CacheBoost: Harnessing Machine Learning for Peak Cache Performance

Sharath Kumar Jagannathan,[1] Maheswari Raja*,[2] P. Vijaya[3], Reena Abraham[3]

[1] Data Science Institute, Frank J. Guarini School of Business, Saint Peter's University, New Jersery,
[2] Sri Eshwar College of Engineering, Coimbatore, Tamilnadu, India
deaninnovations@sece.ac.in*,
[3] Department of Mathematics and Computer Science, Modern College of Business and Science, Bowshar, Sultanate of Oman

Abstract. This research investigates the integration of machine learning (ML) models into cache management systems to enhance overall performance. Two distinct strategies, the Block Cache model and Vector Cache model, are implemented, each incorporating widely used cache replacement policies—Least Recently Used (LRU) and Least Frequently Used (LFU). Furthermore, three ML models—Logistic Regression, K-Nearest Neighbors (KNN), and Neural Network—are integrated into these cache systems. The primary goal is to improve the cache hit rate by combining ML models with Belady's Optimal algorithm. The performance of the five cache models is assessed using key metrics such as cache hit rate, miss rate, and eviction rate. A comparative analysis is undertaken to gauge the effectiveness of each approach and the influence of various ML models on cache performance. This study aims to provide valuable insights into the complex interaction between traditional cache replacement policies and advanced ML techniques, offering a nuanced understanding of the potential enhancements in cache hit rates achieved through machine learning integration. The findings and observations contribute to the ongoing exploration of cache optimization, guiding future developments to enhance system performance.

Keywords: Block cache model , vector Cache model, Beladys optimal algorithm, KNN , logistic regression

1 INTRODUCTION:

In the dynamic realm of computing, the optimization of cache performance stands as a critical aspect for ensuring efficient data access and system responsiveness. This study navigates the complexities of cache optimization by seamlessly integrating Machine Learning models within two distinctive methodologies: the Block cache method and the Vector cache method. Our primary goal is not only to elevate cache hit rates but also to unravel the intricate relationships between Machine Learning algorithms, caching strategies, and the dynamic responsiveness of systems.

N. Bacanin and H. Shaker (eds.), *Proceedings of the 2nd International Conference on Innovation in Information Technology and Business (ICIITB 2024)*,
Advances in Computer Science Research 113,
https://doi.org/10.2991/978-94-6463-482-2_2

The Block cache method zeros in on meticulous block-level optimization, employing predictive algorithms like Logistic Regression, KNN, and Neural Network to make judicious decisions about the specific blocks worthy of caching. Simultaneously, the Vector cache method adopts a holistic approach, optimizing cache at a higher level by considering relationships among different blocks and employing Machine Learning models for intelligent caching decisions [1].

This research embraces a comprehensive exploration of both methodologies, integrating two widely used paging/cache replacement algorithms, LRU and LFU, alongside three distinct Machine Learning models. A pivotal aspect of our investigation involves an in-depth comparison of the five models through graphical representations. Metrics such as cache hit rates, miss rates, and eviction rates undergo systematic analysis, providing a holistic understanding of each approach's performance across diverse workloads.

Belady's optimization algorithm assumes a pivotal role in this exploration. Through the analysis of block traces and cache sizes, it aids in determining optimal cache configurations based on historical data access patterns, setting the foundation for a more informed evaluation of the considered cache management strategies.

Additionally, we introduce a dynamic Machine Learning Policy, continuously assessing the system's state in real-time. This policy dynamically computes cache hit rates, eviction strategies, and miss rates, ensuring adaptive responsiveness to evolving workloads. The amalgamation of Machine Learning models, paging algorithms, and optimization techniques converges into a comprehensive approach to cache management.

In conclusion, this research aims to offer valuable insights into the integration of Machine Learning models and cache management techniques. The nuanced comparative analysis, supported by graphical representations, unveils the distinct performances of each approach, guiding the development of adaptive caching systems tailored for dynamic computing environments.

1.1 Overview and Cache Models Implementation

This research focuses on enhancing cache performance through two distinct approaches - the Block Cache Model and the Vector Cache Model. The Block Cache Model involves the development and optimization of a cache system managing data in fixed-size blocks, with the implementation of traditional cache replacement policies like Least Recently Used (LRU) and Least Frequently Used (LFU). Simultaneously, the Vector Cache Model utilizes vectorized operations to improve cache management efficiency. Both models aim to establish baseline performance metrics, including cache hit rate, miss rate, and eviction rate, forming the foundation for subsequent evaluations [2].

1.2 Machine Learning Integration and Belady's Optimal Algorithm

The research delves into the integration of machine learning (ML) models into cache replacement policies to enhance decision-making processes. Three ML

models—Logistic Regression, KNN, and Neural Network—are implemented and trained using historical cache access patterns to predict future access behaviors. These ML models are then seamlessly integrated with Belady's Optimal algorithm, forming a cohesive framework that combines the advantages of traditional algorithms with the predictive capabilities of machine learning. This integration aims to create a robust cache system that adapts dynamically to varying access patterns, ultimately improving cache hit rates.

