CORAL: An LLM-Native Harness for Production Recommender Systems

Muhammad Rafay Azhar✉
Meta Platforms, Inc.
New York, NY, USA
rafayazhar@meta.com

Yuhang Zhou✉
Meta Platforms, Inc.
New York, NY, USA
zyhang@meta.com

Gilbert Jiang
Meta Platforms, Inc.
Toronto, Ontario, Canada
gjiang@meta.com

Yuchen Wang
Meta Platforms, Inc.
Menlo Park, CA, USA
wang617@meta.com

Rahul Sharma
Meta Platforms, Inc.
New York, NY, USA
rahulsh@meta.com

Matthew DeSousa
Meta Platforms, Inc.
Toronto, Ontario, Canada
matthewdesousa@meta.com

Jiayi Liu
Meta Platforms, Inc.
Menlo Park, CA, USA
liujiayi@meta.com

Xin Guo
Meta Platforms, Inc.
Bellevue, WA, USA
xinguo@meta.com

Lizhu Zhang
Meta Platforms, Inc.
Menlo Park, CA, USA
lizhu@meta.com

Xiangjun Fan
Meta Platforms, Inc.
Menlo Park, CA, USA
maxfan@meta.com

Abstract

Production recommender systems shape what billions of people see, and sustaining their performance is a continual optimization problem: as content, user behavior, and upstream models shift, the choices that govern how these systems retrieve, rank, and serve content must be revisited to keep them near their best operating point. This work has traditionally fallen to human engineers testing changes through online experiments—a slow, reactive process bounded by engineering effort rather than the size of the opportunity, so parts of the system go unrevised and drift as conditions change. Although large language models have been applied to ranking, user modeling, and offline model development, few systems place an agent in a continual, closed loop that acts on a live recommender and learns from the measured consequences of its own decisions. We present CORAL (Constraint-Optimized Recommender via an Agentic Loop), an LLM-native harness that closes such a loop: each cycle, the agent observes operating signals, reasons over a memory of its past decisions and their measured effects, and invokes tools—including a numerical optimizer that keeps every change within a fixed operating budget—to reconfigure the live recommender, after which the measured outcome informs the next cycle. We formulate this as a partially observed, non-stationary, constrained optimization problem in which the policy improves in context, without parameter updates, from the effects of its own prior actions. Across two large-scale social platforms, evaluated with A/B experiments, the same harness improves engagement at no additional serving cost on one and delivers substantial efficiency savings without degrading engagement on the other, together spanning the engagement–efficiency frontier. Its decisions improve as the loop iterates, indicating that a single agentic loop can take on continual optimization work traditionally performed by human algorithm engineers, with a path from human-supervised operation toward autonomous operation under guardrails.

CCS Concepts

• **Computing methodologies** → **Machine learning**; **Machine learning**; • **Applied computing** → **Law, social and behavioral sciences**.

Keywords

Large Language Models, Recommender Systems, User Modeling

1 Introduction

Recommender systems mediate how billions of users discover content and products across social media, e-commerce, and entertainment. Over the past decade they have advanced from collaborative filtering to deep learning ranking models [18] and, more recently, to large sequential and generative architectures [11, 37]. A modern industrial recommender is not a single model but a large, multi-stage system—candidate retrieval, ranking, and serving—and, at every stage, its pipelines and models are governed by a broad collection of parameters and policies: retrieval budgets, ranking weights, serving and caching policies, and per-segment treatments, among others.

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Improving such a system is therefore a continual, iterative endeavor: practitioners form hypotheses, implement changes, evaluate them with online experiments, and act on the results, repeating this cycle to advance the system. Effective as it is, this process is bounded by human effort—progress scales with the number of engineers and experiments rather than with the size of the opportunity.

Many of these choices are made by engineers rather than learned end-to-end with the models, and need not remain optimal as the system evolves. The human-driven iteration used to refine them has structural limitations. Each experiment probes only a small region of a vast design space, so improvements are slow and conservative. Because changes are usually prompted by observed regressions or opportunities, the process is reactive rather than anticipatory. And because each cycle is time-intensive and computationally costly, parts of the system are revisited infrequently and largely held fixed between updates—even as content, user behavior, and upstream models continue to shift—so a system can drift away from its own best operating point between interventions. These effects are most pronounced for low-signal and new users, whose behavior is underrepresented in the aggregate metrics that guide most decisions, and they are compounded by operating constraints—serving capacity and compute budgets—that couple decisions across the system.

