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 propose *learning-augmented systems*, a new paradigm that integrates machine learning as a lightweight, co-designed component within robust system architectures. Rather than replacing core logic, predictions act as hints to improve utility, with correctness and performance guarantees preserved through minimal interfaces, fallback mechanisms, and end-to-end observability.

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 Promise and Pitfalls of Machine Learning

Integrating machine learning (ML) into systems, often termed *ML for systems*, has demonstrated performance improvements across domains such as database tuning, job scheduling, and resource management. By replacing static heuristics with learned policies, these systems adapt to observed workloads, often outperforming traditional approaches under stable conditions. However, widespread adoption has been limited by practical challenges. Many ML-enhanced systems exhibit poor robustness: small shifts in workload distribution can lead to significant prediction errors, causing performance to degrade below that of simple heuristics. Moreover, the overheads introduced by ML, such as inference latency, memory footprint, and training cost, are frequently overlooked in evaluations, leading to solutions that perform well offline but falter in production.

To address these limitations, the framework of *learning-augmented algorithms* (also known as *algorithms with predictions*) has emerged [1]. This approach uses ML predictions, e.g., estimates of job sizes or request arrival patterns, as auxiliary inputs and combines them with classical algorithms that provide worst-case guarantees. The goal is to achieve three key properties: *consistency* (near-optimal performance when predictions are accurate), *robustness* (performance no worse than a baseline algorithm under arbitrary errors), and *smoothness* (graceful degradation as prediction quality declines). As illustrated in Figure 1, this framework offers a principled way to integrate

*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

predictions while bounding worst-case risk. Despite its theoretical appeal, a gap remains between this algorithmic model and real-world system design. The theory typically assumes that predictions are available at negligible cost and without delay. In practice, inference introduces latency, models become stale due to workload drift, and integrating ML components adds operational complexity related to state management, versioning, and failure recovery.

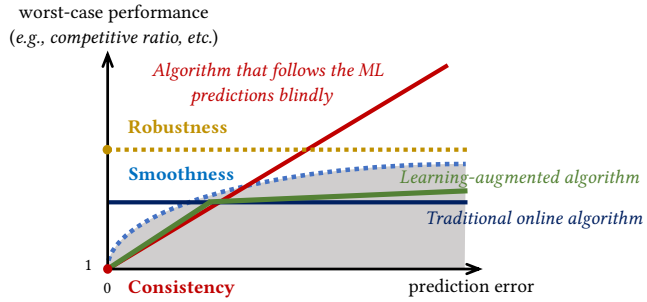

Fig. 1. Performance under varying prediction error. The learning-augmented algorithm achieves consistency (high performance when accurate), smoothness (gradual degradation), and robustness (bounded competitive ratio with respect to a classical online algorithm under arbitrary errors).

This disconnect motivates our focus on *learning-augmented systems*, a systems-first perspective that extends the algorithmic framework to the full system stack. We treat ML not as an external oracle, but as a co-designed component within robust system architectures. Predictions are used to *bias* decisions when reliable, but never to override core logic. The objective is to maximize end-to-end utility, including throughput, tail latency, energy efficiency, and reliability, under realistic constraints. Achieving this requires rethinking not just the algorithms, but the entire design philosophy of how intelligence is integrated into systems.

2 The Three Systemic Challenges

Translating the theoretical benefits of learning-augmented algorithms into real systems requires confronting three interrelated challenges: the resource cost of inference, the temporal mismatch between prediction and execution, and the operational complexity of integrating learned components. These are not merely engineering hurdles but reflect fundamental misalignments between algorithmic abstractions and system realities.

2.1 Inference Overhead: The Resource Cost of Predictions

Learning-augmented algorithms typically assume that predictions are available instantaneously and at no cost. In practice, each inference consumes computational resources and introduces latency that can become part of the critical path. For example, a web server using a learned model to guide cache eviction may achieve high accuracy offline, yet incur 15ms of inference latency per request. If this exceeds the latency saved by a cache hit, the net effect is negative: the predictor degrades overall performance despite its predictive accuracy.

This illustrates a central principle: the utility of a prediction depends not only on its accuracy but on its net resource impact. A less accurate but lightweight model may outperform a more accurate one if it reduces end-to-end overhead. Therefore, predictors must be designed with efficiency as a first-order concern, favoring small models, support for caching or precomputation, and mechanisms to amortize cost. Moreover, systems should monitor the marginal benefit of predictions and dynamically disable them when overhead exceeds gain. In this view, intelligence is

not a plug-in module but a *managed resource*, subject to cost-benefit analysis like any other system component.