1.3 Performance Evaluation and Documentation

The research concludes with a comprehensive performance evaluation of all implemented cache models and machine learning-enhanced systems. Experiments are conducted to compare cache hit rates, miss rates, and eviction rates across the Block and Vector Cache Models, incorporating both traditional and machine learning-driven cache replacement policies. Observations drawn from the results contribute to actionable recommendations for optimizing cache performance. The entire research lifecycle, including design, implementation, and evaluation processes, is documented thoroughly to provide valuable insights for future research and development in the realm of cache optimization.

2 LITERATURE SURVEY

The literature surrounding cache replacement policies is diverse, with researchers exploring innovative strategies to optimize system performance. This review we aim to examine the effectiveness and methods used.

The analysis employed a simulator that accurately depicts cycles to assess the effectiveness of the proposed hit-count based replacement policy in comparison to various other policies. The simulator was configured with a six-stage pipeline, a 256-entry ROB, 32 KB L1-D/I cache, a 256 KB private L2 cache, and a 2 MB shared LLC per core. Furthermore, the study utilized several benchmark workloads to appraise the performance of the policies [3].

The suggested hit-count based replacement policy employs a hit-count predictor to estimate the projected number of hits for each cache block. This predictor is founded on the block's reuse distance, which is calculated by utilizing a stride prefetcher. The policy selects the block with the lowest anticipated hit count as the victim for replacement.

The analysis evaluated the performance of the hit-count based replacement policy against various existing policies, including LRU, PLRU, BRRIP, and EVA. The performance was evaluated using two metrics: MPKI (misses per thousand instructions) and IPC (instructions per cycle). Additionally, the study assessed the performance of the policies on both single-core and multi-core processors.

It was discovered that the hit-count based replacement policy surpassed all other policies in terms of MPKI reduction, boasting an average reduction of 11.2%. The second most effective policy was EVA, with an average reduction of 6.5%. Furthermore, it was observed that the hit-count based policy exhibited

the greatest improvement in terms of IPC on both single-core and multi-core processors, with an average improvement of 3.5% and 12.2% respectively. The second most effective policy was EVA, with an average improvement of 1.3% and 9.1% on single-core and multi-core processors respectively [3].

The aim of this study was to investigate the effect of increasing the delay in updating cache replacement policies on performance as well as energy. To accomplish this objective, the researchers employed a cycle-accurate simulator that replicated a 16-core processor equipped with a shared last-level cache (LLC). A number of cache replacement policies, such as LRU, PLRU, and a random replacement policy, were evaluated, and the effect of rising the complexity of the replacement policy by incorporating executable replacements and policies that would access larger memory structures was explored [4].

To assess performance, the researchers employed several metrics, including IPC (instructions per cycle), MPKI (misses per thousand instructions), and energy consumption. Surprisingly, it was observed that increasing the latency of cache replacement policy updates had minimal impact on performance, even when significant delays of tens-to-hundreds of thousands of cycles were introduced. This suggests that designers have more flexibility to enhance policy complexity and latency than previously believed [5].

Based on their findings, the authors propose that this newfound flexibility opens up opportunities for implementing more intricate replacement policies, such as programmable replacements and policies that access larger memory structures, without adversely affecting performance. Additionally, it was discovered that the location of the delay can influence performance, particularly for significant delays. Delays in updating the final replacement data in complex policies have a greater impact on performance and MPKI compared to early delays that leave simpler parts of the metadata update process unaltered [6].

In conclusion, this study offers valuable insights into the design of cache replacement policies in high-performance computing. The results indicate that designers can increase policy complexity and latency without incurring significant performance repercussions, thus paving the way for more efficient and effective cache replacement policies in the future.

The paper presents a novel approach to address the challenge of updating cache storage units in real-time for distributed online content-popularity learning. The proposed algorithm, based on Thompson sampling, aims to maximize the edge cache-hit-ratio while ensuring a satisfactory quality-of-experience (QoE) for end users in the long term [7].

The authors formulate the problem as a constrained optimization task, wherein they seek to maximize the edge cache-hit-ratio while guaranteeing the QoE of end users over an extended period. However, the absence of content popularity information poses a significant challenge in solving this optimization problem. To overcome this challenge, the authors propose an Online Distributed Cache Replacement (ODCR) algorithm that leverages the Thompson sampling technique to solve the optimization problem in a distributed manner [8].

Simulation results demonstrate that the proposed algorithm outperforms the benchmark methods, namely the random method, the CUCB algorithm, and the ϵ-greedy method, in terms of cache-hit-ratio and QoE in the long run. Overall, the proposed algorithm offers a superior cache-hit-ratio and enhanced QoE compared to the benchmark methods. The authors further delve into the system model and discuss the challenges associated with finding an efficient and practically feasible solution to the problem. Additionally, the content provides a comprehensive account of the methodology, the algorithms employed, the performance analysis, and the results obtained from the proposed algorithm [9].

