Learning-Augmented Systems

HAILIANG ZHAO* and PENG CHEN*, Zhejiang University, China
XUEYAN TANG, Nanyang Technological University, Singapore
JIANWEI YIN, Zhejiang University, China
SHUIGUANG DENG††, Zhejiang University, China

We argue for a new paradigm in systems design: learning-augmented algorithms that combine the predictive power of machine learning with the robustness of classical methods. By embedding intelligence wisely and preserving fail-safe guarantees, these systems achieve both high performance and resilience in real-world environments.

CCS Concepts: • **Computing methodologies** → **Machine learning**; • **Theory of computation** → **Theory and algorithms for application domains**.

Additional Key Words and Phrases: Learning-Augmented Systems, Algorithms with Predictions

ACM Reference Format:
Hailiang Zhao, Peng Chen, Xueyan Tang, Jianwei Yin, and Shuiguang Deng. 2025. Learning-Augmented Systems. *J. ACM* 37, 4, Article 111 (August 2025), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 The Learning-Augmented Algorithmic Framework

At the core of computer systems lies a suite of algorithms designed to manage scarce resources online: memory, bandwidth, and compute cycles. For decades, these algorithms have been crafted under the assumption of worst-case adversarial inputs, ensuring robustness through conservative decision-making. While this paradigm offers strong theoretical guarantees, most notably via competitive analysis [3], it often sacrifices real-world efficiency. In practice, workloads are rarely adversarial; they exhibit patterns, temporal locality, and statistical regularities that can be learned and exploited.

Recent years, an algorithmic framework called *learning-augmented algorithm* arises. It harmonizes the rigor of classical algorithm design with the adaptability of machine learning (ML). Also known as *algorithms with predictions* [6], learning-augmented algorithms use ML-generated hints about future events, such as upcoming memory accesses, job arrivals, or request sequences, to guide decisions, achieving performance that adapts to actual workload dynamics while preserving worst-case resilience. This framework has redefined the boundaries of what is possible in online computation, offering a principled alternative to both rigid worst-case strategies and brittle ML-driven heuristics. Pioneered by Lykouris and Vassilvitskii [5] in the context of online caching and expanded by Purohit et al. [7] to scheduling and beyond, the learning-augmented algorithm paradigm introduces

*Both authors contributed equally to this research.
†Corresponding author.

Authors' Contact Information: Hailiang Zhao, hliangzhao@zju.edu.cn; Peng Chen, pgchen@zju.edu.cn, Zhejiang University, China; Xueyan Tang, Nanyang Technological University, Singapore, asxytang@ntu.edu.sg; Jianwei Yin, Zhejiang University, China, zjuyjw@zju.edu.cn; Shuiguang Deng, Zhejiang University, China, dengsg@zju.edu.cn.

ACM 1557-735X/2025/8-ART111
https://doi.org/10.1145/nnnnnnn.nnnnnnn

a new vocabulary for evaluating algorithmic performance. Rather than a single competitive ratio,[1] the framework assesses algorithms along three interrelated dimensions:

- *Consistency:* The competitive ratio when predictions are perfect. An algorithm with high consistency matches or approaches the performance of an optimal offline oracle when the future is known exactly.
- *Robustness:* The competitive ratio bound under arbitrarily inaccurate predictions. A robust algorithm ensures that poor predictions do not lead to catastrophic performance degradation.
- *Smoothness:* The rate at which performance degrades as the prediction error increases. A smooth algorithm transitions gracefully from consistency to robustness, avoiding abrupt cliffs in quality.

These metrics collectively define a performance landscape where the ideal learning-augmented algorithms dominate both traditional online algorithms (which ignore predictions) and naive "trust-all" strategies (which blindly follow them).

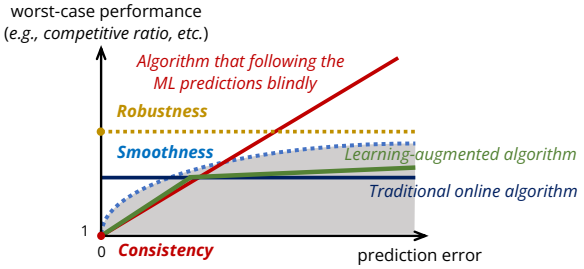

Fig. 1. Performance trade-offs across algorithmic paradigms: traditional online algorithms (flat, worst-case bounded), blind trust in predictions (high reward, high risk), and learning-augmented algorithms (adaptive, smooth, and robust). The shaded region highlights the ideal operating envelope.

