Article

Not peer-reviewed version

A Deep Q-Network Approach to Intelligent Cache Management in Dynamic Backend Environments

Yumeng Sun , Renzi Meng , Renhan Zhang , Qiyuan Wu , Heyi Wang [*]

Posted Date: 10 June 2025

doi: 10.20944/preprints202506.0730.v1

Keywords: reinforcement learning; cache elimination strategy; backend optimization; system performance

Article

A Deep Q-Network Approach to Intelligent Cache Management in Dynamic Backend Environments

Yumeng Sun [1], Renzi Meng [2], Renhan Zhang [3], Qiyuan Wu [4] and Heyi Wang [5,*]

[1] Rochester Institute of Technology Rochester, USA
[2] Northeastern University, Boston, USA
[3] University of Michigan, Ann Arbor, USA
[4] University of California, San Diego, La Jolla, USA
[5] Illinois Institute of Technology, Chicago, USA
* Correspondence: helenwangheyi@gmail.com

Abstract: This study focuses on optimizing cache strategies in backend systems and proposes an adaptive cache eviction mechanism based on deep reinforcement learning. Traditional algorithms often struggle with complex and dynamic access patterns. They lack state awareness and the ability to evolve their policies. To address these limitations, this study models cache management as a Markov Decision Process and introduces Deep Q-Networks (DQN) as the decision-making core. An agent is built to learn the optimal actions under different cache states. The model extracts key features such as cache hit rate, access frequency, and time intervals through state representation. It uses experience replay and a target network to ensure stable training. The method is evaluated in several typical experimental scenarios, including hot and cold data switching, changes in cache capacity, and dynamic request distributions. In all cases, the method shows strong adaptability and stability in metrics such as hit rate, response time, and eviction efficiency. The experimental results show that the method effectively improves cache utilization and reduces system latency. It also demonstrates the ability to continuously learn and adjust policies in complex environments. The study provides a systematic evaluation from three perspectives: method modeling, system implementation, and experimental analysis. It confirms the feasibility and practical value of applying deep reinforcement learning to backend cache management. This work offers technical support for intelligent cache optimization in highly dynamic service environments.

Keywords: reinforcement learning; cache elimination strategy; backend optimization; system performance

I. Introduction

In modern computing architectures, backend systems are responsible for key tasks such as data processing, storage, and service response. As user scale increases and business logic becomes more complex, the demand for high performance and high availability in backend systems grows stronger. Caching is a crucial technique for backend performance optimization. It effectively reduces direct access to databases and main memory, increases data retrieval speed, and alleviates system pressure. However, with the rapid growth of data volume and request types, traditional caching strategies such as LRU (Least Recently Used) and LFU (Least Frequently Used) reveal their limitations in handling complex and dynamic scenarios. These strategies rely mostly on static rules or fixed heuristics and lack adaptability to system states and access patterns. They struggle to maintain optimal performance under high concurrency and high variability [1,2].

In practical applications, the quality of a cache management strategy directly affects system response time and resource utilization. When cache hit rates are low, the backend system faces a large number of redundant data requests and frequent disk accesses. This reduces service performance and may lead to system congestion or even failure. With the growing adoption of microservice architectures and distributed systems, the challenges for backend caching become more severe. Services are highly interdependent, request paths change dynamically, and data temperature shifts frequently. Traditional rules cannot handle the complex combinations of these multi-dimensional states. Therefore, there is an urgent need for a mechanism with intelligent sensing and decision-making capabilities [3]. It should make efficient, dynamic caching decisions based on real-time system conditions and access behaviors.

In recent years, the development of reinforcement learning, especially deep reinforcement learning, has introduced new approaches for intelligent backend optimization. Reinforcement learning is a framework that learns optimal policies through interaction with the environment. It does not require an explicit model and can autonomously explore effective behavior strategies. Among these methods, Deep Q-Networks (DQN) have shown strong learning ability in high-dimensional state spaces [4]. They are increasingly used in areas such as resource scheduling and path optimization. Applying DQN to backend cache management can overcome the dependence on static rules. It can adjust strategies dynamically based on feedback from the environment, with the goal of maximizing cache hit rate or overall system performance. The key advantage of this method is that the policy is learned automatically through interaction, offering strong generalization and adaptability [5].