Large language models (LLMs) offer a way to ease this human bottleneck. Beyond ranking items directly [8, 40], recent work equips LLMs with memory and tools to reason about users and content [3, 24, 26] and to automate portions of the model-development and system-optimization process [9, 12, 17]. Together, these results suggest that LLM agents can take on work that has traditionally required human algorithm engineers. Most such systems, however, act on the model, the user representation, or the offline development pipeline; comparatively few place an agent in a continual, closed loop that acts on a live production system and learns from the measured consequences of its own decisions.

We present CORAL (Constraint-Optimized Recommender via an Agentic Loop), a flexible, LLM-native agentic harness that closes such a loop around a recommender. In each cycle, the agent observes the system's operating signals, reasons over them together with a memory of its own past decisions and their measured effects, and invokes tools—including numerical optimization—to produce a change; once that change has been deployed and measured, the outcome informs the next cycle. In this way, CORAL continually and autonomously drives improvements that would otherwise demand repeated manual iteration, refining its own decisions over time. We instantiate the harness on a high-value class of decisions: the continuous, constraint-aware allocation of a recommender's resources across its components, where the agent reallocates bounded budgets to improve engagement while respecting operating constraints—treating efficiency as a first-class objective alongside engagement.

We evaluate CORAL on two large-scale social platform surfaces with different components and decision types, using A/B experiments. On one, it improves engagement—including for low-signal and new users—at no additional serving cost; on the other, it delivers substantial efficiency savings without degrading engagement. The same harness applies to both surfaces, and its decisions improve as the loop iterates.

Our contributions are as follows:

- We formulate the continual, agent-driven optimization of a recommender as a partially observed, non-stationary, constrained problem, in which an LLM policy refines its decisions from the measured effects of its own prior actions.
- We present the CORAL harness—the context, tools, and closed loop that turn a general-purpose LLM into a continual optimizer of a live recommender—and show that it generalizes across surfaces and decision types rather than being tied to a single lever.
- We report two large-scale deployments, evaluated with A/B experiments, that together span the engagement–efficiency frontier, including gains for low-signal and new users.
- We distill practical lessons from operating such a loop, including the path from human-supervised operation toward increasingly autonomous operation under guardrails.

2 Related Work

Our work lies at the intersection of research on tool use and memory in agents and on LLM-based recommender agents [8, 40, 43].

2.1 Tool Use and Memory in LLM Agents

A broad line of research augments LLM agents with external tools, enabling them to retrieve information, invoke specialized models and APIs [25, 31, 44], execute code [35], and interact with dynamic environments [41]. Rather than relying solely on parametric knowledge, these agents interleave reasoning, tool selection, execution, and observation to solve tasks requiring external information or specialized capabilities [6]. Subsequent work has advanced tool discovery, multi-tool planning, API-call generation, and generalization to previously unseen tools [7, 32]. Nevertheless, reliable tool selection, accurate argument construction, and robust recovery from execution failures remain persistent challenges.

Another line of research equips LLM agents with memory mechanisms that preserve information across interactions and support learning from accumulated experience [4, 15, 22, 23]. Existing systems store and retrieve conversational histories, environmental observations, successful strategies, failures, and self-reflections to inform future planning and decision-making [36, 43]. More advanced approaches organize memories into episodic, semantic, reflective, or hierarchical structures and incorporate mechanisms for summarization, consolidation, updating, and selective forgetting [28, 33]. Despite this progress, fundamental questions remain regarding what information should be retained, how memories should evolve as new evidence arrives, and when outdated, redundant, or conflicting information should be revised or removed.

2.2 LLM Agents in Recommender Systems

One line of work uses LLM agents to simulate user behavior for system evaluation and behavioral analysis, contributing primarily to evaluation methodology rather than directly improving recommendation quality [1, 2, 27, 29, 38, 42]. A second line develops autonomous agents that reason, plan, and invoke conventional recommender models as tools, but typically handles each interaction independently without maintaining persistent user memory [5, 10, 13, 21, 30, 41]. Another line equips recommender agents

with persistent memory through iterative profile refinement, structured memory organization, and collaborative preference propagation [3, 14, 16, 19, 34, 39]. However, these approaches generally rely on flat memory representations and lack a complete lifecycle governing preference extraction, consolidation, updating, and forgetting.

