Computer Life
ISSN: 1819-4818 | Vol. 14, No. 1, 2026

When AI Meets Cache: Intelligent Memory Hierarchy Management in Modern CPUs

Yihan Wang [1,*] and Suchuan Xing [2]

[1] University of Pennsylvania, United States
[2] Duke University, United States
*** Corresponding author Email:** wangyh1@alumni.upenn.edu

Abstract: The persistent and widening gap between processor execution speed and memory access latency—commonly termed the "memory wall"—has made efficient cache management one of the most consequential challenges in modern computer architecture. Traditional heuristic-based strategies such as least recently used (LRU), re-reference interval prediction (RRIP), and signature-based hit prediction (SHiP) have long served as foundations of cache control, yet they fundamentally fail to capture complex, application-specific, or workload-diverse memory access patterns. The rapid maturation of artificial intelligence (AI) and machine learning (ML) techniques now offers transformative opportunities to reimagine every tier of the memory hierarchy—from L1 instruction caches to last-level cache (LLC) partitioning and speculative hardware prefetching. This review surveys the state of the art at the intersection of AI and cache management, examining techniques including deep reinforcement learning (RL), long short-term memory (LSTM) networks, transformer-based sequence models, and graph neural networks (GNN) as applied to the three principal problem domains of cache replacement, hardware prefetching, and adaptive resource partitioning. We critically analyze trade-offs between prediction accuracy, inference latency, hardware overhead, and practical deployability, synthesizing empirical findings from recent simulation studies and silicon implementations. Key open challenges are identified, including online learning constraints, workload generalization, and the integration of AI inference within microarchitectural timing budgets.

Keywords: Cache replacement policy; Memory hierarchy; Machine learning; Hardware prefetching; Reinforcement learning; Last-level cache; Deep learning; Neural network; Microarchitecture; Prefetcher.

1. Introduction

Modern central processing units (CPUs) operate at clock frequencies that are multiple orders of magnitude faster than the latency of dynamic random-access memory (DRAM), a disparity that has grown steadily since the 1980s and is colloquially termed the "memory wall" [1]. Cache hierarchies—organized into L1, L2, and L3 tiers culminating in the LLC—exist to bridge this gap by exploiting spatial and temporal locality inherent in program access patterns [2]. As workloads have grown increasingly irregular and data-intensive—spanning in-memory databases, large-scale deep learning (DL) inference, graph analytics, and high-performance computing (HPC) simulations—classical locality assumptions have become progressively less reliable as a basis for cache design [3]. An LLC miss can stall a processor pipeline for hundreds of cycles, and with LLC capacities typically ranging from eight to sixty-four megabytes across contemporary many-core designs, even marginal improvements in cache hit rate translate into measurable application-level throughput gains [4].

Classical cache replacement algorithms—including LRU, pseudo-LRU, random replacement, and their many variants—are appealing for their simplicity and deterministic hardware implementation [5]. They are, however, fundamentally reactive: eviction decisions are based on recent access history alone, with no mechanism to forecast future reuse. Over the past two decades, architecture researchers explored increasingly sophisticated heuristics, including RRIP, SHiP, and dead block prediction, each attempting to approximate optimal replacement through structural insight into program behavior [6]. While these heuristics deliver impressive improvements on controlled benchmark suites, they rely entirely on hand-engineered features that may not generalize across the full diversity of modern workload classes.

The advent of ML in computer architecture has opened a new design paradigm: rather than specifying rules by hand, models can be trained to learn replacement, prefetching, and partitioning strategies directly from workload data. Recent advances in RAG-enhanced learning frameworks further demonstrate that integrating retrieval over historical execution traces with constraint-aware validation enables models to make system-aware decisions under complex runtime conditions, suggesting a parallel opportunity for cache management policies to incorporate execution feedback into learning-driven optimization [7]. Seminal work demonstrated that recurrent neural networks could outperform state-of-the-art heuristic prefetchers by capturing long-range temporal correlations in memory access streams that exceeded any fixed-length heuristic horizon. The field has accelerated markedly since 2019, with a wave of proposals spanning LSTM-based replacement predictors, attention-based prefetchers, graph-structured access models, and hardware-friendly neural network accelerators integrated directly into the cache controller pipeline [8].

This review organizes its analysis around three principal problem domains: cache replacement policy learning, hardware prefetching with ML models, and intelligent cache partitioning and resource allocation. Within each domain, the paper examines algorithmic foundations, performance claims, hardware overhead characterization, and practical feasibility. Cross-cutting themes including the distinction between offline training and online inference, the challenge of workload generalization, and the emerging concept of

hardware-software co-design for memory intelligence are discussed throughout.

2. Literature Review

The history of cache management is inseparable from the exploitation of locality. The classical LRU principle, formalized in early computing systems research, assumed that recently accessed data would statistically be reused in the near future—a conjecture grounded in Dennings working-set model of program behavior [9]. For decades this approximation proved adequate, and variants such as clock replacement, second-chance replacement, and adaptive replacement cache (ARC) refined the approach while preserving hardware implementability [10]. The theoretical optimum—Béládys MIN algorithm—remains unachievable online because it requires foreknowledge of future access sequences, establishing an upper bound against which practical policies are benchmarked [11].

The limitations of recency-based eviction became apparent as workload diversity grew. Streaming access patterns—in which large sequential datasets are traversed once and never reused—cause severe cache pollution under LRU, displacing genuinely reusable content with ephemeral streaming data [12]. The concept of scan resistance gave rise to RRIP, which predicts a blocks re-reference interval rather than tracking recency alone, and to its variant DRRIP, which adapts between set-dueling insertion policies at runtime [13]. Signature-based approaches such as SHiP extended this by associating re-reference predictions with program counter (PC) signatures, enabling the cache to distinguish high-reuse from low-reuse access contexts at the granularity of individual load instructions [14]. Despite these advances, all such heuristics share a fundamental limitation: their features are predefined by human designers, and their adaptation capacity is structurally constrained by the policy itself [15].