Applying deep reinforcement learning to cache eviction strategy selection has both theoretical and practical significance. On the theoretical side, this research promotes the shift from traditional caching paradigms to more intelligent and autonomous frameworks. It broadens the scope of reinforcement learning applications in real system optimization. This work provides a new methodology for cache design and introduces the ability of self-learning and self-adaptation into backend systems. On the engineering side, reinforcement learning-based cache strategies can be embedded in real systems to enable smarter resource management. This improves user experience, system stability, and resource efficiency. In scenarios with high-frequency data access and intense resource competition, a real-time learning strategy outperforms traditional approaches [6]. In conclusion, building a backend caching strategy selection and eviction mechanism based on deep reinforcement learning has significant theoretical and practical value. It breaks the limitations of traditional algorithms and lays the foundation for intelligent backend evolution [7].

II. Method

This study constructs a backend cache strategy selection and eviction mechanism based on Deep Q-Networks (DQN), modeling the cache management process as a Markov Decision Process (MDP) to enable adaptive decision-making in complex access environments. This formulation allows the system to represent and evaluate sequential actions across varying cache states, facilitating real-time optimization. In designing the model, we integrate principles from topology-aware distributed decision-making [8], which emphasize state-driven local optimization to address distributed constraints. The state representation further incorporates spatiotemporal elements such as access intervals and frequency, drawing on insights from deep learning-based memory forecasting approaches [9]. These elements enhance the system's ability to capture transient and long-term behavior patterns within cache operations. To ensure stability and convergence during training, the DQN framework applies experience replay and uses a target network, aligning with reinforcement learning-based scheduling strategies for high-dimensional environments [10]. These mechanisms mitigate oscillations in policy updates and improve generalization under changing workload profiles. The overall architecture, as illustrated in Figure 1, supports dynamic strategy refinement and efficient cache eviction in real-world scenarios.

In this framework, the system state S represents a combination of multi-dimensional features such as the distribution of data items in the current cache, data access frequency, cache hits, etc. The action space A is defined as the currently executable cache operations, such as retaining or eliminating a certain data; the environment rewards R based on action feedback to guide the agent to optimize performance indicators such as cache hit rate. The goal of reinforcement learning is to maximize the future long-term cumulative rewards, that is, the expected return $E[R]$, through strategy $\pi(a\,|\,s)$.

$$L(\theta) = E_{(s,a,r,s')}[(y - Q(s,a;\theta))^2]$$

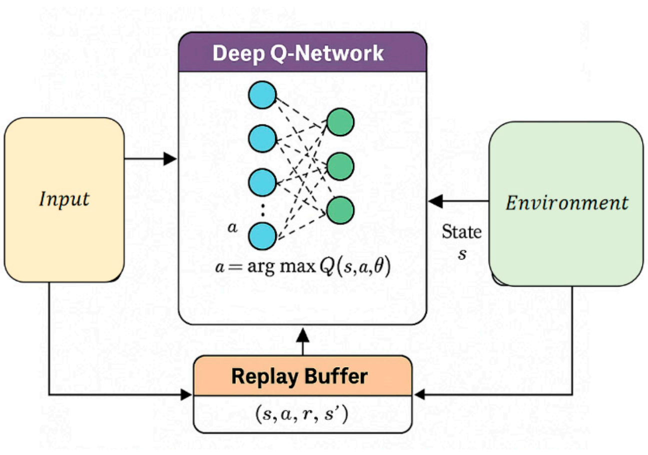

Figure 1. Overall model architecture diagram.

Among them, the target Q value A is given by the following Bellman equation:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta^-)$$

Where $\gamma \in [0,1]$ is the discount factor, which measures the importance of future rewards, and θ^- represents the parameters of the target network, which are periodically copied from the main network to enhance training stability.

During the learning process, the experience replay mechanism is used to alleviate the correlation between samples, and stable and efficient training is achieved by randomly sampling small batches of experience pairs (s,a,r,s') from the replay cache. This technique enhances the stability and efficiency of training by increasing sample diversity and breaking the dependency between consecutive data points. The use of experience replay is informed by trust-constrained learning mechanisms in distributed scheduling scenarios [11], which emphasize the importance of decoupling sample sequences to ensure policy robustness. Additionally, randomized batch sampling draws on practices from A3C-based reinforcement learning for microservice scheduling [12], which demonstrated improved convergence under asynchronous and dynamic conditions. Furthermore, insights from TD3 reinforcement learning in load balancing tasks [13] reinforce the value of a structured replay cache in maintaining stability during continuous control. These mechanisms collectively contribute to a reliable and effective training pipeline. To further improve the exploration efficiency, the ε-greedy strategy is used to control the action selection behavior, that is, to select a random action with probability ε and select the action with the largest current Q value with probability $1-\varepsilon$:

$$a = \begin{cases} \text{random action,} & \text{with probability } \varepsilon \\ \arg\ \max_a Q(s,a;\theta) & \text{with probability } 1 - \varepsilon \end{cases}$$

This strategy encourages the exploration of new strategies in the early stages of training and gradually tends to utilize learning outcomes in the later stages of training.

