faced by SSDs, making them valuable for experiments on SSD caching policies. Furthermore, we supplement the evaluation with Ali and YCSB traces to validate eCache's efficacy under modern, SSD-targeted workloads.

As shown in Fig. 21, eCache consistently outperforms all baseline cache schemes across both workloads. Under the



(a) Normalized Avg. RT. (b) Normalized WA.

Fig. 21. Normalized average response time (a) and WA (b) of different schems under YCSB and AliCloud traces.

YCSB trace, the 4-MB-cache of eCache reduces average response time by 22.47%, 28.43%, 39.36%, 20.60%, 25.02%, and 29.53% compared to LRU, LAC, VS, ARC, NCache, and CCache, respectively. For ali-152, the corresponding reductions are 63.87%, 16.82%, 66.91%, 62.47%, 62.72%, and 64.21%. eCache leverages an XGBoost-based model for hot/cold data classification, enhanced by the OPT-inspired learning and parallel framework.

In terms of WA, for the 64-MB cache under YCSB, eCache reduces WA by up to 36.6%, 25.4% compared to ARC and LRU. With a 256-MB cache, it achieves further reductions of 49.40% and 20.50% compared to CCache and VS. Under ali-152, the 4-MB-cache reduces WA by 35.90% and 9.10% compared to VS-batch and NCache, respectively, and by 40.50% relative to ARC at 16-MB. By prioritizing the proactive write-back of PL dirty data, eCache employs a conservative write-back strategy in large-cache scenarios to cut back its WA.

VI. RELATED WORK

A. ML-Based Cache Schemes

ML-based cache schemes mostly include C-ML, R-ML, and P-ML ones according to the training granularity.

1) Coarse Granularity: In C-ML cache schemes, the size of a sample granularity exceeds that of a request (e.g., GL-cache [26] and Cacheus [36]). GL-cache groups objects by similar write time and employs a group-utility function for ranking these groups. During cache replacement, it evicts the lowest-utility group based on model predictions, mitigating excessive inference overhead. However, high I/O intensity can increase the load on the flash memory due to bulk group eviction. Cacheus divides workloads into four types using two experts. Based on the hit ratio, it switches between experts via regret minimization to maximize benefit. However, online reinforcement learning requires frequent host-device communication, which introduces additional latency and is challenging for resource-limited SSDs. Furthermore, the experts determine Cacheus' performance, and it cannot handle access patterns beyond the capabilities of these experts.

2) Request Granularity: R-ML cache schemes utilize the R-ML model to assist with cache replacement. For example, RC-RNN [37] enhances cache accuracy by employing a multi-model structure that first classifies workloads and then makes decisions for requests based on their categories. Unluckily, more models result in higher computational and spatial overhead. The authors of [38] strive to train the model with the

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

TABLE VIII
COMPARISON OF ML-BASED CACHING SCHEMES

Caching Schemes	Prediction Targets	Model Types	Task Types	Ea	Tcc	Ttc
CCache	frequency	DT	Bc	✓	×	×
Neache	Rd	RF	Mc	✓	×	×
GLCache	Group utility	XGBoost	Reg	×	✓	✓
LRB	Rd	GBM	Reg	✓	×	×
Cacheus	Experts	Rm	RL	×	✓	✓
eCache	Rd	XGBoost	Mc	✓	✓	✓

Note: Symbols Ea, Tcc, Ttc, Bc, Rd, Mc, Reg, Rm, and RL represent eviction accuracy, total inference computational cost, total inference time costs, binary classification, reuse distance, multi-classification, regression, regret minimization, and reinforcement learning, respectively.

most critical features, aiming to achieve low overhead and high accuracy. However, this work fails to consider the impact of cache size on request access patterns.

3) Page Granularity: P-ML cache schemes (*e.g.,* NCache [12], CCache [11]) assign labels to each page in a workload. NCache constructs a decision-tree model to predict whether data is re-accessed before it is evicted, prioritizing the eviction of data unlikely to be re-accessed in the cache. CCache, on the other hand, uses a random forest to classify data into hot, warm, and cold ones, and constructs a multi-level linked list in the cache to evict cold, warm, and hot data sequentially.