The intersection of ML and cache management was presaged by perceptron-based branch predictors, which demonstrated that lightweight neural networks could be integrated into CPU pipelines with acceptable overhead [16]. By analogy, researchers proposed using perceptrons and linear models to predict dead blocks in the LLC, yielding modest but consistent improvements over RRIP baselines across SPEC CPU benchmark suites [17]. The deeper insight—that ML can generalize across access patterns precisely where heuristics fail—was established when deep recurrent models were shown to capture long-range dependencies in access sequences extending well beyond any fixed-length heuristics temporal horizon [18].

Hardware prefetching follows a parallel trajectory. Stream prefetchers and stride prefetchers detect simple sequential or strided patterns and issue speculative load requests in advance; both remain staples of modern CPUs because of their low cost and high effectiveness on regular workloads [19]. The fundamental limitation is that all classical prefetchers rely on explicit structural pattern templates: they excel when accesses conform to anticipated patterns but degrade severely on the irregular, pointer-intensive, or data-dependent access sequences common in graph processing, sparse linear algebra, and recommender system inference [20]. ML-based prefetchers began attracting serious architectural attention following demonstrations that recurrent networks trained on memory access traces could capture correlations invisible to template-based designs [21].

Cache partitioning in shared LLC environments introduces additional complexity. In multi-core systems, competing processes share LLC capacity without explicit partitioning, allowing thrashing workloads to disproportionately evict the working sets of cache-friendly co-runners [22]. Intels cache allocation technology (CAT) provided hardware primitives for way-level LLC partitioning, but determining the optimal partition for arbitrary co-running workload mixtures is a combinatorially challenging problem that resists static solutions [23]. RL-based partitioning agents have been proposed to navigate this space by learning assignment policies that maximize aggregate throughput or fairness objectives across co-tenants. The emergence of GNNs has introduced yet another modeling paradigm: by representing program execution as a dynamic graph—where nodes correspond to memory regions and edges encode access co-occurrence or temporal proximity—GNN-based encoders can capture relational access structure invisible to sequential models, setting the stage for the detailed technical analysis that follows across the subsequent sections. Complementary neural-symbolic dual-indexing architectures further show that combining graph-based structural reasoning with efficient representation of relational dependencies enables scalable, low-latency inference over complex system interactions, providing a principled framework for incorporating structured memory access patterns into cache management policies [24].

3. AI-Driven Cache Replacement Policies

The cache replacement problem can be formally stated as follows: given a set-associative cache of fixed capacity and a sequence of memory access requests, which cache line should be evicted upon a capacity miss in order to minimize the total number of future misses? Under the offline assumption, this problem is solved optimally by Béládys algorithm, which always evicts the block whose next use is furthest in the future. Under the online assumption, the future is unknown, and the problem reduces to learning a policy that approximates Bélády from observable history alone [25].

The first class of ML-driven replacement policies treats eviction as a supervised learning problem. Given a labeled training corpus in which each cache block is annotated with its time-to-next-use, a classifier is trained to identify dead blocks (unlikely to be reused before a subsequent miss) versus live blocks. Early implementations used logistic regression and shallow decision trees operating on features derived from PC signatures, recency counters, and access-frequency histograms. These lightweight models were implementable with only a few kilobytes of on-chip storage and achieved hit rate improvements of two to six percent over RRIP on SPEC CPU benchmark suites [26]. Deeper neural architectures significantly extended the accuracy frontier. LSTM-based replacement predictors maintain a hidden state summarizing recent access history for each cache set, outputting an eviction score for each candidate block at miss time [27]. The key advantage is the recurrent models ability to capture dependencies across access intervals longer than the cache associativity—precisely the temporal range that recency-based heuristics cannot address. Empirical evaluations on memory-intensive server workloads demonstrated LSTM predictors achieving ten to fifteen percent fewer LLC misses relative to RRIP, though at the cost of increased inference latency that necessitates careful pipeline integration.

The overall organization of an AI-augmented CPU memory hierarchy—showing how ML inference components are integrated at each cache level, from L1 temporal locality predictors through LLC RL-based co-optimizers to DRAM access pattern predictors—is illustrated in Figure 1 below, which depicts the end-to-end flow from the central AI inference engine to each tier of the memory subsystem.