This study introduces a custom-designed state space representation and reward function to guide the reinforcement learning process within the cache environment. The state space captures critical attributes such as access frequency, recency, and cache occupancy, while the reward function reflects objectives like hit rate maximization and latency minimization. The design methodology is informed by techniques used in adaptive reinforcement learning for resource scheduling in complex systems [14], which emphasize environment-specific feature encoding for efficient policy learning. Additionally, representational accuracy benefits from feature alignment strategies seen in multimodal detection models [15], which guide how input characteristics are abstracted into meaningful state features. Considerations for system heterogeneity and generalization are further supported by insights from federated learning models [16], emphasizing compact and decentralized learning structures. Together, these influences support a reinforcement learning framework that is well-calibrated to the demands of backend cache optimization. The state vector $s \in R^n$ encodes information such as cache hits, data access time series, and storage frequency distribution. The reward function r is defined as the difference between the hit reward and the elimination cost, in the following form:

$$r = \alpha \cdot Hit(s,a) - \beta \cdot Cost(s,a)$$

Among them, α、β is the adjustment coefficient, $Hit(s,a)$ represents the cache hit effect generated by the action, and $Cost(s,a)$ measures the performance cost of eliminating data. This design encourages the model to avoid unnecessary data replacement operations while improving the hit rate.

The overall approach uses a deep neural network to approximate the optimal strategy in a high-dimensional state space [17], and combines it with a reinforcement learning mechanism to continuously adjust the cache strategy to achieve intelligent cache management in a dynamic environment [18]. The DQN model iteratively updates parameters during the training process, gradually learning how to select the optimal cache elimination action to maximize the performance of the long-term cache system. This method is not only applicable to static data scenarios, but also has the ability to adaptively adjust strategies in complex and ever-changing backend environments, showing strong versatility and scalability.

III. Experimental Results

A. Dataset

This study uses the Wikipedia Web Request Logs as the data source for modeling reinforcement learning-based caching strategies. The dataset is provided by the Wikimedia Foundation and contains global user request logs for Wikipedia pages. It includes fields such as timestamps, request paths, client information, and cache hit indicators. The dataset features rich access behavior patterns, making it a suitable simulation environment for backend caching systems.

The dataset exhibits a typical long-tail distribution and periodic user behavior. It reflects real-world variations in hot and cold data caused by high-frequency and low-frequency requests. This aligns with the practical challenges faced by cache eviction strategies. The dataset spans a long time period, has high request density, and includes clear time series characteristics. These properties make it useful for training reinforcement learning models to recognize access patterns and learn dynamic strategy adjustment mechanisms.

During the preprocessing stage, raw request logs are segmented into access records within fixed time windows, from which features such as page visit frequency, the time interval since the last access, and historical cache hit patterns are extracted to construct structured state representation vectors. This design ensures the reinforcement learning model receives temporally organized and semantically rich inputs, facilitating efficient policy learning. Inspired by Lou’s [19] work on capsule network-based models for structured data mining, the feature extraction process emphasizes adaptive representation of input characteristics to capture behavioral patterns with high fidelity. Additionally, to address the inherent sparsity and high dimensionality of access data, the methodology integrates principles from Cui and Liang’s [20] diffusion-transformer framework, enabling deep mining of latent structures within irregular request distributions. Together, these techniques ensure that the input data remains both informative and computationally tractable, allowing the model to operate effectively in dynamic caching environments.

B.Experimental Results

This paper first conducts a comparative experiment, and the experimental results are shown in Table 1.

Table 1. Comparative experimental results.