As illustrated in Figure 1, the power of learning-augmented algorithms lies in their ability to achieve near-optimal performance when predictions are accurate (ideal consistency), degrade gracefully as prediction quality declines (smooth interpolation), and fall back safely to robust worst-case guarantees when predictions are completely wrong (strong robustness). This balance has fueled rapid theoretical progress, with learning-augmented algorithms now applied to caching [4, 5], scheduling [7], network optimization [1], and beyond. Yet, despite their theoretical elegance, a critical gap remains: *how to deploy learning-augmented algorithms effectively in real systems.* The transition from theory to practice is fraught with practical constraints that are rarely captured in idealized models.

2 From Theory to Practice: Bridging the Systems Gap

The theoretical learning-augmented algorithm framework assumes clean, cost-free access to predictions. In reality, integrating ML into system algorithms introduces overheads that can easily negate performance gains. Consider a learning-augmented cache in a low-latency inference pipeline: if the prediction model requires 30ms to estimate the next access time, but the target model inference completes in just 20ms, then the predictor itself becomes the bottleneck. The very component designed to improve efficiency ends up degrading performance, turning a would-be accelerator into a liability. This is not an edge case, but a fundamental tension between prediction cost and system benefit. As shown in Figure 2, three key challenges stand in the way of practical deployment:

[1]Competitive ratio is a metric used to evaluate the worst-case performance of an online algorithm. Generally, it is defined as the maximum cost ratio between an online algorithm and an offline optimum over all possible arrival instances.

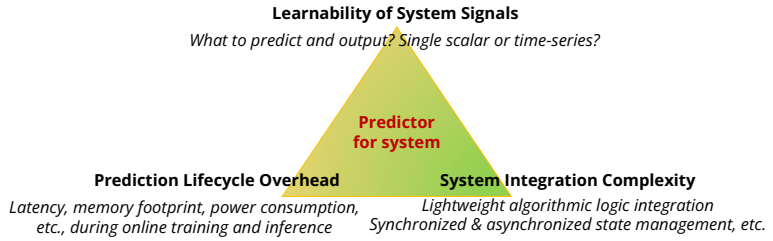

Fig. 2. Key challenges when applying learning-augmented algorithms to systems.

(1) *Prediction Lifecycle Overhead:* Encompasses both inference latency (e.g., prediction query time) and training cost (e.g., data logging, model updates, synchronization). In online settings, frequent retraining to adapt to shifting workloads can introduce CPU, memory, and I/O burdens that compete with core system tasks. For example, a predictor updated every few seconds may saturate CPU resources or delay critical operations.
(2) *Learnability of System Signals:* Not only must system signals be predictable (e.g., access patterns), but the predictor must adapt quickly to changes. This includes challenges like concept drift, cold-start for new items, and feedback loop delays (e.g., when access labels arrive late). Online learning exacerbates this: fast adaptation is needed, but noisy or sparse updates can degrade model quality.
(3) *System Integration Complexity:* The mechanism that fuses predictions with algorithmic logic must remain lightweight and resilient. This includes managing state consistency between the system and predictor (e.g., synchronized clocks, feature alignment), handling asynchronous model updates, and ensuring fast fallbacks when predictions fail, whether due to inaccuracy or unavailability during retraining.

To overcome these challenges, we advocate for a *systems-first* design philosophy for learning-augmented algorithms. This means rethinking the learning-augmented algorithm stack not just as a theoretical construct, but as a co-designed system where prediction, decision logic, and hardware constraints are optimized together. Our key design principles are as follows: (i) *Lightweight Prediction Interface:* Predictors must be fast, low-latency, and ideally precomputed or cached. Techniques such as model distillation, quantization, or sketch-based feature extraction can reduce prediction cost by orders of magnitude. (ii) *System-Aware Integration:* The algorithm must be aware of the underlying hardware, e.g., GPU memory hierarchies, NUMA topology, or vectorized instructions, to avoid data movement bottlenecks and maximize throughput. (iii) *Error Resilience with Fast Fallbacks:* The system should detect prediction errors quickly via lightweight consistency checks and revert to a robust baseline without incurring expensive recovery paths.