A more recent line of research shifts the role of agents from generating recommendations to improving recommender systems themselves. These approaches use agents to search over models, code, and system configurations, thereby automating parts of system development and optimization [9, 12, 17, 26]. In most cases, however, the agent operates on the recommendation model, user representation, or offline development pipeline. Comparatively little attention has been given to agents that continually intervene in a deployed recommender system, observe the measurable effects of their actions, and adapt subsequent decisions based on this feedback. Our work addresses this gap by placing the agent in a continual, closed-loop interaction with a recommender system.

3 Problem Formulation

We consider a recommender system whose behavior is governed by a set of tunable control parameters, and we study an agent that updates these parameters over time to improve the system.

Let s denote a configuration of these parameters. Its effect depends on the surrounding operating context—the users and content the recommender serves and the upstream models it uses—which the agent cannot observe directly and which changes over time. We capture this dependence by indexing the system's response by the cycle t: under configuration s at cycle t, the recommender attains a business objective $J_t(s)$, which we take to be engagement, and incurs an operating cost $c_t(s)$ that must not exceed a fixed budget B. Both the objective and the operating constraint are set by the operator rather than by the method; the formulation assumes nothing particular about either, and any objective that an operator wishes to improve under a bounded resource constraint fits the same form. It therefore covers not only maximizing the objective under a cost budget but also its dual—minimizing cost while holding the objective at a target level—and an operator may pose either; our two deployments take one form each.

We group the tunable surface into N control units, indexed $i = 1, \ldots, N$; each unit is a component whose setting $s_{t,i}$ the agent controls, drawn from a feasible set—a bounded continuous value or a choice from a discrete set. For example, a control unit might be a retrieval source whose setting is its share of a fixed candidate budget, or a user segment whose setting is one treatment chosen from a discrete serving menu. In each decision cycle t, the agent selects a configuration $s_t = (s_{t,1}, \ldots, s_{t,N})$.

The best configuration at cycle t is

$$s_t^* = \arg\max_s \; J_t(s) \quad \text{subject to} \quad c_t(s) \le B. \tag{1}$$

Here s_t^* is the best feasible configuration at cycle t (the oracle optimum), while s_t is the configuration chosen by the agent. Because the operating context evolves, s_t^* is not fixed: it moves from cycle to cycle, and a configuration left unchanged grows increasingly suboptimal. The agent's aim is therefore not to converge to a single optimum but to keep the deployed configuration close to this moving target while respecting the budget. Across successive cycles, we state this as minimizing the cumulative shortfall relative to the best feasible configuration,

$$\min_\pi \; \sum_t \left[J_t(s_t^*) - J_t(s_t) \right] \quad \text{subject to} \quad c_t(s_t) \le B \;\; \forall t, \tag{2}$$

where the configurations s_t are produced by the agent's policy π.

The policy π is an LLM that maps the current observation and memory to the next configuration, $s_t = \pi(o_t, M_t)$. The agent observes the system only through aggregate signals. At each cycle it receives an observation o_t that summarizes per-unit operating statistics over the preceding window, together with their change from the previous window. It also maintains a memory M_t over the most recent m cycles: the observations it has seen, the configurations it has chosen, and the outcomes it has attributed to them. The policy reaches a configuration through a short sequence of actions rather than a single step: it analyzes the current statistics, retrieves relevant history from memory, estimates the effect of its previous configuration, and invokes a numerical optimizer that projects a candidate configuration onto the budget-feasible set, so that every emitted configuration satisfies $c_t(s_t) \le B$ by construction.

4 The CORAL Harness

The previous section formalizes what the agent must do; here we describe the system that lets a general-purpose language model do it reliably. We call this system the *harness*. It surrounds the language model with three elements the model alone does not provide: the *context* needed to understand the current state of the recommender, a set of *tools* with which to analyze that state and act on it, and a *loop* that runs on a fixed cadence and carries experience from one cycle to the next. The model contributes reasoning; the harness makes that reasoning grounded, budget-feasible, and cumulative. Figure 1 shows the overall design.