The paper presents a caching strategy based on Graph Neural Network (GNN) in order to optimize the caching performance in the context of Named Data Networking (NDN) networks. The main objective is to enhance the cache hit ratio (CHR) and diminish the latency in data delivery. In order to assess the effectiveness of the proposed GNN-based caching algorithm, it is compared with three other deep learning-based caching algorithms as well as three traditional caching algorithms.

The experiments are carried out on the Mini-NDN platform, where different network parameters including network topologies, network sizes, content popularity distributions, node cache sizes, and content sizes are taken into consideration. The performance of all caching strategies is evaluated by employing various metrics such as CHR, byte hit ratio (BHR), and average latency time (ALT). The outcomes of the experiments indicate that the proposed GNN-based caching algorithm surpasses all other caching strategies in terms of CHR, BHR, and ALT, irrespective of the network parameters. Additionally, the paper explores the performance of the GNN-based strategy for different types of information aggregators within the proposed layer.

The dataset is obtained by running each experiment for a duration of 100 minutes, during which no caching algorithm is applied, and by collecting the number of content requests from each node. The empirical results suggest that the GNN-based caching approach can achieve a cache hit ratio that is approximately 50% higher and a latency that is around 30% lower in the best case compared to other deep learning-based caching strategies. In conclusion, the paper argues that the proposed GNN-based caching algorithm can significantly enhance the caching performance in NDN networks [10].

The article examines the relationship between cache size and the duration required to complete a process. The research methodology involved running a combination of four benchmarks simultaneously, with each benchmark assigned to a separate core using the numactl command. The benchmarks were executed multiple times until all four were completed at least once, and the execution time of each benchmark during the initial run was recorded. This evaluation was repeated five times, and the average result was documented.

The benchmarks were categorized into three groups based on their responsiveness to cache: cache-friendly, cache-fitting, and cache-polluting. The sensitivity of the cache was measured by varying the allocated cache size through the page coloring technique and measuring the execution time of each benchmark

when run individually. The cache-polluting category included benchmarks that exhibited a high cache miss rate but were not affected by changes in cache size. The cache-fitting category consisted of benchmarks that showed improved performance as the allocated cache size increased, but reached a point where further gains were not possible. The cache-friendly category included benchmarks that were sensitive to cache performance [11].

The study demonstrated the effectiveness of the approach by running combinations of four benchmarks from a pool of six benchmarks. The results of this grouping, along with selected benchmarks, were used to create offline LLC miss rate curves (MRCs) that aided in identifying the optimal static cache partitioning configuration. In summary, this study provides valuable insights into the impact of cache size on process performance and presents a methodology for evaluating cache sensitivity and optimizing cache partitioning configurations.

3 EXISTING SYSTEM

The existing system comprises of the following cache replacement algorithms or a combination of these with cache replacement policies with an optimizing algorithm to optimally manage cache in a system.

1. **Random Replacement:** Random replacement is a straightforward cache management policy where cache lines are evicted at random when new data needs to be loaded. While simple to implement, it lacks sophistication in capturing access patterns, making it unpredictable and less effective in scenarios where temporal and spatial locality play a crucial role.
2. **Least Recently Used (LRU):** LRU is an intuitive policy that evicts the cache line least recently accessed. It leverages the principle that recently used data is more likely to be used again in the near future. While effective in capturing temporal locality, its drawback lies in the potential computational overhead of maintaining a complete access history, especially in large-scale systems.
3. **FIFO (First-In-First-Out):** FIFO, or First-In-First-Out, follows a simple rule: the first cache line brought in is the first to be evicted. While easy to implement, FIFO may not perform optimally when there's no correlation between the order of data insertion and its subsequent access, leading to potential inefficiencies.
4. **Least Frequently Used (LFU):** LFU tracks the frequency of each cache line's access and evicts the least frequently accessed data. It's effective in scenarios where certain data has consistent and predictable access patterns. However, it may struggle to adapt to dynamic workloads or sudden shifts in access frequencies.
5. **Most Recently Used (MRU):** MRU operates by evicting the cache line most recently accessed. While capturing recent access patterns, it may struggle to adapt to situations where the temporal locality of data access is not the sole determinant of future use, potentially leading to suboptimal performance.