In Table VIII, eCache againsts representative state-of-the-art ML-based caching schemes in terms of prediction targets, model types, task types, eviction accuracy, total inference computational cost, and total inference time. Table III reveals that existing cache schemes fail to simultaneously optimize all factors. For instance, CCache delivers high eviction accuracy, but cannot mitigate the high total inference computational cost associated with its high-accuracy models. GLcache addresses the computational cost issue but suffers from compromised eviction accuracy. Notably, eCache achieves high eviction accuracy while significantly reducing both computational and time costs introduced by ML modules in SSDs.

B. Traditional Heuristic Cache Schemes

VS-Batch [7] uses a visibility graph to unify I/O access's spatiotemporal locality. LAC [5] exploits spatial locality by writing back pages with adjacent or nearby logical addresses in parallel, improving the clean data conversion rate during short request intervals. Co-Active [8] employs a chip-level monitor to detect flash chip status, allowing its write-back module to write cold and dirty data in the cache to flash memory, thereby creating clean data and prioritizing its replacement. GCaR [21] and LCR [39] incorporate SSD back-end flash status into cache schemes to alleviate conflicts between external and internal I/Os. Unfortunately, traditional heuristic cache schemes largely depend on I/Os spatiotemporal locality and fixed access patterns; thus, the performance can fluctuate during low-locality phases or when access patterns change.

Unlike traditional heuristic cache schemes, eCache utilizes an XGBoost-based ML model to assist the cache schemes in the hot-cold separation. Thanks to the P-ML model's precise classification of each page, eCache delivers superior performance. Additionally, eCache combines the ML-aware active write-back mechanism with an ML module to reduce write amplification and response time.

VII. CONCLUSION

We developed a sample-inference-based intelligent cache scheme, eCache, which aims to reduce the overhead caused by the ML module in SSDs. In particular, we implemented a page-granularity XGBoost model to differentiate between hot and cold pages. We designed a parallel framework to eliminate the negative impact of inference time on cache performance. We devised a random-group-sampling method to reduce computational overhead. After the hot-cold separation, eCache employs an ML-aware active write-back module to convert cold and dirty pages into clean pages, which cuts back cache replacement costs. We implemented eCache in MQSim and conduct extensive experiments to evaluate the performance of eCache, compared with the state-of-the-art cache schemes. The results confirm that eCache shortens the average response time by 44.08% with a maximum improvement of up to 79.68%. In addition, eCache scales back computational overhead by an average of 66.00% and up to 85.23% compared to the page-granularity-based cache schemes. Future work may explore more advanced batching strategies for model training and validation on real hardware platforms like OpenSSD.

REFERENCES

[1] Z. Yang, et al., "Autotiering: automatic data placement manager in multi-tier all-flash datacenter," in *Proc. 36th Int. Perform. Comput. Commun. Conf. (IPCCC)*, 2017, pp. 1–8.
[2] B. S. Kim, J. Choi, and S. L. Min, "Design tradeoffs for SSD reliability," in *Proc. Fast*, 2019, pp. 281–294.
[3] M. Tarihi, H. Asadi, A. Haghdoost, M. Arjomand, and H. Sarbazi-Azad, "A hybrid non-volatile cache design for solid-state drives using comprehensive i/o characterization," *IEEE Trans. Comput.*, vol. 65, no. 6, pp. 1678–1691, Jun. 2016.
[4] A. Dan and D. Towsley, "An approximate analysis of the LRU and FIFO buffer replacement schemes," in *Proc. ACM SIGMETRICS Conf. Meas. Modeling Comput. Syst.*, 1990, pp. 143–152.
[5] H. Sun, H. Tong, Y. Yue, and X. Qin, "LAC: A workload intensity-aware caching scheme for high-performance SSDs," *IEEE Trans. Comput.*, vol. 73, no. 7, pp. 1738–1752, Jul. 2024.
[6] N. Megiddo and D. S. Modha, "ARC: A self-tuning, low overhead replacement cache," in *Proc. Fast*, 2003, vol. 3, pp. 115–130.
[7] Z. Sha, Z. Cai, F. Trahay, J. Liao, and D. Yin, "Unifying temporal and spatial locality for cache management inside SSDs," in *Proc. Des., Automat. Test Europe Conf. Exhib. (DATE)*, 2022, pp. 891–896.
[8] H. Sun, S. Dai, J. Huang, and X. Qin, "Co-active: A workload-aware collaborative cache management scheme for NVMe SSDs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 6, pp. 1437–1451, Jun. 2021.
[9] H. Sun, S. Dai, J. Huang, Y. Yue, and X. Qin, "DAC: A dynamic active and collaborative cache management scheme for solid state disks," *J. Syst. Archit.*, vol. 140, Jul. 2023, Art. no. 102896.
[10] Systor'17 traces. Accessed: May 21, 2024. [Online]. Available: http://iotta.snia.org/traces/block-io/4928
[11] H. Sun, C. Sun, H. Tong, Y. Yue, and X. Qin, "A machine learning-empowered cache management scheme for high-performance SSDs," *IEEE Trans. Comput.*, vol. 73, no. 8, pp. 2066–2080, Aug. 2024.
[12] H. Sun, Q. Cui, J. Huang, and X. Qin, "NCache: A machine-learning cache management scheme for computational SSDs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 6, pp. 1810–1823, Jun. 2023.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.