> **Design Insight.** *Predictors should be lightweight, support output reuse, and be invoked only when expected utility justifies the cost.*

2.2 Temporal Mismatch: Drift, Delay, and the Limits of Predictability

Theoretical analyses often assume that predictions, while imperfect, reflect current system conditions. In reality, models are trained on historical data and updated periodically, leading to temporal mismatches due to workload drift and feedback delay. For instance, a content preloading system may rely on models trained from prior user behavior. When user interests shift abruptly due to external events or seasonal trends, the model continues making recommendations based on outdated patterns, resulting in wasted network bandwidth and device resources.

This challenge highlights the limitations of predictability in dynamic environments. While learning-augmented algorithms provide robustness guarantees under arbitrary errors, real systems face subtler degradation: gradual declines in prediction quality that may go undetected by standard monitoring. Furthermore, feedback loops for retraining are often delayed or incomplete, limiting adaptability. As a result, systems must go beyond passive error tolerance and actively estimate prediction confidence, detect staleness, and incorporate adaptive mechanisms such as conservative fallbacks or online updates.

> **Design Insight.** *Systems must gracefully handle both sudden shifts and slow degradation in prediction quality, using confidence estimation and proactive adaptation.*

2.3 Operational Complexity: State Management and Integration Overhead

Integrating ML components into production systems introduces new sources of statefulness, asynchrony, and failure modes absent in traditional heuristic-based designs. Consider a database buffer manager that uses a learned model for page eviction. If the model runs in a separate process and updates hourly, a failed update due to resource contention or communication loss may leave the system using stale weights without triggering any alert. This leads to suboptimal decisions and degraded performance, yet remains invisible to standard health checks.

This reflects the integration burden of learned components: unlike deterministic rules, ML predictors introduce dependencies on external processes, versioned models, and feature pipelines. They may produce invalid outputs, fail silently, or fall out of sync with system state. The interface between predictor and decision logic thus becomes a critical fault boundary. Without well-defined protocols for versioning, health checking, and fallback, the system risks silent failures that undermine reliability.

> **Design Insight.** *The prediction interface should be simple, synchronous where possible, and include fast, safe fallbacks to ensure resilience.*

3 Toward a Systems-First Design Philosophy

3.1 A System-First Perspective

The challenges outlined in Section 2 are not isolated engineering issues but manifestations of a broader design mismatch: learning-augmented systems often treat prediction as a substitute for decision logic, rather than a conditional input to it. When predictions are slow, stale, or incorrect, such systems exhibit degraded performance or silent failures because the core decision mechanism lacks independence. To address this, we advocate for a systems-first design philosophy that rethinks the integration of learned components. At its core is a *dual-path architecture*: a *robust decision core* that operates correctly under all conditions, augmented by a *lightweight prediction module*

that biases decisions toward higher utility when predictions are reliable. The core is not replaced, but informed. This separation ensures functional correctness even when the predictor fails or is disabled. This philosophy shifts the focus from optimizing prediction accuracy to co-designing the entire system around the operational realities of inference. It emphasizes *end-to-end utility*, treats uncertainty as a first-class concern, and elevates the prediction interface to a critical system boundary. From this perspective, we derive three design principles that guide the construction of efficient, resilient, and maintainable learning-augmented systems.

Principle. *Optimize for Total System Cost, Not Just Prediction Accuracy.*

In learning-augmented systems, the objective is to maximize end-to-end performance under resource constraints. A prediction contributes positively only if its performance benefit exceeds its total cost, including inference latency, memory footprint, and maintenance overhead. Therefore, predictors should be designed for computational efficiency, support output caching, and be conditionally invoked based on expected utility. Techniques such as model distillation, early exiting, or coarse-grained prediction (e.g., per session instead of per request) reduce resource consumption. The system should monitor the marginal gain of predictions and disable them when net utility becomes negative. In this framework, intelligence is not a standalone enhancement, but a *managed resource*, governed by cost-benefit trade-offs like CPU time or memory bandwidth.

Principle. *Design for Imperfect and Evolving Predictions.*

Predictions are inherently uncertain and temporally bounded. They reflect past data and may not generalize to current conditions. Rather than assuming correctness, systems should treat predictions as advisory signals that inform, but do not override, established decision logic. This principle entails four practices: (i) using predictions to bias decisions rather than determine them; (ii) incorporating confidence estimates or error bounds into control flow; (iii) enabling fast fallbacks to robust baseline policies when predictions are untrusted; and (iv) supporting online adaptation via incremental updates or fine-tuning on recent feedback. The goal is graceful degradation: performance may decline under poor predictions, but remains above a known lower bound, even under complete prediction failure.