This problem calls for a policy of an unusual kind. The decision at each cycle rests on many heterogeneous, partly qualitative per-unit signals—raw metrics, funnel behavior, and how each unit is trending—weighed together with domain knowledge about what those signals imply, and it resists a fixed rule or a single tunable formula. The right allocation also drifts as the operating context changes, so the policy must reinterpret the current signals each cycle rather than settle on a static mapping. An LLM is well suited to both: it reasons over heterogeneous evidence and prior knowledge to decide where to reallocate, adapts in context as new outcomes arrive without retraining, and articulates a rationale for each change—the property the loop relies on for human oversight and self-attribution. What it cannot guarantee on its own—a hard budget and consistent, well-formed decisions—the harness supplies through the optimizer and guardrails.

4.1 Memory

The harness maintains a persistent memory that spans the most recent m cycles and gives the model the context to reason about both the system and its own past behavior. We organize it into three stores. An *observation* store records the raw observations o_t—the per-unit operating statistics and their recent changes—so the model can see the current state of each part of the system. An *assessment* store holds the model's own synthesized assessments from earlier

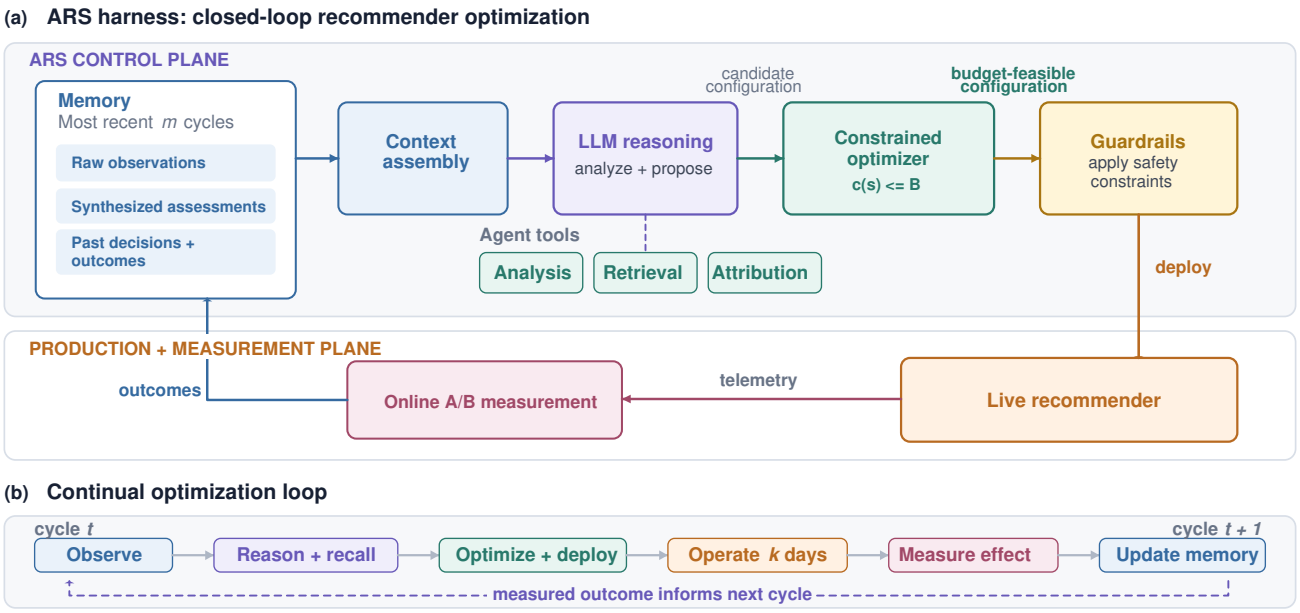

Figure 1: Overview of the CORAL harness. (a) The control plane combines persistent memory and LLM reasoning with deterministic tools, constrained optimization, and guardrails. A validated, budget-feasible configuration is applied to the live recommender; telemetry and online A/B results are measured, attributed, and returned to memory. (b) On each k-day cycle, the measured effect of the deployed configuration becomes context for the next decision, allowing the policy to adapt in context.

cycles: concise, natural-language judgments of which parts of the system are performing well or poorly and how their behavior is trending. Finally, a *decision* store records the configurations the agent has previously deployed together with the outcomes later attributed to them. Together, these stores let the model ground each decision not only in the present state of the system but in the consequences of what it has already tried.