[13] W. Liu, J. Cui, J. Liu, and L. T. Yang, "MLCache: A space-efficient cache scheme based on reuse distance and machine learning for NVMe SSDs," in *Proc. 39th Int. Conf. Computer-Aided Des.*, 2020, pp. 1–9.
[14] H. Wang, X. Yi, P. Huang, B. Cheng, and K. Zhou, "Efficient SSD caching by avoiding unnecessary writes using machine learning," in *Proc. 47th Int. Conf. Parallel Process.,* 2018, pp. 1–10.
[15] G. Vietri et al., "Driving cache replacement with ML-based LeCaR," in *Proc. 10th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, 2018, pp. 1–6.
[16] Z. Song et al., "Learning relaxed belady for content distribution network caching," in *Proc. 17th USENIX Symp. Netw. Syst. Des. Implementation (NSDI)*, 2020, pp. 529–544.
[17] D. Li, H. Sun, and X. Qin, "iCache: An intelligent cache allocation strategy for multi-tenant in high-performance solid state disks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 44, no. 5, pp. 1690–1703, May 2024.
[18] M. Cho and D. Kang, "ML-CLOCK: Efficient page cache algorithm based on perceptron-based neural network," *Electronics*, vol. 10, no. 20, 2021, Art. no. 2503.
[19] Y. Zhang et al., "A machine learning based write policy for SSD cache in cloud block storage," in *Proc. Des., Autom. Test Europe Conf. Exhib. (DATE)*, 2020, pp. 1279–1282.
[20] A. Tavakkol, J. Gómez-Luna, M. Sadrosadati, S. Ghose, and O. Mutlu, "MQSim: A framework for enabling realistic studies of modern multi-queue SSD devices," in *Proc. 16th USENIX Conf. File Storage Technol. (FAST)*, 2018, pp. 49–66.
[21] S. Wu, Y. Lin, B. Mao, and H. Jiang, "GCaR: Garbage collection aware cache management with improved performance for flash-based SSDs," in *Proc. Int. Conf. Supercomput.*, 2016, pp. 1–12.
[22] Nytro 4350 SSD datasheet. Accessed: Jul. 21, 2025. [Online]. Available: https://www.seagate.com/content/dam/seagate/en/content-fragments/products/datasheets/enterprise-rebranding/nytro-4350/nytro-4350-SC29-1-2306US-en_US.pdf
[23] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper, "Workload analysis and demand prediction of enterprise data center applications," in *Proc. IEEE 10th Int. Symp. Workload Characterization*, 2007, pp. 171–180.
[24] "MSR Cambridge traces." SNIA. Accessed: May 21, 2024. [Online]. Available: http://iotta.snia.org/traces/388
[25] Q. Zou, Y. Zhu, J. Chen, Y. Deng, and X. Qin, "Characterization of I/O behaviors in cloud storage workloads," *IEEE Trans. Comput.*, vol. 72, no. 10, pp. 2726–2739, Oct. 2023.
[26] J. Yang, Z. Mao, Y. Yue, and K. V. Rashmi, "{GL-Cache}: Group-level learning for efficient and high-performance caching," in *Proc. 21st USENIX Conf. File Storage Technol. (FAST)*, 2023, pp. 115–134.
[27] Jerome, H. Saltzer, and M. F. Kaashoek, *Principles of Computer System Design: An Introduction.* San Mateo, CA, USA: Morgan Kaufmann, 2009.
[28] L. A. Belady, "A study of replacement algorithms for a virtual-storage computer," *IBM Syst. J.*, vol. 5, no. 2, pp. 78–101, 1966.
[29] J. Ren et al., "A machine learning assisted data placement mechanism for hybrid storage systems," *J. Syst. Archit.*, vol. 120, 2021, Art. no. 102295.
[30] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," *ACM Sigplan Notices*, vol. 44, no. 3, pp. 229–240, 2009.
[31] D. Nielsen, "Tree boosting with XGBoost-why does XGBoost win 'every' machine learning competition?" Master's thesis, NTNU, 2016.
[32] L. Shi, L. Luo, Y. Lv, S. Li, C. Li, and E. H.-M. Sha, "Understanding and optimizing hybrid SSD with high-density and low-cost flash memory," in *Proc. IEEE 39th Int. Conf. Comput. Des. (ICCD)*, 2021, pp. 236–243.
[33] C. Lee, T. Kumano, T. Matsuki, H. Endo, N. Fukumoto, and M. Sugawara, "Understanding storage traffic characteristics on enterprise virtual desktop infrastructure," In *Proc. 10th ACM Int. Syst. Storage Conf.,* 2017, pp. 1–11.
[34] W. Bux and I. Iliadis, "Performance of greedy garbage collection in flash-based solid-state drives," *Perform. Eval.*, vol. 67, no. 11, pp. 1172–1186, 2010.
[35] M. Jung and M. T. Kandemir, "Sprinkler: Maximizing resource utilization in many-chip solid state disks," in *Proc. IEEE 20th Int. Symp. High Perform. Comput. Archit. (HPCA)*, 2014, pp. 524–535.
[36] L. V. Rodriguez et al., "Learning cache replacement with CACHEUS," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, 2021, pp. 341–354.
[37] S. Ebrahimi, R. Salkhordeh, S. A. Osia, A. Taheri, H. R. Rabiee, and H. Asadi, "RC-RNN: Reconfigurable cache architecture for storage systems using recurrent neural networks," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 3, pp. 1492–1506, Jul./Sep. 2021.
[38] S. Azadvar, A. Tavakkol, H. Asadi, and H. Sarbazi-Azad, "Quick generation of SSD performance models using machine learning," *IEEE Trans. Emerg. Topics Comput.*, vol. 10, no. 4, pp. 1821–1836, Oct./Dec. 2021.
[39] C. Liu et al., "LCR: Load-aware cache replacement algorithm for flash-based SSDs," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, 2018, pp. 1–10.