Crucially, the objective is not to minimize prediction error, nor to achieve theoretical ideals, such as consistency approaching the offline optimum or robustness matching the best classical online algorithm. Instead, the true goal is to *maximize system-level utility*, measured in end-to-end latency, throughput, energy efficiency, tail performance, etc. A learning-augmented algorithm that reduces cache misses by 20% but inflates tail latency by 50% may win on paper yet fail in practice. Success requires optimizing the entire pipeline, not just the algorithmic core.

3 Case Study: Deep Learning Recommendation Models

To ground the learning-augmented algorithmic framework in a real-world setting, we examine its application in Deep Learning Recommendation Models (DLRMs), a cornerstone of modern AI infrastructure powering search, advertising, and content recommendation systems. In these models, user

interactions are encoded as high-dimensional sparse features (e.g., user ID, item category), which are mapped to dense vectors via embedding tables. These tables, often spanning hundreds of gigabytes, far exceed the capacity of GPU high-bandwidth memory (HBM), necessitating hierarchical storage across CPU DRAM and SSD, with only a "hot" subset cached on-device.

This architecture introduces a critical performance bottleneck: *embedding lookups dominate inference latency.* Due to irregular and sparse access patterns, the PCIe bandwidth between CPU and GPU becomes a severe constraint. Traditional caching policies like Least Recently Used (LRU), used in frameworks such as HugeCTR [9] and Fleche [10], are ill-suited to this workload. While LRU assumes recency implies future reuse, real-world access patterns in DLRMs exhibit complex temporal and structural dependencies that deviate from this heuristic. As shown in Table 1, the gap between LRU and the optimal offline algorithm (Belady's MIN [2]) can exceed 20 percentage points on production datasets like Kwai-Item and Kwai-User, revealing substantial room for improvement.

Table 1. Cache Hit Rate Gap Between LRU and OPT on Real-World DLRM Datasets.

Dataset	**Cache Capacity**			
	1%	2%	5%	**Max Gap**
Kwai-Item (OPT)	42.05%	53.14%	68.63%	—
Kwai-Item (LRU)	18.93%	30.17%	50.40%	—
Gap	23.12%	22.97%	18.23%	**23.12%**
Kwai-User (OPT)	35.31%	42.28%	53.72%	—
Kwai-User (LRU)	17.93%	23.10%	32.94%	—
Gap	17.38%	19.18%	20.78%	**20.78%**

This motivates the use of learning-augmented caching, which leverages predictive models to anticipate future access patterns. However, naively integrating ML into eviction logic risks instability under inaccurate predictions, which is a fatal flaw in production systems. To address this, we present LAURA (Learning-Augmented Unified Replacement Architecture), a practical, robust, and efficient framework designed specifically for GPU caching in AI workloads. At its core, LAURA combines the efficiency of classical heuristics with the adaptability of machine learning. It employs a lightweight Gradient Boosting Machine (GBM) predictor trained online to estimate the next access times of cached embedding vectors. Unlike prior approaches that blindly follow predictions (e.g., BlindOracle [5]) or incur prohibitive overhead (e.g., F&R [8]), LAURA uses predictions to bias eviction decisions while maintaining a safety net: *when prediction errors are detected, it gracefully degrades to LRU.*

The algorithm perspective. The algorithm operates in phases, starting each time a requested item is not in the cache and the cache is full. At the beginning of a phase, a hyper-parameter λ is initialized as $\lambda = 1$ and all cached items are tracked as candidates for eviction. During the phase, if a requested item was previously evicted due to a prediction, the algorithm responds by falling back to LRU, i.e., evicting the least recently used item, and reduces λ with $\lambda \leftarrow \lambda/b$ ($b > 1$) to shrink the scope of future prediction-based decisions. Otherwise, it selects an item to evict from the top λk candidates favored by LRU (where k is the cache size), choosing the one predicted to be accessed farthest in the future. This approach limits the influence of predictions over time if they prove unreliable, while leveraging them when they work. The result is a robust hybrid policy: it achieves strong performance under accurate predictions and maintains $O(k)$-competitive guarantees even when predictions fail completely.