6. **Adaptive Replacement Cache (ARC):** ARC dynamically adjusts between LRU and LFU based on recent access patterns, aiming to provide a balanced approach. It's designed to adapt to changing workload characteristics, making it more versatile than fixed policies. However, its implementation complexity may be a drawback in some contexts.
7. **Clock (or Second Chance):** The Clock replacement policy combines elements of FIFO with a "second chance" mechanism. It maintains a circular list of cache lines and a hand pointing to the next candidate for replacement. If a line is accessed, it gets a "second chance." While simpler than full LRU, it strikes a balance between recency consideration and implementation complexity, making it suitable for certain scenarios.
8. **Belady:** Belady's Optimal Page Replacement Algorithm, often referred to as simply "Belady's Algorithm," is a theoretical and benchmark caching strategy used for evaluating the effectiveness of other page replacement algorithms. Unlike practical algorithms, Belady's Algorithm possesses omniscient knowledge about future memory references. It selects the page for replacement that will be referenced farthest in the future, aiming to minimize the number of page faults. While impractical for real-world implementation due to its reliance on future information, Belady's Algorithm serves as a reference point to measure the optimality of other algorithms. Its use in simulations helps assess how well practical algorithms approximate the optimal page replacement strategy, providing insights into the efficiency of algorithms under different memory access patterns.

3.1 Model Used

The algorithms used in the research work are given below:

1. **Knn(K-Nearest_Neighbors):** K-Nearest Neighbors (KNN) is a cache replacement algorithm that operates on the principle of associating each cache entry with its "neighbors" in the data access pattern. In the context of cache replacement, these neighbors are the other data items that are accessed closely together in time. KNN maintains a record of the historical access sequence and, when a cache miss occurs, evaluates the proximity or similarity of the requested data to its neighbors. The algorithm then selects the cache entry whose neighbors' access patterns are most similar to the current request, effectively replacing the entry that is least similar or least relevant. By considering the local context of data accesses, KNN aims to improve cache hit rates by adapting to the specific access patterns exhibited by the running application. This approach can be particularly effective in scenarios where certain data items are frequently accessed together, enhancing cache utilization and overall system performance.
2. **Logistic_regression:** In cache management, the logistic regression-based method involves assigning probabilities to cache lines or entries based on the likelihood of being accessed in the future. The algorithm leverages historical access patterns and features associated with each cache entry to model

the probability of a cache hit. These features may include factors such as recency of access, frequency of access, or other relevant metrics. Logistic Regression transforms these features into probabilities using a logistic function, allowing the algorithm to make informed decisions about which cache lines to retain or replace. By treating cache replacement as a probabilistic classification problem, Logistic Regression introduces a nuanced and adaptable mechanism for optimizing cache utilization, considering the dynamic nature of data access patterns in modern computing environments.

3. **Multi-layer perceptron classification:** The algorithm employs a multi-layered architecture, where each layer processes and extracts feature from cache access patterns. These features are then fed into the MLP, which serves as a classifier. The MLP, through its learning process, discerns intricate patterns in data access and assigns probabilities or scores to cache entries. This enables the algorithm to make dynamic decisions on cache replacement, considering factors such as recency, frequency, and other relevant features of access. The use of MLP in cache replacement underscores its ability to capture intricate relationships in access patterns, providing a more nuanced and adaptable approach to optimizing cache utilization in diverse computing scenarios.

4 PROPOSED WORK

In response to the growing need for efficient data access and system responsiveness in the dynamic landscape of computing, this research proposes a comprehensive exploration of cache optimization. We aim to seamlessly integrate Machine Learning models within two distinctive methodologies: the Block cache method and the Vector cache method. The overarching goal is not only to elevate cache hit rates but also to unravel the intricate relationships between Machine Learning algorithms, caching strategies, and the dynamic responsiveness of systems.

The first objective involves the meticulous implementation of the Block cache method, where algorithms for block-level optimization will be developed. Utilizing predictive Machine Learning models, including Logistic Regression, KNN, and Neural Network, this approach aims to make judicious decisions about specific blocks worthy of caching. Subsequently, a series of experiments will be designed and executed to evaluate the Block cache method's effectiveness in enhancing cache hit rates based on the decisions made by the Machine Learning models.

In parallel, the exploration of the Vector cache method will emphasize a holistic approach to cache optimization, considering relationships among different blocks at a higher level. Here, Machine Learning models will be employed to make intelligent caching decisions, and the impact of these decisions on cache performance will be systematically assessed.

The second set of objectives involves a comparative analysis of widely-used paging/cache replacement algorithms—LRU and LFU—within both the Block and Vector cache methods. This will provide insights into the strengths and

weaknesses of each algorithm under diverse workloads, considering their impact on cache hit rates, miss rates, and eviction rates.

Simultaneously, the third set of objectives centers around the comparative evaluation of three distinct Machine Learning models—Logistic Regression, KNN, and Neural Network. Their performances within the cache optimization framework will be systematically compared through graphical representations, offering a nuanced understanding of their effectiveness.

Furthermore, the integration of Belady's optimization algorithm will contribute to the determination of optimal cache configurations based on historical data access patterns. This will set the foundation for a more informed evaluation of the considered cache management strategies. Lastly, the introduction of a dynamic Machine Learning Policy will continuously assess the system's state in real-time. This policy aims to dynamically compute cache hit rates, eviction strategies, and miss rates, ensuring adaptive responsiveness to evolving workloads. Figure 1 represents the working model of the Belady's optimization algorithm integration.