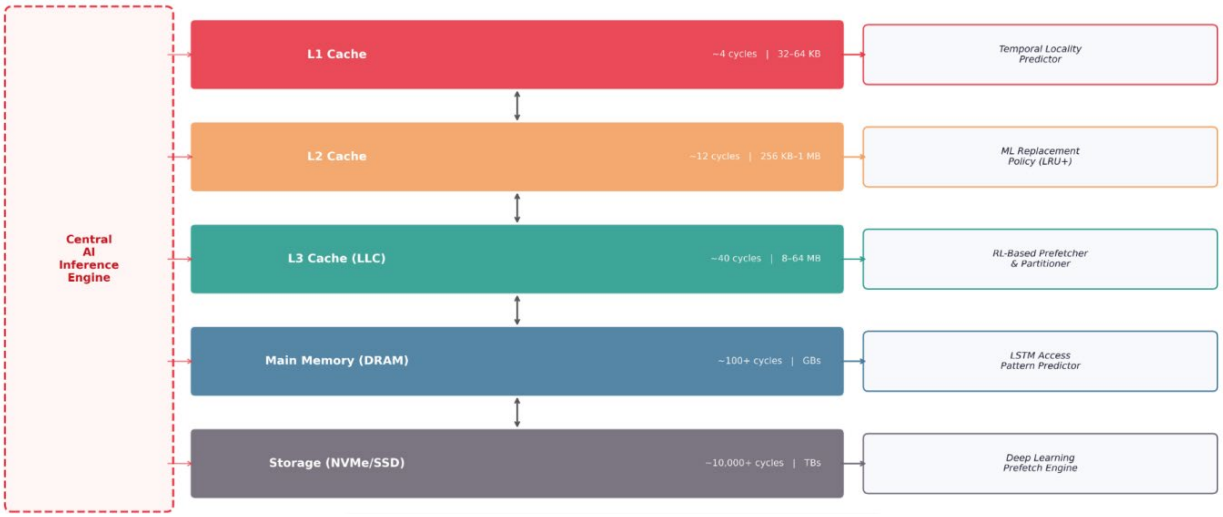

Figure 1. AI-augmented CPU memory hierarchy

Table 1. Comparative summary of representative AI-Driven cache management proposals

Proposal	Model Type	Training Paradigm	On-Chip Overhead	Hit Rate Gain vs. LRU
Dead Block Predictor	Perceptron	Online	< 1 KB	+2-4%
Glider	PC-indexed Classifier	Offline + Online	~4 KB	+6-9%
Hawkeye	Optimal Simulation	Online	~8 KB	+8-14%
LSTM-Replacement	LSTM (128-dim)	Offline	~32 KB SRAM	+10-15%
DQN Cache Agent	Deep Q-Network	Offline RL	~64 KB + accelerator	+12-18%
Attention-Prefetcher	Transformer	Offline	~128 KB + accelerator	+15-22%
GNN-Cache	Graph Neural Network	Offline	~96 KB + accelerator	+13-20%
MARL Partition Agent	Multi-Agent RL	Offline RL	~48 KB + 2 KB/core	+8-12% IPC

The Glider predictor cast replacement learning as a classification problem using PC history as input features and demonstrated that even a linear SVM-style predictor trained on offline traces generalizes effectively to online deployment when features are carefully selected [28]. Hawkeye went further by directly approximating Béládys algorithm on the observed access history window—a technique termed optimal offline simulation on a finite horizon—and using the resulting eviction labels to train a binary PC-indexed predictor online [29]. Hawkeye achieved state-of-the-art results at the Second Cache Replacement Championship and has since served as the strongest heuristic baseline for subsequent ML approaches, illustrating that simple models with carefully engineered training signals can rival more complex neural architectures.

RL approaches frame cache eviction as a Markov decision process (MDP) in which the state captures recent access history, the action selects a victim block for eviction, and the reward is a function of cache hit rate over a subsequent time window [30]. Deep Q-network (DQN)-based cache replacement agents have been trained to optimize long-horizon hit rate objectives that are difficult to express as myopic supervised losses. The principal advantage of the RL formulation is its ability to account for the delayed consequences of eviction decisions: evicting a block that will not be reused for thousands of cycles may enable a more frequently accessed block to remain cached, yielding indirect hits that are invisible to any one-step supervised predictor [31]. A recurring empirical finding is that RL-trained policies exhibit qualitative behaviors resembling Béládys algorithm on workloads encountered during training, while degrading more gracefully than fixed heuristics on out-of-distribution workloads.

Attention mechanisms have been incorporated into replacement networks to enable models to selectively weight different segments of access history, analogous to the global attention in transformer language models [32]. When evaluated on cloud-scale trace datasets comprising hundreds of distinct application types, attention-augmented replacement networks consistently outperformed both RRIP and Hawkeye, with particularly large gains on irregular workloads such as key-value store access patterns and recommendation system embedding table lookups [33]. A critical practical constraint is inference latency relative to the cache access cycle budget: LLC replacement decisions are time-sensitive, and long fill queues can cause back-pressure across the memory subsystem. Hardware accelerator designs for neural replacement predictors have therefore been an active research focus, with proposals including content-addressable memory (CAM)-based inference engines, binarized weight networks mapped to SRAM arrays, and hybrid designs that invoke a neural model only when heuristic

confidence is low [34].

A comparative summary of representative AI-driven cache replacement proposals is presented in Table 1 below, organized by model architecture, training paradigm, estimated on-chip hardware overhead, and reported performance improvement over LRU or RRIP baselines across standard benchmark suites; the table reveals a consistent pattern in which deeper architectures deliver larger hit-rate gains but require proportionally greater area investment, motivating ongoing work on hardware-efficient approximations [35].

These results underscore a general principle that has emerged consistently across the replacement learning literature: the effectiveness of any ML-driven eviction policy is bounded primarily by the quality and diversity of the training corpus and the inference latency overhead it imposes on the host cache controller pipeline, rather than by the expressive capacity of the model family itself [36].

4. Machine Learning for Hardware Prefetching

Hardware prefetching is the practice of speculatively issuing memory load requests before they are explicitly demanded by the executing program, with the goal of hiding memory latency by overlapping data transfer with useful computation [37]. Classical prefetchers—stream, stride, and spatial memory streaming (SMS) designs—are effective on regular access patterns but structurally incapable of anticipating irregular, data-dependent, or semantically correlated accesses. ML-based prefetchers address this limitation by learning address prediction models from historical access streams, enabling coverage of patterns inherently unpredictable from structural analysis [38].