4.2 Tools

A language model cannot, on its own, guarantee that its proposals are numerically sound or that they respect a hard budget, so the harness equips it with a small set of tools that it invokes while reasoning. An analysis tool computes and summarizes statistics and their changes from the raw observations. A retrieval tool surfaces the relevant history—past configurations, assessments, and outcomes—from memory. An attribution tool estimates the effect of the agent's previous configuration, as described below. Most important for reliability is a constrained optimizer: given the bounded per-unit adjustments proposed by the model, it returns the closest configuration that satisfies the operating budget—the projection of the proposal onto the budget-feasible set $\{s : c_t(s) \le B\}$. When the proposal already respects the budget, this projection returns it unchanged; it binds only when the proposal would overspend, redistributing across units so that the deployed configuration provably satisfies $c_t(s_t) \le B$. A final tool applies the accepted configuration to the recommender's control surface. This division of labor lets the model do what it is best at—weighing many heterogeneous signals and articulating why a change should help—while delegating numerical feasibility to a component that can guarantee it. Each tool is deterministic and returns a structured result the model reasons over, so the model contributes judgment while the tools supply computation and hard guarantees.

4.3 The Optimization Loop

The harness runs these components as a closed loop on a fixed cadence of k days, invoking them in the same fixed order every cycle rather than leaving control flow to the model. It assembles the current observations and the relevant memory into the model's context; the model then analyzes the state, recalls what it has tried, estimates the effect of its last configuration, and proposes a new one, which the optimizer renders budget-feasible before it is applied. Once deployed, the configuration remains in effect until the next cycle, and its effect is measured with an A/B experiment whose result is written back to memory—closing the loop and becoming part of the context for the following cycle.

In our deployments we set the cadence to $k = 3$ days—long enough for a configuration's effect to surface in the metrics, yet short enough for the agent to act promptly on what it observes—and the memory horizon to $m = 3$ cycles, which keeps enough recent outcomes for the agent to learn from without carrying older results that the drifting operating context may have made less relevant. We chose these as sensible defaults rather than tuning them; the best cadence may also vary with seasonality, and systematically selecting k and m is a direction we are actively exploring.

This feedback is what separates the loop from a system that is merely re-run on a schedule. Because each configuration persists for a full cycle, the agent can attribute observed changes to its own most recent decision, comparing the period before the change with the period after and, where an A/B experiment is available, obtaining a measured treatment effect. Recording these attributed outcomes lets the agent reinforce the changes that helped and reverse those that did not, so its decisions improve over successive cycles. This improvement requires no retraining: the agent adapts entirely in context, through the observations, assessments, and outcomes accumulated in memory.

The loop runs autonomously, applying each cycle's configuration to the live recommender directly. A human supervises its operation, tracking the effect of its changes and the decisions it makes; as confidence in the agent grows, this supervision is progressively replaced by the harness's automated guardrails—feasibility checks, bounded-change limits, and safety constraints.

5 Case Studies

We evaluate CORAL on two large-scale social platforms, each instantiating the harness described above on a different control problem and a different service, and each evaluated with A/B experiments. The first targets engagement and shows how the loop improves across cycles and benefits low-signal users; the second targets serving efficiency and shows that the same harness transfers to a very different decision.

5.1 Allocating Retrieval Budget Across Candidate Sources

Our first deployment is a video-recommendation service that assembles each user's candidates from a set of complementary retrieval sources. Each source is allotted a share of a fixed retrieval budget, determining how many candidates it may contribute. These shares are typically hand-set and seldom revised, yet the best allocation shifts as content and behavior change, and a source that is efficient for one population can be wasteful for another.

In our formulation, the control units are the retrieval sources; a configuration assigns each source a budget multiplier within a bounded range, and the operating cost is the total retrieval budget consumed. Each cycle, the agent observes per-source signals—how many items a source contributes, how well those items convert into engaged views, and how far they survive the downstream funnel—and proposes a reallocation, trimming budget from sources whose candidates convert poorly and redirecting it to sources that deliver engaged views efficiently, all within the total budget.

The agent operates as an autonomous closed loop under human oversight, and its policy improved over three successive rounds of the loop, each evaluated with an A/B experiment (Table 1). A first, zero-shot proposal, formed from a single window of statistics, produced a small watch-time gain but no significant change in sessions (sessions are individual user app visits containing at least one video view). In the second round the agent shifted budget more aggressively between sources but overcorrected, and its measured effect was neutral. Incorporating that outcome, the agent refined the allocation over several further cycles and arrived at the deployed configuration reported below, which improved watch time further and produced significant session gains. This progression is not monotonic—the second round did not improve on the first—yet it is precisely the behavior the loop is designed to produce: rather than a gain in every round, the agent reacts to the measured effect of its own decisions and, over successive rounds, converges on a better configuration.