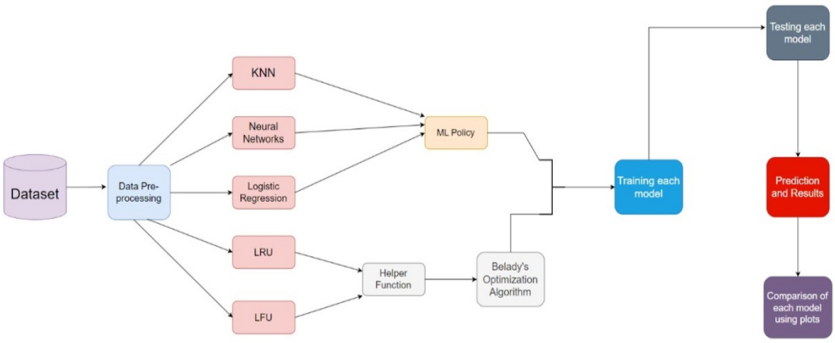

Fig. 1. Representation of working model of the Belady's optimization algorithm integration

Fig. 2 illustrates an overview of a cache partitioning approach based on page reusability. This method aims to enhance cache efficiency by dynamically allocating cache space to pages with higher reusability, thereby reducing cache thrashing and improving overall system performance.

Figure 3 depicts the physical mapping between physical memory and the last level cache using a straightforward hash function. This mapping scheme facilitates efficient cache access by assigning physical memory pages to cache sets based on their hashed addresses, enabling rapid retrieval and storage of frequently accessed data in the cache.

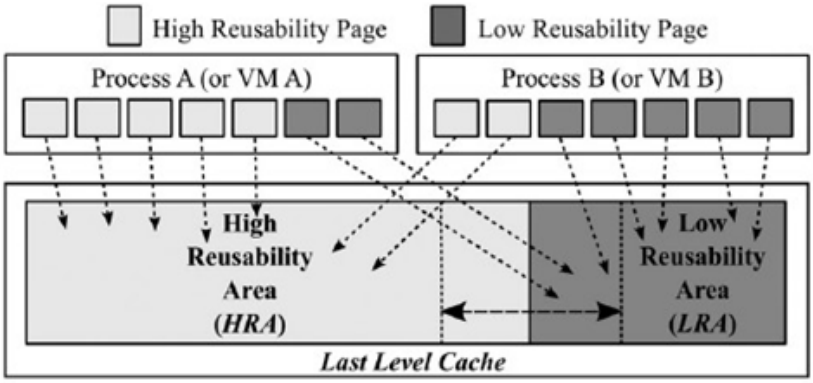

Fig. 2. The overview of page reusability-based cache partitioning

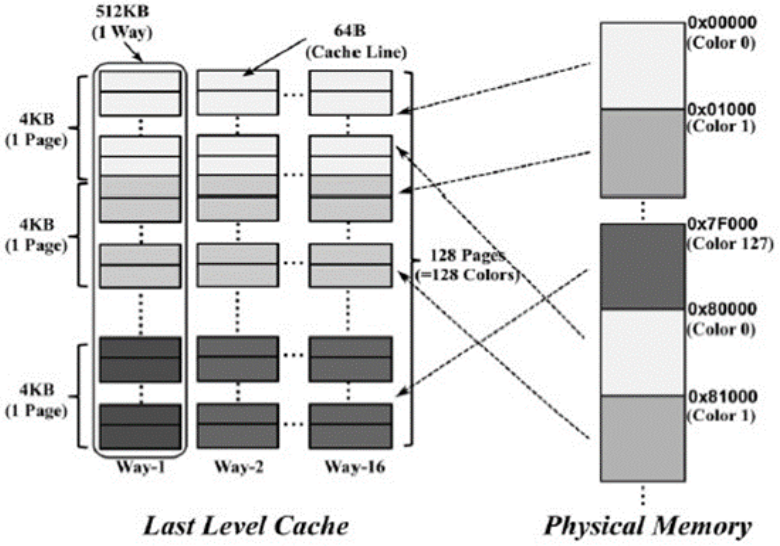

Fig. 3. Physical mapping between physical memory and the last level cache

5 Screenshot Output

5.1 Hit rate using machine learning algorithms

Figure 4 - 6 demonstrate the insight of hit rate using various machine learning algorithm such as logistic regression (Fig. 4), K-Nearest Neighbour (Fig. 5), Multi-layer perceptron classifier (Fig. 6).