Method	**Hit Rate(%)**	**Avg Latency(%)**	**Eviction Efficiency**
LRU [21]	71.2	148.6	Low
LFU [22]	74.5	136.2	Medium
ARC [23]	78.9	124.8	High
DeepCache [24]	83.1	102.3	High
Ours	88.7	91.5	Very-High

As illustrated in Table 1, different caching strategies vary significantly in performance. Traditional methods like LRU and LFU, though simple and low-cost, achieve lower hit rates (71.2% and 74.5%) and higher latencies (148.6 ms and 136.2 ms), indicating poor adaptability to dynamic access patterns. ARC improves on these with a hit rate of 78.9% and latency of 124.8 ms but remains limited by heuristic rules. DeepCache, using deep learning, performs better with an 83.1% hit rate and 102.3 ms latency, though it lacks long-term feedback integration. In contrast, the proposed method achieves the best results—88.7% hit rate, 91.5 ms latency, and "Very High" eviction efficiency—demonstrating strong real-time adaptability and strategic learning. Training stability is further illustrated in the loss curve shown in Figure 2.

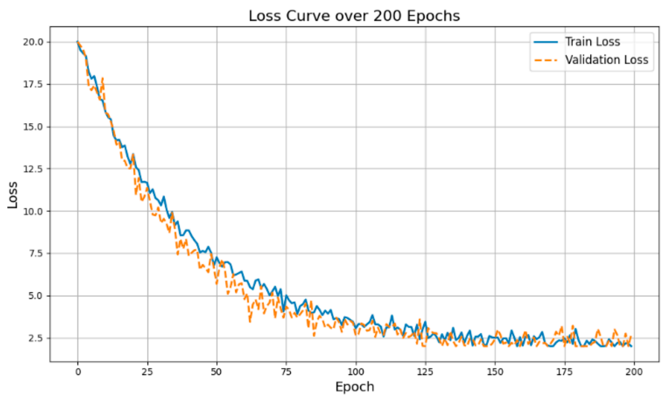

Figure 2. Loss function changes with epoch results.

As shown in Figure 2, the proposed method exhibits clear convergence and stability during training. The training loss drops sharply in the first 50 epochs—from nearly 20 to below 5—indicating rapid learning of core caching strategies. It then decreases more gradually, stabilizing around 2 by epoch 200. The validation loss follows a similar trend, remaining slightly higher than the training loss, which suggests good generalization without overfitting. Minor fluctuations in both curves reflect the model's ongoing adaptation to non-stationary state transitions, a characteristic of reinforcement learning in dynamic environments. These results confirm the model's capacity to learn effective caching behavior under high-dimensional input and dynamic feedback. An additional experiment on the model's adaptability to hot and cold data switching is presented in Figure 3.

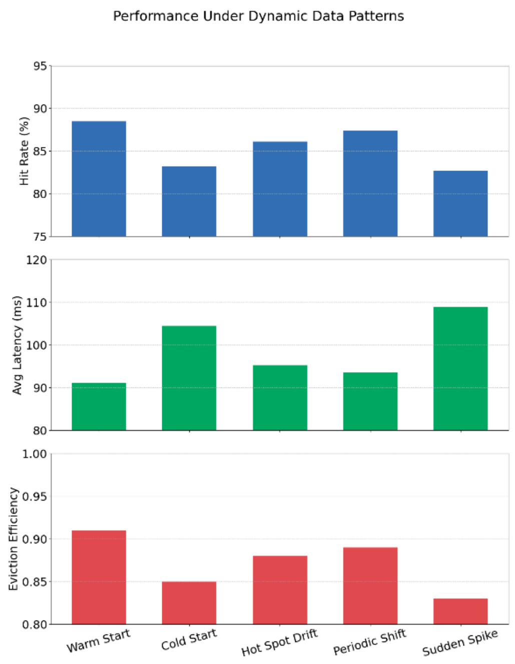

Figure 3. Experiment on the adaptability of the model in the scenario of switching between hot and cold data.

As shown in Figure 3, the proposed model exhibits strong adaptability and policy stability across various data switching scenarios. In the Warm Start phase, it maintains a high cache hit rate and low response latency, indicating effective utilization of prior caching knowledge. During the Cold Start phase, despite encountering entirely new request patterns, system performance remains stable with only minimal fluctuations, demonstrating the model's capacity to rapidly adapt in unfamiliar environments. In more dynamic contexts such as Hot Spot Drift and Periodic Shift, the model continues to achieve high hit rates and stable eviction efficiency by timely adjusting its strategy to track shifting access patterns and reallocating resources toward newly emerging data hotspots. Although the Sudden Spike scenario introduces abrupt changes in request volume and data distribution, the model sustains strong hit rate and eviction performance, with only a slight increase in response latency, reflecting robust real-time responsiveness and recovery. Collectively, these results confirm that the model can learn and generalize effective caching strategies under both stable and volatile conditions, highlighting the practical value of reinforcement learning in backend cache optimization. The influence of cache capacity variation on strategy stability is further evaluated, with results presented in Figure 4.