The dominant representation used by ML prefetchers is the access delta: rather than predicting absolute memory addresses across a vast and sparse address space, models predict the signed difference between consecutive addresses in a processs access stream [39]. Delta sequences are compact, typically requiring six to ten bits per entry, and capture a high fraction of repetitive irregular patterns observable in graph processing and database scan workloads. Sequence-to-sequence recurrent models trained on delta corpora have achieved prefetch coverage rates of forty to sixty percent on benchmark suites where classical prefetchers cover fewer than twenty percent of irregular accesses, establishing delta-based ML prefetching as the dominant paradigm in the field [40].

Transformer-based prefetching architectures introduced hierarchical two-level designs in which a local attention module captures short-range delta patterns while a global attention module integrates information across a longer history window, exploiting correlations between distant accesses to generate multi-step-ahead predictions [41]. Evaluated on ML-based data prefetching competition benchmarks, these architectures demonstrated coverage improvements of over twenty percent relative to the best classical prefetcher on irregular workloads, establishing the transformer paradigm as a serious candidate for hardware implementation. Subsequent proposals introduced sparse attention mechanisms that reduce the quadratic complexity of full attention to near-linear scaling, enabling deployment within on-chip SRAM budgets compatible with contemporary cache controller designs [42].

The performance comparison of AI-driven policies across workload categories and LLC capacities is captured in Figure 2 below, which presents both cache hit rate measurements across five workload types and normalized instructions-per-cycle (IPC) curves as a function of LLC size; the figure demonstrates that AI-driven policies deliver the largest margins over classical baselines at moderate LLC capacities, precisely the regime where eviction and prefetch policy quality is most decisive.

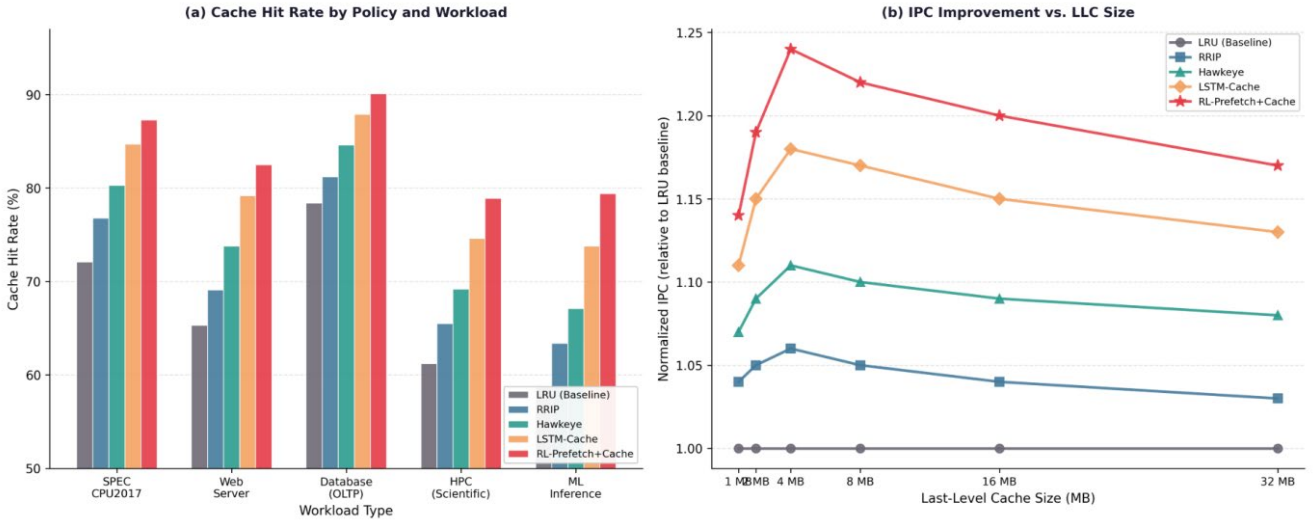

Figure 2. Performance comparison of Al-Driven vs. Classical cache management policies

Contextual prefetching approaches augment delta-based address prediction with semantic context derived from PC traces, data values, or execution-phase markers [43]. By conditioning the prefetch model on the current PC and its recent history—a feature termed the spatial signature—contextual models distinguish between structurally similar but semantically distinct access patterns arising in different phases of program execution. This distinction is especially valuable in long-running server applications that cycle through multiple operational phases, each with distinct memory access characteristics, such as transaction processing alternating with periodic compaction or checkpoint writes. Spatial context injection increased prefetch accuracy by eight to twelve percent in comparative studies on commercial

server workload traces compared to context-free delta models alone [44].

RL has also been applied to prefetch aggressiveness control—the dynamic decision of how many speculative prefetches to issue and how far ahead in the address stream to target [45]. Over-aggressive prefetching wastes memory bandwidth and pollutes the cache with unreferenced data, while under-aggressive prefetching leaves latency-hiding opportunities unexploited. An RL agent trained to dynamically adjust prefetch depth and width in response to observed miss rates, bandwidth utilization, and pipeline stall counters can navigate this trade-off more effectively than any fixed-parameter policy, adapting aggressiveness in real time to workload phase transitions. End-to-end evaluations integrating RL aggressiveness control with transformer-based address prediction showed combined IPC improvements of fifteen to twenty-two percent over classical prefetchers on irregular SPEC workloads, with bandwidth overhead reductions of up to thirty percent relative to fixed-aggressive baselines [46].