5.2 Logistic_regression

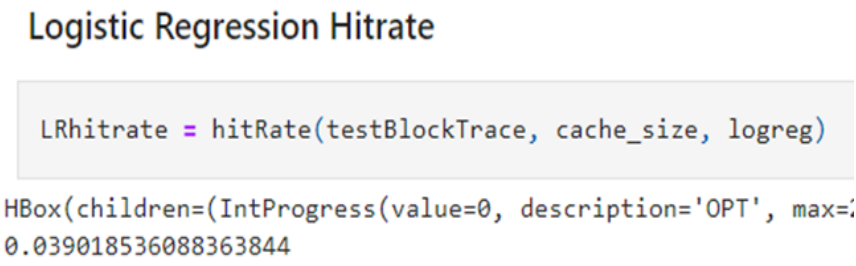

Fig. 4. Logistic Regression Hit Rate

5.3 K-Nearest Neighbour

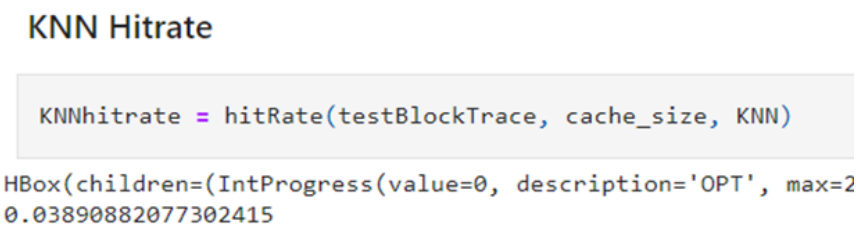

Fig. 5. K-Nearest Neighbour Hit Rate

5.4 Multi-layer perceptron classifier

Graphical representation

NeuralNet Hitrate

```
NNhitrate = hitRate(testBlockTrace, cache_size, NN)
```

```
HBox(children=(IntProgress(value=0, description='OPT', max=2068991, style=ProgressS
tyle(description_width='ini…
0.038370877398693375
```

Fig. 6. Multi-layer Perceptron –Neural Net Hit Rate

The hit rate, miss rate, and eviction rate of cache mechanisms are three important variables that are taken into account while performing performance analysis. The graphical representations for these machine learning models are captured in Figures (7-11) pertaining to analysis for hitrate with different train/test split, accuracy with different train/test split, accuracy with different train/test split, hit rate for various machine learning models, hit rate vs eviction count.

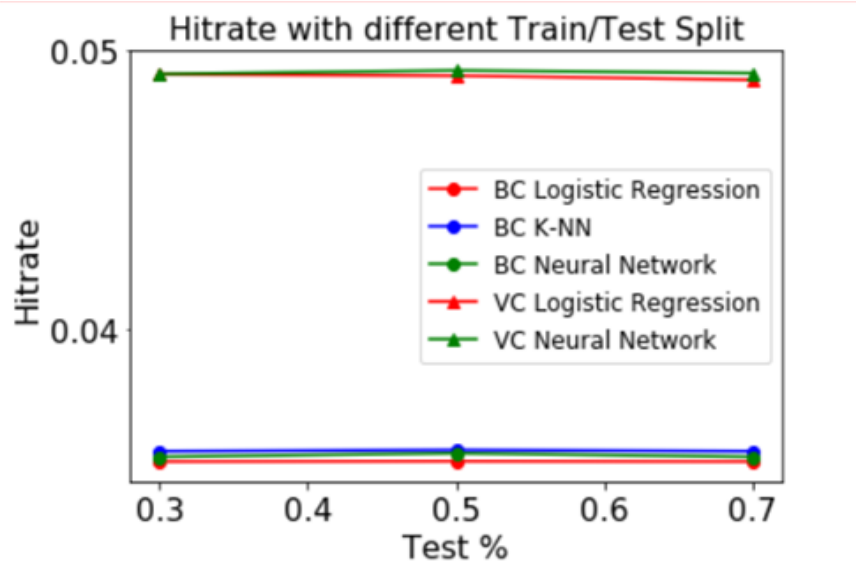

Fig. 7. Hit Rate with different Train/Test Split

Thus these measures provide light on how successful and efficient certain caching strategies and models are demonstrated over various machine learning algorithms.