The system perspective. LAURA is designed for practical deployment: (i) The predictor and cache metadata reside in CPU DRAM, avoiding GPU memory contention. (ii) Eviction decisions are derived on the CPU, minimizing PCIe traffic. (iii) It integrates seamlessly with the production framework HugeCTR, acting as a modular execution layer. To support efficient GPU access, LAURA builds upon the *Slab-Hash* data structure used in HugeCTR, which partitions the cache into fixed-size buckets for warp-aligned memory access. While bucketing improves parallelism, it can lead to contention when popular items collide in the same bucket. LAURA mitigates this through lightweight frequency-aware placement and avoids lock contention via fine-grained synchronization.

In evaluation, LAURA significantly outperforms LRU on real-world DLRM workloads, improving hit rates by up to 5-23% on Kwai datasets and reducing end-to-end latency by 10-35%.

4 Conclusion

The growing complexity of AI workloads demands a rethinking of classical algorithms. As shown in LAURA, simply replacing heuristics like LRU with learned models risks instability under prediction errors. Instead, we advocate for a new paradigm: learning-augmented systems that enhance, rather than replace, proven mechanisms. The key insight is judicious augmentation: *using predictions to guide decisions while preserving robustness through fallbacks, bounded scope, and system-aware design.* LAURA's hybrid eviction policy, which combines LRU's stability with lightweight ML guidance, achieves significant performance gains without sacrificing reliability. This principle extends beyond caching: wherever algorithms face uncertainty, intelligence should bias decisions, not dictate them.

To make this vision widespread, we need modular architectures, online predictors with low overhead, and formal guarantees on worst-case behavior. The goal is not fully learned systems, but wise ones: adaptive when possible, conservative when necessary. As AI systems grow more autonomous, the challenge is not just to make them smarter, but to make them safe to be smart. The future of systems lies not in choosing between heuristics and learning, but in uniting them with purpose.

References

[1] Vamsi Addanki, Maciej Pacut, and Stefan Schmid. 2024. Credence: augmenting datacenter switch buffer sharing with ML predictions. In *Proceedings of the 21st USENIX Symposium on Networked Systems Design and Implementation* (Santa Clara, CA, USA) *(NSDI'24)*. USENIX Association, USA, Article 34, 22 pages.

[2] Laszlo A. Belady. 1966. A study of replacement algorithms for a virtual-storage computer. *IBM Systems journal* 5, 2 (1966), 78–101.

[3] Allan Borodin and Ran El-Yaniv. 2005. *Online computation and competitive analysis.* cambridge university press.

[4] Peng Chen, Hailiang Zhao, Jiaji Zhang, Xueyan Tang, Yixuan Wang, and Shuiguang Deng. 2025. Toward a Lightweight and Robust Design for Caching. arXiv:2507.16242 [cs.DS]

[5] Thodoris Lykouris and Sergei Vassilvitskii. 2021. Competitive caching with machine learned advice. *Journal of the ACM (JACM)* 68, 4 (2021), 1–25.

[6] Michael Mitzenmacher and Sergei Vassilvitskii. 2022. Algorithms with predictions. *Commun. ACM* 65, 7 (June 2022), 33–35. doi:10.1145/3528087

[7] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving online algorithms via ML predictions. *Advances in Neural Information Processing Systems* 31 (2018).

[8] Karim Ahmed Abdel Sadek and Marek Elias. 2024. Algorithms for Caching and MTS with reduced number of predictions. In *The Twelfth International Conference on Learning Representations.* https://openreview.net/forum?id=QuIiLSktO4

[9] Yingcan Wei, Matthias Langer, Fan Yu, Minseok Lee, Jie Liu, Ji Shi, and Zehuan Wang. 2022. A gpu-specialized inference parameter server for large-scale deep recommendation models. In *Proceedings of the 16th ACM Conference on Recommender Systems.* 408–419.

[10] Minhui Xie, Youyou Lu, Jiazhen Lin, Qing Wang, Jian Gao, Kai Ren, and Jiwu Shu. 2022. Fleche: an efficient GPU embedding cache for personalized recommendations. In *Proceedings of the Seventeenth European Conference on Computer Systems.* 402–416.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009