For the converged global allocation, the A/B experiment, spanning millions of users, showed a 0.16% increase in video-viewing sessions across all users, alongside a 0.15% increase in total watch time—all at no additional serving cost, as the reallocation consolidated the retrieval budget rather than expanding it.

We then extended the agent beyond a single global allocation to produce distinct allocations for different user segments, defined by engagement level and account tenure—for example, highly active users, low-signal users, and newly joined users. This per-segment control matters most for low-signal and new users, whom an allocation tuned to highly active users tends to underserve. For this cohort, whose historical engagement is sparse, the agent shifted budget toward retrieval sources that draw on content and current-context signals—which stay reliable when per-user data is scarce—and away from sources that rely on rich user histories. This raised video-viewing sessions for new low-signal users by 0.23%. The result indicates that the same harness can specialize its policy for segments that a single global allocation leaves behind, directly addressing the systematic under-service of low-signal users.

5.2 Allocating Serving Capacity Across User Segments

Our second experiment is on a different service, where the agent allocates serving capacity across user segments. For each segment, the service can select a treatment from a menu that ranges from lightweight to compute-intensive, varying how aggressively it retrieves and ranks, how much it caches, and how much it prefetches. More intensive treatments can raise engagement but cost more to serve, and the total serving cost is capped by a fixed budget—so choosing a treatment per segment is precisely the constrained allocation our formulation describes.

Here the control units are the user segments; a configuration assigns each segment a treatment from the discrete menu, and the operating cost is the compute required to serve it, bounded by the budget. Each cycle, the agent observes per-segment engagement and cost and decides where to spend more and where to spend less—raising the treatment for segments where added compute yields the most engagement and lowering it for segments where it yields little, so that the reclaimed budget funds the increases.

Because a treatment assignment stays in effect for a full cycle, the agent can compare a segment's engagement and cost before and after its previous decision, attribute the change to that decision, and use this to decide whether to continue in that direction or reverse course in the next cycle.

This behavior is evident in our experiments. In an A/B test involving millions of users, the agent improved its result over two successive rounds. In the first, working within a subset of user segments, the agent reduced serving cost substantially, saving millions of USD in annualized capacity expenditure. Reading that result against its previous decision, the agent recognized that the same

Table 1: Effect of CORAL's retrieval-budget policy on a large-scale video service across three successive rounds (R1–R3) of the loop, each evaluated with an A/B experiment. R1 is a zero-shot proposal; R3 aggregates several decision cycles and is the deployed configuration. Sessions are video-viewing sessions; "neutral" denotes no statistically significant change.

	R1	R2	R3
Watch time	+0.13%	neutral	+0.15%
Sessions (all users)	neutral	neutral	+0.16%
Sessions (largest market)	neutral	neutral	+0.77%

Table 2: Per-call cost parameters for the two deployments (estimated from prompt and payload sizes; the pipelines do not log token usage).

Case study	Calls per cycle	Input tokens/call	Output tokens/call
Retrieval-budget	∼10	∼1,500	∼2,500
Serving-capacity	∼ 8	∼2,000	∼2,500

change could be applied safely to additional user segments, and in the second round it widened its allocation to include them, increasing the savings of the first round by 44% while leaving engagement statistically unchanged—and thereby freeing capacity that can be reinvested elsewhere. This deployment exercises the efficiency side of the objective: it avoids degrading engagement while directing the operating budget where it is most productive.

5.3 Discussion

Across the two studies, the same harness addressed markedly different control problems—a continuous reallocation of retrieval budget across sources and a discrete assignment of serving treatments across segments—and improved the system along complementary axes: engagement, including for low-signal users, in the first, and serving efficiency without degrading engagement in the second. Together they trace the engagement–efficiency frontier that a production recommender must manage, and they show that a single, flexible harness generalizes across different services and decision types. These gains came from a process that required no per-decision engineering effort. Producing such allocations by hand is a heavy, periodic undertaking: an engineer forms a hypothesis, runs an experiment, and revises a single lever over the course of weeks, with the effort growing as the number of levers and segments grows. The harness instead adjusts every control unit on a short, fixed cadence, autonomously and at negligible compute cost—compressing a tuning cycle from several engineer-weeks to a few autonomous days, an order-of-magnitude faster turnaround with no engineer in the loop.