Hui Sun received the Ph.D. degree from Huazhong University of Science and Technology, in 2014. He is a Professor of Computer Science with Anhui University. His research interests include computer systems, non-volatile memory-based storage systems, file systems, and I/O architectures.

Yinan Fu is currently working toward the M.S. degree with Anhui University. His research interests include SSD storage and intelligent cache management.

Xin Wang is currently working toward the M.S. degree with Anhui University. His research interests include SSD storage and intelligent cache management.

Yi Zhou received the B.E.E and M.S.E.E. degrees in electronic engineering from Beijing University of Technology, in 2006 and 2010, respectively, and the Ph.D. degree in computer science from Auburn University, in 2018. He is currently an Associate Professor with TSYS School of Computer Science, Columbus State University. His research interests include cloud computing, edge computing, database systems, distributed systems, and parallel computing.

Xiao Qin (Senior Member, IEEE) received the B.S. and M.S. degrees in computer science from Huazhong University of Science and Technology, in 1992 and 1999, respectively. He received the Ph.D. degree in computer science from the University of Nebraska-Lincoln, in 2004. He is currently an Alumni Professor and the Director of Graduate Programs with the Department of Computer Science and Software Engineering, Auburn University. He won an NSF CAREER Award in 2009. His research interests include parallel and distributed systems, storage systems, and performance evaluation.

Authorized licensed use limited to: Auburn University. Downloaded on November 18,2026 at 18:39:22 UTC from IEEE Xplore. Restrictions apply.