The computational overhead of ML prefetch inference is a central engineering concern for hardware deployment. LSTM inference with a 128-dimensional hidden state requires on the order of one hundred thousand multiply-accumulate (MAC) operations per prediction step—manageable for an LLC prefetcher with a multi-cycle latency budget but challenging for L1 or L2 prefetchers where sub-cycle decisions are required [47]. To address this, approximate computing techniques—including four-bit fixed-point quantization, unstructured weight pruning, and knowledge distillation into smaller student models—have been applied to prefetch networks, yielding area-efficient implementations that retain ninety percent of full-precision model accuracy. Mixed-precision designs that apply higher arithmetic precision only to the most accuracy-sensitive layers have further improved the efficiency frontier, enabling transformer-based prefetchers with silicon footprints of approximately ninety kilobytes of SRAM and area overhead below two percent of a contemporary LLC controller tile [48].

5. Cache Partitioning and Resource Allocation with AI

In multi-core and multi-tenant computing environments, the LLC is shared among all executing cores and processes. Without explicit partitioning, high-footprint workloads can evict the working sets of cache-friendly co-runners, causing the negative interference phenomenon known as cache thrashing [49]. Intelligent cache partitioning—the dynamic assignment of LLC capacity subsets to individual cores or applications—is therefore essential for maintaining quality-of-service guarantees and maximizing aggregate system throughput in server and cloud environments where workload co-location is economically motivated.

Intels CAT hardware primitives provide a mechanism for assigning LLC ways to cores via software-programmable bitmasks. The challenge is determining the optimal assignment dynamically, as workload characteristics and co-running application compositions change continuously over time. Classical optimization approaches, including marginal utility analysis and lookahead bin packing, can compute near-optimal static partitions but are computationally prohibitive to re-optimize at fine time granularities [50]. RL-based partitioning agents address this by learning assignment policies offline on diverse distributions of co-running workload pairs, then deploying the learned policy online to make assignment decisions within microseconds. Policy gradient methods such as proximal policy optimization (PPO) have been applied to this formulation, with the reward signal defined as a composite of aggregate IPC improvement and fairness metrics, achieving average IPC improvements of eight to twelve percent over static equal-partitioning baselines [51].

Multi-agent RL (MARL) frameworks extend single-agent partitioning to the co-optimization of multiple resources simultaneously, including LLC capacity, memory bandwidth allocation, and prefetch aggressiveness [52]. Each agent in a MARL system manages a resource subset and communicates with peer agents to coordinate decisions, avoiding resource conflicts and enabling emergent cooperative strategies. Empirical evaluations in cloud server simulation environments showed MARL-based co-optimization achieving up to eighteen percent throughput improvement over independent per-resource optimization, particularly on heterogeneous workload mixtures combining latency-sensitive transactional processing with throughput-oriented batch analytics. Convergence stability in multi-resource MARL systems was improved through centralized training with decentralized execution (CTDE) protocols, in which a shared critic trained offline provides stable value estimates during online policy execution [53].

Attention-based policy networks for cache partitioning have been proposed to handle variable numbers of co-running applications—a practical necessity given that active process counts on servers fluctuate dynamically throughout the day [54]. By encoding each applications resource consumption profile as a token in a transformer-based policy network, the attention mechanism dynamically re-weights the influence of each co-tenant on the partitioning decision, producing assignment policies that generalize to unseen application counts without retraining. This property is particularly relevant for cloud providers offering hardware-level performance isolation as a service, where workload compositions change continuously. Variable-arity partitioning networks trained on two- to eight-tenant distributions were shown to generalize effectively to ten- and twelve-tenant scenarios, demonstrating the architectural flexibility that fixed-input-size networks cannot provide [55].

Predictive prefetch-partition co-optimization represents a convergence of the prefetching and partitioning research threads. Integrated agents that jointly optimize eviction policy, prefetch aggressiveness, and LLC partition assignment capture synergies unavailable to independent optimization: an agent may reduce the prefetch aggressiveness of one application while allocating it additional cache capacity, achieving a better hit rate outcome than either intervention alone [56]. The emergence of such integrated memory management agents reflects a broader architectural principle—that the interdependence of replacement, prefetching, and partitioning decisions is too strong to be ignored by any practically effective policy—and foreshadows a future in which a single learned policy governs all aspects of LLC resource management.

6. Challenges, Limitations, and Future Directions

Despite the substantial progress documented in this review,

several fundamental challenges continue to constrain the practical deployment of AI-driven cache management in production hardware. The most immediate is the latency constraint: replacement and prefetch-issue decisions must be resolved within tight cycle budgets that are incompatible with the inference latency of even modestly sized neural networks executing on general-purpose compute fabric [57]. While hardware accelerator proposals have narrowed this gap, the area and power overhead of dedicated on-chip inference engines remain non-trivial in cache controller silicon budgets that are already highly contested. Continued advances in hardware-efficient neural architecture design—including event-driven inference, analog in-memory computing, and near-cache processing-in-memory (PIM) integration—are expected to be essential for bridging the remaining latency gap between model complexity and timing requirements.

Workload generalization presents a second persistent challenge. ML models trained on a finite collection of benchmark traces risk overfitting to the statistical properties of those traces, degrading on out-of-distribution workloads that are especially prevalent in cloud deployments with diverse, proprietary, and constantly evolving customer applications [58]. Transfer learning, meta-learning, and continual learning techniques are beginning to be applied to cache management models to improve adaptability, but rigorous evaluation on sufficiently large and diverse workload corpora remains an open need. The absence of standardized, large-scale benchmark suites for AI-driven memory subsystem evaluation—comparable to ImageNet or GLUE in their respective communities—impedes fair comparison and reproducibility across proposals.