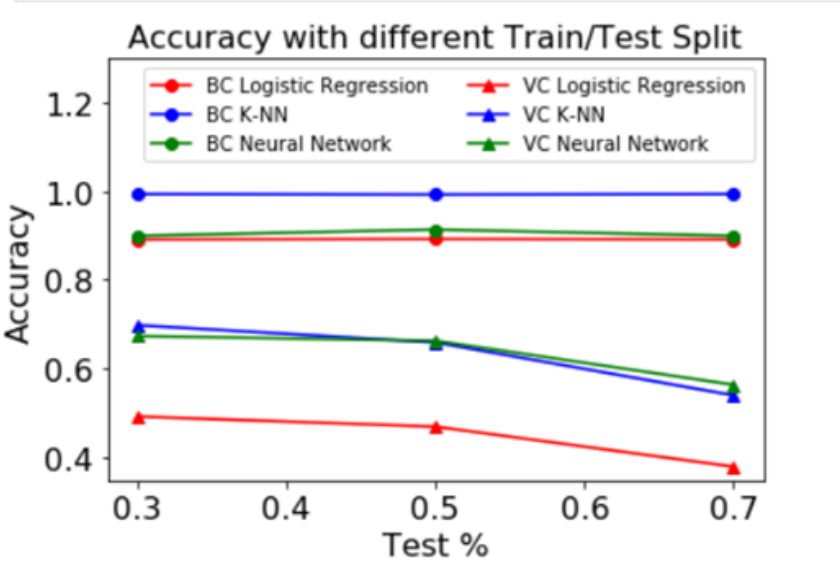

Fig. 8. Accuracy with different Train/Test Split

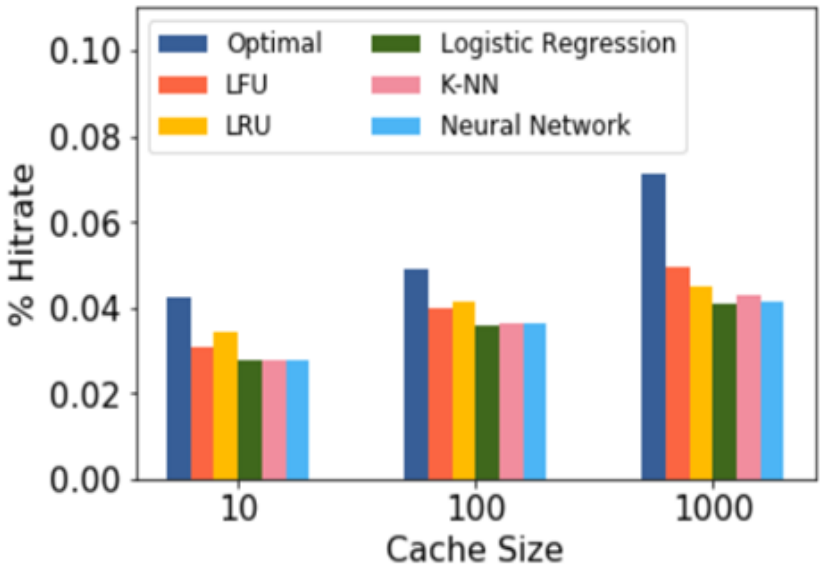

Fig. 9. Hit Rate for Six Machine Learning Models

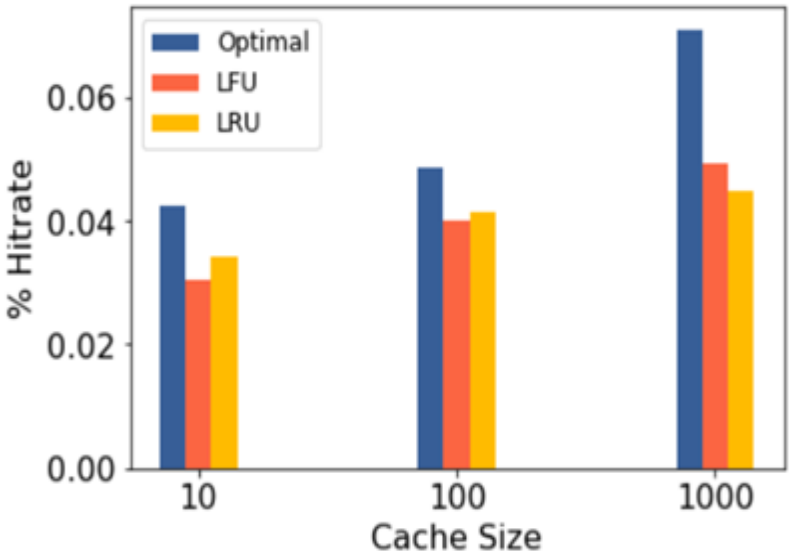

Fig. 10. Hit Rate for Three Machine Learning Models

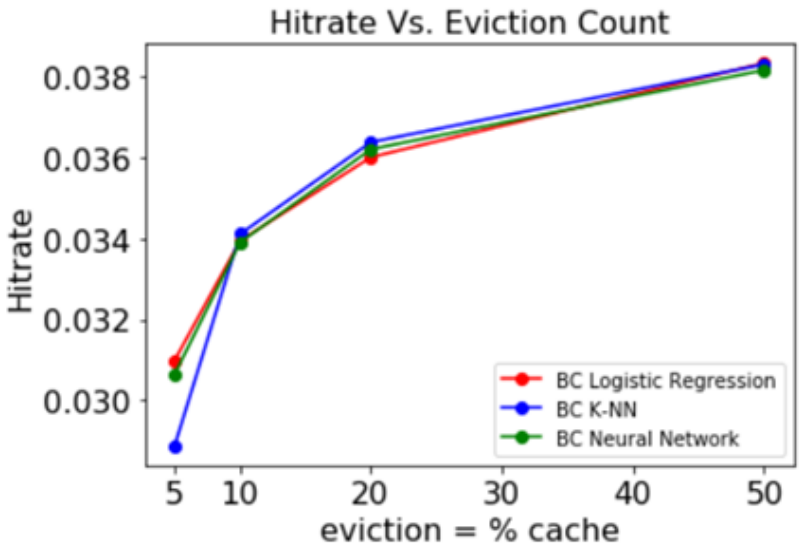

Fig. 11. Hit Rate Vs Eviction Count

6 CONCLUSION

In conclusion, this research has undertaken a thorough exploration of cache performance enhancement by implementing and evaluating two novel cache models, namely the Block Cache Model and the Vector Cache Model. Through the integration of traditional cache replacement policies such as LRU and LFU, both models were established as baseline systems. The infusion of machine learning (ML) models—Logistic Regression, KNN, and Neural Network—into cache replacement strategies showcased promising results, with the ML-enhanced systems demonstrating the potential to adapt dynamically to changing access patterns.

The integration of Belady's Optimal algorithm with ML models further solidified the research's objective of creating an intelligent and adaptive cache system. The experiments conducted to compare the cache hit rate, miss rate, and eviction rate across different models and approaches yielded valuable insights. The findings suggest that the incorporation of machine learning can indeed contribute to improving cache performance, particularly in scenarios where access patterns are challenging to predict accurately using traditional methods alone. The successful integration of machine learning with traditional cache management techniques opens avenues for further exploration and refinement, offering potential breakthroughs in the ongoing pursuit of creating efficient and adaptive caching systems in diverse computing environments.

7 FUTURE WORK

Future work in this domain presents exciting opportunities to extend the impact of cache optimization beyond the confines of the current research. One avenue for exploration involves the development of an application designed to analyze and optimize running applications on a computer. This application could leverage the insights gained from the implemented cache models and machine learning algorithms to dynamically adjust cache management strategies based on real-time application behavior. By providing a user-friendly interface and seamless integration into existing systems, such an application could offer a practical solution for enhancing overall system performance.