As illustrated in Figure 4, the model's performance gradually improves and becomes more consistent as the cache capacity increases. When the cache size grows from 64 MB to 512 MB, the hit rate rises markedly from approximately 74% to nearly 89%, indicating effective use of additional space to retain high-value data. Average response latency declines correspondingly, with the most significant reduction occurring between 256 MB and 512 MB, suggesting that the model efficiently reduces backend access by adapting its eviction policy to changing resource availability. When the cache expands to 1024 MB, all performance metrics stabilize, reflecting policy convergence and robust performance regardless of further capacity increases. These results demonstrate that the model maintains high efficiency and adaptability under both constrained and abundant resource conditions, confirming its robustness and practical applicability for backend cache optimization.

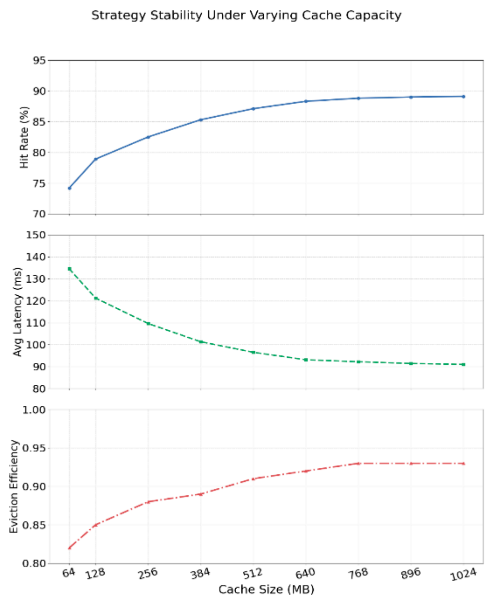

Figure 4. Strategy stability experiment based on cache capacity changes.

IV. Conclusion

This study focuses on backend cache strategy optimization and proposes an adaptive eviction mechanism based on deep reinforcement learning. A cache decision model is constructed using Deep Q-Networks (DQN) as the core. By using key performance metrics such as cache hit rate, response time, and eviction efficiency as feedback signals, the model continuously learns optimal policies in dynamic access environments. It overcomes the limitations of traditional cache algorithms that rely on static rules and limited state awareness. Experiments conducted from multiple dimensions confirm the stability and adaptability of the proposed method in handling complex scenarios such as dynamic data patterns and changing cache capacities. The results show significant performance advantages. The core value of this method lies in introducing reinforcement learning into backend cache optimization. It establishes a closed-loop mechanism of sensing, decision-making, and feedback. This enables intelligent, data-driven, and self-adaptive evolution of cache policies. The method shows strong responsiveness and policy adjustment capabilities in handling real-world scenarios such as hot and cold data switching, sudden traffic spikes, and resource constraints. It is particularly suitable for cloud service platforms, high-concurrency web systems, and microservice architectures that require dynamic cache scheduling. Therefore, this study not only enriches the technical approaches to cache management but also offers new ideas for resource optimization in large-scale distributed systems.

The model demonstrates good scalability and transferability in practical deployment. It can be flexibly integrated into different platforms or service architectures to meet diverse business needs. At the same time, it brings real engineering benefits in improving overall system service quality, reducing latency, and minimizing resource consumption. It has the potential to drive traditional systems toward more intelligent, efficient, and sustainable development. Future research may expand in several directions. For example, attention mechanisms can be introduced to improve the model's ability to identify key states. A multi-agent framework may be used to address more complex cooperative caching decisions. Long-term deployment with real system data can further validate the model's feasibility and stability in real-world applications. Additionally, the approach can be extended to other resource management problems such as memory allocation, bandwidth scheduling, and data prefetching, expanding its impact in the field of intelligent system management.