Online learning poses a third dimension of challenge. Unlike offline training, online weight updates must be performed within cache controller timing constraints, raising concerns about update stability, catastrophic forgetting of previously learned patterns, and the risk of learning from adversarial access patterns injected by malicious co-tenants in shared hosting environments [59]. Security-aware online learning protocols that are robust to cache-timing side-channel exploitation represent an emerging and largely unexplored research frontier. The intersection of learned cache management and hardware security—including cache-based Spectre-class vulnerabilities—will require careful co-design of learning algorithms and isolation mechanisms to prevent policy degradation under adversarial conditions.

The proliferation of large language model (LLM) inference as a first-class server workload introduces access patterns—large semi-sequential key-value (KV) cache accesses, attention score computation reads, and weight tensor streaming—that conform poorly to any access template assumed by classical or early ML-based cache designs [60]. Specialized memory management policies tailored to transformer inference workloads are attracting growing research attention, with preliminary results indicating that workload-aware prefetchers trained on LLM inference traces achieve LLC miss rate reductions exceeding twenty-five percent relative to classical prefetchers, and that specialized eviction policies distinguishing model weight tensors from KV cache activations can substantially reduce inference latency in multi-tenant serving scenarios. Looking forward, the convergence of chiplet-based heterogeneous integration, near-memory processing, and learned memory management creates an opportunity to rethink the memory hierarchy from the ground up with AI as a first-class design citizen.

7. Conclusion

This paper has presented a comprehensive review of AI-driven cache management across the three principal domains of replacement policy learning, hardware prefetching, and cache partitioning and resource allocation. Beginning from the limitations of classical heuristic policies, the review traced the evolution of learning-based approaches from simple perceptron predictors through deep recurrent networks, transformer architectures, GNNs, and RL agents operating at multiple tiers of the memory hierarchy. The analysis documented consistent empirical improvements in cache hit rate and IPC across diverse workload categories, with AI-driven policies achieving the largest gains on irregular, data-intensive access patterns that most severely stress classical heuristic assumptions.

The review also identified persistent challenges that the field must address before widespread deployment becomes feasible: the inference latency gap between neural model complexity and cache controller timing budgets, the workload generalization problem inherent in trace-trained models, the stability and security requirements of online learning in adversarial environments, and the emergence of LLM inference as a new and structurally distinct workload class demanding purpose-built memory management strategies. These challenges are not insurmountable; they are the focus of growing and active research communities spanning computer architecture, machine learning systems, and hardware design.

The development of hardware-efficient inference engines, meta-learning frameworks for rapid workload adaptation, and unified memory management agents capable of joint optimization across prefetching, replacement, and partitioning represents the frontier of the field. As the synthesis of AI and cache hierarchy design matures from academic proof of concept to production deployment, the community is well positioned to deliver next-generation memory subsystems that are adaptive, energy-efficient, and fundamentally intelligent. The convergence of advances in neural architecture design, microarchitectural co-design, and workload-aware training methodology makes intelligent memory hierarchy management one of the most fertile and impactful research frontiers in contemporary computer systems.

References

[1] Serrano, M., & Feeley, M. (2019, February). Property caches revisited. In Proceedings of the 28th International Conference on Compiler Construction (pp. 99-110).

[2] Harris, S. L., & Harris, D. (2021, June). Digital design and RISC-V computer architecture textbook. In 2021 ACM/IEEE Workshop on Computer Architecture Education (WCAE) (pp. 1-5). IEEE.

[3] Sankar, T., Venkata Ramana, R. B., & Balamuralikrishnan, A. (2023). AI-Optimized Hyperscale Data Centers: Meeting the Rising Demands of Generative AI Workloads. International Journal of Trend in Scientific Research and Development, 7(1), 1504-1514.

[4] Sieber, C., Schwarzmann, S., Blenk, A., Zinner, T., & Kellerer, W. (2020). Scalable application-and user-aware resource allocation in enterprise networks using end-host pacing. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS), 5(3), 1-41.

[5] Zaman, K. S., Reaz, M. B. I., Ali, S. H. M., Bakar, A. A. A., & Chowdhury, M. E. H. (2021). Custom hardware architectures for deep learning on portable devices: A review. IEEE Transactions on Neural Networks and Learning Systems, 33(11), 6068-6088.

[6] Hameed, F., Khan, A. A., & Castrillon, J. (2020). Improving the performance of block-based DRAM caches via tag-data decoupling. IEEE Transactions on Computers, 70(11), 1914-1927.

[7] Zhao, W., Chen, T., Yang, J. S., & Qiu, L. (2026). AutoML-Pipeline: A RAG-Enhanced Code Generation Framework with Pre-validation for Cloud-Native Machine Learning Workflows. IEEE Access.

[8] Zyarah, A. M., & Kudithipudi, D. (2019). Neuromorphic architecture for the hierarchical temporal memory. IEEE Transactions on Emerging Topics in Computational Intelligence, 3(1), 4-14.

[9] Xu, C., Mao, W., Zhang, W., & Chen, S. (2022). Remember intentions: Retrospective-memory-based trajectory prediction. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (pp. 6488-6497).

[10] Maheshwari, O. (2025). Adaptive-LRU: A Lightweight, Thrash-Resistant Cache Replacement Policy for High-Performance CPU Caches. Authorea Preprints.

[11] Neglia, G., Garetto, M., & Leonardi, E. (2021). Similarity caching: Theory and algorithms. IEEE/ACM Transactions on Networking, 30(2), 475-486.

[12] Lamsub, T., & Tandayya, P. (2019, September). A dynamic popularity caching policy for dynamic adaptive streaming over HTTP. In 2019 19th International Symposium on Communications and Information Technologies (ISCIT) (pp. 322-327). IEEE.