Principle. *Treat the Prediction Interface as a Critical System Boundary.*

The prediction interface is a first-class system boundary. It must be designed with the same rigor as core APIs. An effective interface is: (i) *simple*, exchanging minimal, interpretable signals (e.g., a scalar or binary hint); (ii) *synchronous in the critical path*, to avoid staleness from asynchronous pipelines; (iii) *resilient*, with versioning, health checks, and fallbacks; and (iv) *observable*, exposing metrics such as latency, drift, and fallback frequency. Crucially, when the system feeds data to the predictor, e.g., features or training labels, this reverse flow must not interfere with the critical path. It should be asynchronous, sampled when needed, and isolated from core logic. This ensures that the cost of generating intelligence does not degrade system utility. Like power or logging, prediction support is a system service: essential, but never privileged.

This systems-first philosophy is illustrated in Figure 2, which emphasizes the architectural separation, the flow of predictions, and the role of the interface in ensuring reliability and efficiency.

3.2 Case Study: A Learning-Augmented Embedding Cache

We illustrate the principles through a learning-augmented cache for embedding tables in recommendation systems. Due to limited GPU memory, embeddings are cached in CPU memory, where hit rate and access latency directly affect end-to-end performance. While learned models can improve hit rates by predicting future accesses, naive integration risks overhead and fragility. Our design applies the three principles to balance efficiency and robustness.

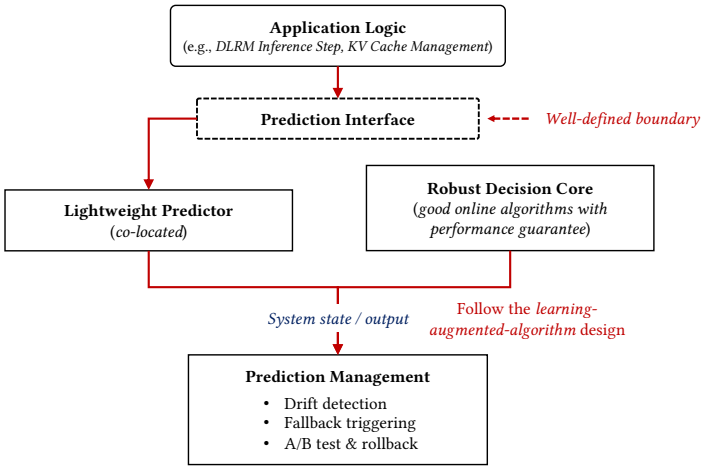

Fig. 2. Architecture of a learning-augmented system. The *robust decision core* (right) ensures correctness and handles the critical path. A *lightweight predictor* (left) runs co-located inference to generate advisory signals. The *prediction interface* (center) is a first-class boundary, designed for simplicity, synchrony, and fast fallback. Management components (bottom) monitor accuracy, detect drift, and support rollback. This design embodies three principles: optimize for total system cost, design for imperfect predictions, and treat the interface as a critical boundary.

To minimize *total system cost*, the predictor is a compact model co-located with cache metadata in CPU memory. Predictions are computed synchronously during eviction, avoiding cross-device communication. The model prioritizes low latency and small footprint over accuracy, and prediction is disabled when utility drops. To handle *imperfect predictions*, the system combines the learned policy with a robust baseline (e.g., LRU). A dynamic blending mechanism adjusts their relative weights based on observed prediction reliability. When errors increase, the system shifts toward the baseline, ensuring graceful degradation. The *prediction interface* is minimal: each cached item carries a scalar (e.g., predicted next access time), updated asynchronously but consumed synchronously. No raw features or complex payloads are exchanged, reducing coupling and enabling independent evolution.

4 Conclusion

Integrating ML into systems requires more than accurate models; it demands architectural co-design. Learning-augmented systems treat predictions as advisory inputs to a robust decision core, optimizing for end-to-end utility rather than accuracy. They tolerate prediction errors through fallbacks and graceful degradation, and treat the prediction interface as a critical, observable system boundary. This approach shifts the paradigm from *ML for Systems* to *Learning-Augmented Systems*, where intelligence is a managed resource, not a control plane. Future advances will come not from standalone intelligence, but from systems that intelligently manage uncertainty.

References

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

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