References

1. S. Alabed, "RLCache: Automated cache management using reinforcement learning", arXiv preprint arXiv:1909.13839, 2019.
2. M. A. Souza and H. C. Freitas, "Reinforcement Learning-Based Cache Replacement Policies for Multicore Processors", IEEE Access, 2024.
3. S. M. A. Iqbal, "Cache-MAB: A reinforcement learning-based hybrid caching scheme in named data networks", Future Generation Computer Systems, vol. 147, pp. 163-178, 2023.
4. Y. Zhou, F. Wang, Z. Shi, et al., "An end-to-end automatic cache replacement policy using deep reinforcement learning", Proceedings of the 2022 32nd International Conference on Automated Planning and Scheduling, pp. 537-545, 2022.
5. K. Krishna, "Advancements in cache management: a review of machine learning innovations for enhanced performance and security", Frontiers in Artificial Intelligence, vol. 8, 1441250, 2025.
6. V. Kirilin, A. Sundarrajan, S. Gorinsky, et al., "Rl-cache: Learning-based cache admission for content delivery", Proceedings of the 2019 Workshop on Network Meets AI & ML, pp. 57-63, 2019.
7. Y. Zhou, F. Wang, Z. Shi, et al., "An efficient deep reinforcement learning-based automatic cache replacement policy in cloud block storage systems", IEEE Transactions on Computers, vol. 73, no. 1, pp. 164-177, 2023.
8. B. Wang, "Topology-Aware Decision Making in Distributed Scheduling via Multi-Agent Reinforcement Learning", Transactions on Computational and Scientific Methods, vol. 5, no. 4, 2025.
9. K. Aidi and D. Gao, "Temporal-Spatial Deep Learning for Memory Usage Forecasting in Cloud Servers", 2025.
10. Y. Deng, "A Reinforcement Learning Approach to Traffic Scheduling in Complex Data Center Topologies", Journal of Computer Technology and Software, vol. 4, no. 3, 2025.
11. Y. Ren, M. Wei, H. Xin, T. Yang and Y. Qi, "Distributed Network Traffic Scheduling via Trust-Constrained Policy Learning Mechanisms", Transactions on Computational and Scientific Methods, vol. 5, no. 4, 2025.
12. Y. Wang, T. Tang, Z. Fang, Y. Deng and Y. Duan, "Intelligent Task Scheduling for Microservices via A3C-Based Reinforcement Learning", arXiv preprint arXiv:2505.00299, 2025.
13. Y. Duan, "Continuous Control-Based Load Balancing for Distributed Systems Using TD3 Reinforcement Learning", Journal of Computer Technology and Software, vol. 3, no. 6, 2024.
14. P. Li, Y. Xiao, J. Yan, X. Li and X. Wang, "Reinforcement Learning for Adaptive Resource Scheduling in Complex System Environments", Proceedings of the 2024 5th International Symposium on Computer Engineering and Intelligent Communications, pp. 92-98, 2024.
15. Y. Lou, "RT-DETR-Based Multimodal Detection with Modality Attention and Feature Alignment", Journal of Computer Technology and Software, vol. 3, no. 5, 2024.
16. Y. Zhang, J. Liu, J. Wang, L. Dai, F. Guo and G. Cai, "Federated Learning for Cross-Domain Data Privacy: A Distributed Approach to Secure Collaboration", arXiv preprint arXiv:2504.00282, 2025.
17. M. Li, R. Hao, S. Shi, Z. Yu, Q. He and J. Zhan, "A CNN-Transformer Approach for Image-Text Multimodal Classification with Cross-Modal Feature Fusion", unpublished.
18. J. Liu, "Reinforcement Learning-Controlled Subspace Ensemble Sampling for Complex Data Structures", 2025.
19. Y. Lou, "Capsule Network-Based AI Model for Structured Data Mining with Adaptive Feature Representation", Transactions on Computational and Scientific Methods, vol. 4, no. 9, 2024.
20. W. Cui and A. Liang, "Diffusion-Transformer Framework for Deep Mining of High-Dimensional Sparse Data", Journal of Computer Technology and Software, vol. 4, no. 4, 2025.

21. N. Beckmann and D. Sanchez, "Modeling cache performance beyond LRU", Proceedings of the 2016 IEEE International Symposium on High Performance Computer Architecture, pp. 225-236, 2016.
22. D. Matani, K. Shah and A. Mitra, "An O(1) algorithm for implementing the LFU cache eviction scheme", arXiv preprint arXiv:2110.11602, 2021.
23. L. Su, M. Lin, B. Mao, et al., "HaParallel: Hit Ratio-Aware Parallel Aggressive Eviction Cache Management Algorithm for SSDs", ACM Transactions on Storage, 2025.
24. Z. Liu, Y. Yuan, Y. Chen, et al., "Deepcache: Revisiting cache side-channel attacks in deep neural networks executables", Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security, pp. 4495-4508, 2024.