Additionally, the research success in integrating three specific machine learning models—Logistic Regression, KNN, and Neural Network—paves the way for evaluating and incorporating a broader spectrum of machine learning algorithms. Future research could delve into exploring and assessing additional ML models to identify which ones exhibit optimal performance in managing cache. This could involve experimenting with reinforcement learning techniques, ensemble methods, or other advanced machine learning approaches to further enhance the adaptability and efficiency of the cache management system. A comprehensive evaluation of various ML models could provide valuable insights into the strengths and limitations of each approach, guiding the development of more sophisticated and effective cache optimization strategies. In essence, the future

work outlined aims to translate the research theoretical insights and experimental successes into practical applications, advancing the field of cache optimization towards more adaptive and intelligent solutions for managing the complexities of modern computing environments.

References

1. Vakil-Ghahani, Armin, et al. "Cache replacement policy based on expected hit count." IEEE computer architecture letters 17.1 (2017): 64-67.
2. Nematallah, Ahmed, Chang Hyun Park, and David Black-Schaffer. "Exploring the Latency Sensitivity of Cache Replacement Policies." IEEE Computer Architecture Letters (2023).
3. Sun, Zhenfeng, and Mohammad Reza Nakhai. "Distributed learning-based cache replacement in collaborative edge networks." IEEE Communications Letters 25.8 (2021): 2669-2672.
4. Hou, Jiacheng, et al. "A graph neural network approach for caching performance optimization in ndn networks." IEEE Access 10 (2022): 112657-112668.
5. Park, Jiwoong, Heonyoung Yeom, and Yongseok Son. "Page reusability-based cache partitioning for multi-core systems." IEEE Transactions on Computers 69.6 (2020): 812-818.
6. Mohan Aparna, R. Maheswari, J. V. Thomas Abraham (2021), Systematic Approach to Deal with Internal Fragmentation and Enhancing Memory Space during COVID-19. Applied Learning Algorithms for Intelligent IoT, ISBN: 9781003119838.
7. M. Hassan, C. H. Park and D. Black-Schaffer, "Architecturally-independent and time-based characterization of SPEC CPU 2017", Proc. IEEE Int. Symp. Perform. Anal. Syst. Softw., pp. 107-109, 2020.
8. E. Z. Liu, M. Hashemi, K. Swersky, P. Ranganathan and J. Ahn, "An imitation learning approach for cache replacement", Proc. 37th Int. Conf. Mach. Learn., pp. 6237-6247, 2020.
9. I. Shah, A. Jain and C. Lin, "Effective mimicry of belady's MIN policy", Proc. IEEE Int. Symp. High-Perform. Comput. Architecture, pp. 558-572, 2022.
10. Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. Deep Q-learning from demonstrations. In Proceedings of the AAAI Conference on Artificial Intelligence, 2018.
11. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J.,Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems, pp. 8024–8035, 2019