6 Computational Cost

Because the harness acts on the control surface rather than on individual requests, its language-model cost is charged per decision cycle, not per user. Over a deployment spanning T days at a cadence of k days, the total inference cost is

$$\text{Cost} = \underbrace{(T/k)}_{\text{cycles}} \cdot C \cdot \left(\tau_{\text{in}}\, p_{\text{in}} + \tau_{\text{out}}\, p_{\text{out}}\right), \tag{3}$$

where C is the number of LLM calls per cycle, τ_{in} and τ_{out} are the average input and output tokens per call, and $p_{\text{in}}, p_{\text{out}}$ are the per-token prices. Three properties follow. First, cost is inversely proportional to the cadence: a shorter k improves responsiveness at proportionally higher cost. Second, it is bounded per cycle, since τ_{out} cannot exceed the model's output-token limit. Third—and most consequential at scale—C is fixed by how the control surface is partitioned into decision groups (a handful per cycle), not by the number of users or requests served, so the cost is independent of traffic and remains negligible even on billion-user surfaces. This is in stark contrast to per-item or per-user LLM inference, where cost grows with traffic.

Table 2 lists per-call parameters for the two deployments. Each cycle issues only a handful of calls of a few thousand tokens each, so over a deployment of several cycles the total is on the order of 10^6 tokens. Using representative frontier-LLM pricing as a reference, this places the end-to-end inference cost of each deployment on the order of tens of U.S. dollars—negligible against the engagement gains and operating-cost savings it produces.

7 Conclusion

We presented CORAL, an LLM-native harness that places a language model in a continual, closed loop around a recommender. Rather than optimizing a model offline or serving recommendations directly, the agent acts on the recommender's live control surface: each cycle it observes the system's operating signals, reasons over them together with a memory of its past decisions and their measured effects, applies a change through tools that keep it budget-feasible, and learns from the measured outcome. We formulated this as a partially observed, non-stationary, constrained optimization problem, and showed that an LLM policy can address it by refining its decisions in context, without retraining. Across two large-scale social platforms, evaluated with A/B experiments, the same harness improved both engagement—including for low-signal and new users—and serving efficiency, providing evidence that a single agentic loop can carry out the continual optimization work that has traditionally fallen to human algorithm engineers.

Our approach has limitations, several of which point to future work. First, although the loop runs autonomously, it still operates under human supervision; strengthening the harness's automated guardrails so that this supervision can be reduced is a natural next step. Second, both studies instantiate the harness on a single class of decision—the constrained allocation of a recommender's resources across its components—and extending the same loop to qualitatively different levers, such as the retrieval and ranking logic itself, would further test the generality it is designed for. Finally, our evidence comes from A/B experiments that measure real effects but are costly to run and specific to their setting; a standardized way to evaluate agent-driven system optimization before deployment remains an open problem, and one we hope this work helps motivate.

Ethics and Privacy Statement

CORAL operates on a recommender's aggregate control parameters, not on any individual user's data: at each cycle it reads summary statistics for each control unit and writes back a configuration, collecting and processing no personal information. The harness does not choose the objective it pursues; it optimizes whatever objective, and respects whatever constraints, the operator supplies. Its effect on users therefore depends on that choice, and the responsibility for setting an objective and guardrails that serve users well remains with the operator. The harness is designed to keep that oversight practical: it runs under human supervision, holds every change within an explicit, budget-feasible set, and records a rationale for each decision that a person can review. These properties also limit the risk of operating the loop autonomously, and we do not see the method introducing societal risks beyond those already governed by the review that applies to any change to the systems it optimizes. We also utilize GPT-based models [20] solely for language polishing and proofreading.

References

[1] Nicolas Bougie and Narimasa Watanabe. 2025. SimUSER: Simulating User Behavior with Large Language Models for Recommender System Evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*.

[2] Luyu Chen, Quanyu Dai, Zeyu Zhang, Xueyang Feng, Mingyu Zhang, Pengcheng Tang, Xu Chen, Yue Zhu, and Zhenhua Dong. 2025. RecUserSim: A Realistic and Diverse User Simulator for Evaluating Conversational Recommender Systems. In *Proceedings of the ACM Web Conference 2025, Industry Track*.