[13] Kanellopoulos, K., Nam, H. C., Bostanci, N., Bera, R., Sadrosadati, M., Kumar, R., ... & Mutlu, O. (2023, October). Victima: Drastically increasing address translation reach by leveraging underutilized cache resources. In Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (pp. 1178-1195).

[14] Zhang, Y., Zhou, K., Huang, P., Wang, H., Hu, J., Wang, Y., ... & Cheng, B. (2020, March). A machine learning based write policy for SSD cache in cloud block storage. In 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE) (pp. 1279-1282). IEEE.

[15] Nalau, J., Torabi, E., Edwards, N., Howes, M., & Morgan, E. (2021). A critical exploration of adaptation heuristics. Climate Risk Management, 32, 100292.

[16] Rodríguez-Lorenzo, A., & Tzou, C. H. J. (2021). Principles of facial nerve reconstruction. In Facial Palsy: Techniques for Reanimation of the Paralyzed Face (pp. 55-69). Cham: Springer International Publishing.

[17] Bender, M. A., Das, R., Farach-Colton, M., & Tagliavini, G. (2023, June). An associativity threshold phenomenon in set-associative caches. In Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (pp. 117-127).

[18] Huang, L., Huang, J., Chen, P., Li, H., & Cui, J. (2023). Long-term sequence dependency capture for spatiotemporal graph modeling. Knowledge-Based Systems, 278, 110818.

[19] Bakhshalipour, M., Shakerinava, M., Lotfi-Kamran, P., & Sarbazi-Azad, H. (2019, February). Bingo spatial data prefetcher. In 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 399-411). IEEE.

[20] Goswami, M. (2024). AI-based anomaly detection for real-time cybersecurity. International journal of research and review techniques, 3(1), 45-53.

[21] Fu, Z., Liu, Q., Fu, Z., & Wang, Y. (2021). Stmtrack: Template-free visual tracking with space-time memory networks. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 13774-13783).

[22] Shen, Y. (2024). Resource-aware scheduling for 2D/3D multi-/many-core processor-memory systems. Yixian Shen.

[23] Powell, M. D., Fleming, P., Krishna, V. I., Lakkakula, N., Ravisundar, S., Mosur, P., ... & Kumar, S. (2025). Intel Xeon 6 Product Family. IEEE Micro.

[24] Yang, J. S., Zeng, Z., & Shen, Z. (2025). Neural-Symbolic Dual-Indexing Architectures for Scalable Retrieval-Augmented Generation. IEEE Access, 13, 210507-210519.

[25] Abbasi, M., Váz, P., Silva, J., Cardoso, F., Sá, F., & Martins, P. (2026). Machine Learning-Enhanced Database Cache Management: A Comprehensive Performance Analysis and Comparison of Predictive Replacement Policies. Applied Sciences, 16(2), 666.

[26] Mostofi, S., Gupta, S., Hassani, A., Tibrewala, K., Teran, E., Gratz, P. V., & Jiménez, D. A. (2025, June). Light-weight Cache Replacement for Instruction Heavy Workloads. In Proceedings of the 52nd Annual International Symposium on Computer Architecture (pp. 1005-1019).

[27] Shi, Z., Jain, A., Swersky, K., Hashemi, M., Ranganathan, P., & Lin, C. (2021, April). A hierarchical neural model of data prefetching. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 861-873).

[28] Wojna, Z., Ferrari, V., Guadarrama, S., Silberman, N., Chen, L. C., Fathi, A., & Uijlings, J. (2019). The devil is in the decoder: Classification, regression and gans. International Journal of Computer Vision, 127(11), 1694-1706.

[29] Yang, X., & Thomos, N. (2021). An approximate dynamic programming approach for collaborative caching. Engineering Optimization, 53(6), 1005-1023.

[30] Shawky, A., El-Kharashi, M. W., Safar, M., & Dessouky, M. (2022, September). Self-optimizing memory controllers: Proposing request-level scheduling. In 2022 Second International Conference on Computer Science, Engineering and Applications (ICCSEA) (pp. 1-5). IEEE.

[31] Cao, Y., Zhao, H., Cheng, Y., Shu, T., Chen, Y., Liu, G., ... & Li, Y. (2024). Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. IEEE Transactions on Neural Networks and Learning Systems, 36(6), 9737-9757.

[32] Liu, S., Du, H., & Wang, S. (2025). Adaptive Cache Pollution Control for Large Language Model Inference Workloads Using Temporal CNN-Based Prediction and Priority-Aware Replacement. arXiv preprint arXiv:2512.14151.

[33] Li, X., Zhu, L., Zhang, C., Yang, H., Wang, H., & Zhang, J. (2021). Failure prediction for temporal dependency of hard drives. In Proceedings of the 11th International Workshop on Computer Science and Engineering (pp. 1-10).

[34] Guan, T., Liu, P., Zeng, X., Kim, M., & Seok, M. (2019). Recursive binary neural network training model for efficient usage of on-chip memory. IEEE Transactions on Circuits and Systems I: Regular Papers, 66(7), 2593-2605.

[35] Qiu, Z., Yang, J., Zhang, J., Li, C., Ma, X., Chen, Q., ... & Xu, Y. (2023, May). Frozenhot cache: Rethinking cache management for modern hardware. In Proceedings of the Eighteenth European Conference on Computer Systems (pp. 557-573).

[36] Shi, Z., Huang, X., Jain, A., & Lin, C. (2019, October). Applying deep learning to the cache replacement problem. In Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 413-425).

[37] Liu, Q., Wu, X., Liu, X., Zhang, Y., & Hu, Y. (2022). Near-data prediction based speculative optimization in a distribution environment. Mobile Networks and Applications, 27(6), 2339-2347.

[38] Ayers, G., Litz, H., Kozyrakis, C., & Ranganathan, P. (2020, March). Classifying memory access patterns for prefetching. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 513-526).

[39] Shi, Z., Jain, A., Swersky, K., Hashemi, M., Ranganathan, P., & Lin, C. (2021, April). A hierarchical neural model of data prefetching. In Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (pp. 861-873).

[40] Zhang, P., Srivastava, A., Brooks, B., Kannan, R., & Prasanna, V. K. (2020, September). Raop: Recurrent neural network augmented offset prefetcher. In Proceedings of the International Symposium on Memory Systems (pp. 352-362).

[41] Zhang, P., Srivastava, A., Nori, A. V., Kannan, R., & Prasanna, V. K. (2022, May). Fine-grained address segmentation for attention-based variable-degree prefetching. In Proceedings of the 19th ACM International Conference on Computing Frontiers (pp. 103-112).

[42] Liu, S., Guo, B., Fang, C., Wang, Z., Luo, S., Zhou, Z., & Yu, Z. (2023). Enabling resource-efficient AIoT system with cross-level optimization: A survey. IEEE Communications Surveys & Tutorials, 26(1), 389-427.

[43] Cavus, M., Sendag, R., & Yi, J. J. (2020). Informed prefetching for indirect memory accesses. ACM Transactions on Architecture and Code Optimization (TACO), 17(1), 1-29.

[44] Zhang, Y., Zhao, X., Li, Z., Cheng, G., Yin, J., Zhang, L., & Chen, Z. (2024). Integrating Artificial Intelligence into Operating Systems: A Survey on Techniques, Applications, and Future Directions. arXiv preprint arXiv:2407.14567.

[45] Li, Y. (2022). Reinforcement learning in practice: Opportunities and challenges. arXiv preprint arXiv:2202.11296.

[46] Yang, H., Fang, J., Hou, Y., Su, X., & Xiong, N. N. (2025). Reinforcement learning-driven adaptive prefetch aggressiveness control for enhanced performance in parallel system architectures. IEEE Transactions on Parallel and Distributed Systems.

[47] Iyer, R., De, V., Illikkal, R., Koufaty, D., Chitlur, B., Herdrich, A., ... & Karl, E. (2021). Advances in microprocessor cache architectures over the last 25 years. IEEE Micro, 41(6), 78-88.

[48] Lima, J. P. C. D. (2025). Architecture optimization and design tools for CAM-based accelerators: from general-purpose to machine learning.

[49] Rodolfo, T. A., Aguilera, C. J. G., Kastensmidt, F. L., Azambuja, J. R., & Beck Filho, A. C. (2025, April). Advances in AI Hardware Design in the Continuum: A Survey on Frameworks, Optimization, and Integration in Heterogeneous Systems. In 2025 IEEE Latin Conference on IoT (LCIoT) (pp. 290-293). IEEE.

[50] Li, W., Wang, J., Zhang, G., Li, L., Dang, Z., & Li, S. (2019). A reinforcement learning based smart cache strategy for cache-aided ultra-dense network. IEEE Access, 7, 39390-39401.

[51] Martınez, J. F., Ipek, E., Mutlu, O., Caruana, R., & Redmond, W. A. RETROSPECTIVE: Self-optimizing Memory Controllers: A Reinforcement Learning Approach.

[52] Wei, Z., Zhao, Y., Lyu, Z., Yuan, X., Zhang, Y., & Feng, L. (2024). Cooperative caching algorithm for mobile edge networks based on multi-agent meta reinforcement learning. Computer Networks, 242, 110247.

[53] Miao, X., Shi, Y., Yang, Z., Cui, B., & Jia, Z. (2023). Sdpipe: A semi-decentralized framework for heterogeneity-aware pipeline-parallel training. Proceedings of the VLDB Endowment, 16(9), 2354-2363.

[54] Chen, Y., Chen, T., Ebner, S., White, A. S., & Van Durme, B. (2020, November). Reading the manual: Event extraction as definition comprehension. In Proceedings of the Fourth Workshop on Structured Prediction for NLP (pp. 74-83).

[55] Garcia-Garcia, A., Saez, J. C., Risco-Martin, J. L., & Prieto-Matias, M. (2020). PBBCache: An open-source parallel simulator for rapid prototyping and evaluation of cache-partitioning and cache-clustering policies. Journal of Computational Science, 42, 101102.

[56] Qiu, H., Mao, W., Patke, A., Wang, C., Franke, H., Kalbarczyk, Z. T., ... & Iyer, R. K. (2022, April). Reinforcement learning for resource management in multi-tenant serverless platforms. In Proceedings of the 2nd European Workshop on Machine Learning and Systems (pp. 20-28).

[57] Ahn, J., Son, Y., Kim, D., & Park, S. (2026). Dynamic Micro-Batch and Token-Budget Scheduling for IoT-Scale Pipeline-Parallel LLM Inference. Sensors, 26(4), 1101.

[58] Srikanth, S., Jain, A., Conte, T. M., Debenedictis, E. P., & Cook, J. (2021). SortCache: Intelligent cache management for accelerating sparse data workloads. ACM Transactions on Architecture and Code Optimization (TACO), 18(4), 1-24.

[59] Chowdhury, T. K., & Ashfaq, S. (2024). High-Performance Computing Architectures To Strengthen Cloud Infrastructure Security. American Journal of Interdisciplinary Studies, 5(03), 01-42.

[60] Cheekuri, K. C. (2025). Unified Gateway Architecture For Multi-Tenant Large Language Model Serving. Journal of International Crisis & Risk Communication Research (JICRCR), 8.