[3] Weixin Chen, Yuhan Zhao, Jingyuan Huang, Zihe Ye, Clark Mingxuan Ju, Tong Zhao, Neil Shah, Li Chen, and Yongfeng Zhang. 2026. MemRec: Collaborative Memory-Augmented Agentic Recommender System. *arXiv preprint arXiv:2601.08816* (2026).

[4] Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. 2025. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory. *arXiv preprint arXiv:2504.19413* (2025).

[5] Yu Deng, Jianxun Lian, Yuxuan Lei, Chongming Gao, Kexin Huang, and Jiawei Chen. 2025. RecBot: Agent-Based Recommendation System. *arXiv preprint arXiv:2509.21317* (2025).

[6] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536* (2025).

[7] Jie He, Jennifer Neville, Mengting Wan, Longqi Yang, Hui Liu, Xiaofeng Xu, Xia Song, Jeff Z Pan, and Pei Zhou. 2025. GenTool: Enhancing Tool Generalization in Language Models through Zero-to-One and Weak-to-Strong Simulation. In *Findings of the Association for Computational Linguistics: ACL 2025*. 1097–1122.

[8] Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2024. Large Language Models are Zero-Shot Rankers for Recommender Systems. In *Proceedings of the 46th European Conference on Information Retrieval (ECIR)*.

[9] Jinxin Hu, Hao Deng, Lingyu Mu, Hao Zhang, Shizhun Wang, Yu Zhang, and Xiaoyi Zeng. 2026. Rethinking Recommendation Paradigms: From Pipelines to Agentic Recommender Systems. arXiv:2603.26100 [cs.IR] https://arxiv.org/abs/2603.26100

[10] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. 2023. Recommender AI Agent: Integrating Large Language Models for Interactive Recommendations. *arXiv preprint arXiv:2308.16505* (2023).

[11] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*.

[12] Changxin Lao, Fei Pan, Guozhuang Ma, Han Li, Huihuang Lin, Jijun Shi, Kangzhi Zhao, Kun Gai, Mo Zhou, Qinqin Zhou, Quan Chen, Ruochen Yang, Shifu Bie, Shijie Yi, Shuang Yang, Shuo Yang, Wenhao Li, Wentao Xie, Xiao Lv, Xuming Wang, Yijun Wang, Yiming Chen, Yusheng Huang, Zhongyuan Wang, Zibo Zhao, Zijie Zhuang, Baoning Xia, Chao Liu, Chaoyi Ma, Chubo He, Dawei Cong, Feng Jiang, Gang Wang, Guilin Xia, Hanwen Xu, Jiahong Xie, Jiahui Qiao, Jian Liang, Jiangfan Yue, Jing Wang, Jinghan Yang, Jinghui Jia, Kan Qin, Lei Wang, Ming Li, Peilin Song, Pengbo Xu, Qiang Luo, Ruiming Tang, Shiyang Liu, Shuxian Jin, Tao Wang, Tao Zhang, Xiang Gao, Xianghan Li, Yingsong Luo, Yiwen Ning, Yongcheng Liu, Yueyang Liu, Yuan Guo, Zhaojie Liu, and Zhenkai Cui. 2026. AgentX: Towards Agent-Driven Self-Iteration of Industrial Recommender Systems. arXiv:2606.26859 [cs.AI] https://arxiv.org/abs/2606.26859

[13] Zhefan Lei, Hengxu Wang, Jiawei Zhang, and Shuai Chen. 2024. MACRec: A Multi-Agent Collaboration Framework for Recommendation. *arXiv preprint arXiv:2402.15235* (2024).

[14] Bingqian Li, Xiaolei Wang, Junyi Li, Weitao Li, Long Zhang, Sheng Chen, Wayne Xin Zhao, and Ji-Rong Wen. 2026. RecNet: Self-Evolving Preference Propagation for Agentic Recommender Systems. *arXiv preprint arXiv:2601.21609* (2026).

[15] Zhiyu Li, Chenyang Xi, Chunyu Li, Ding Chen, Boyu Chen, Shichao Song, Simin Niu, Hanyu Wang, Jiawei Yang, Chen Tang, et al. 2025. MemOS: A Memory OS for AI System. *arXiv preprint arXiv:2507.03724* (2025).

[16] Yuxin Liao, Le Wu, Min Hou, Yu Wang, Han Wu, and Meng Wang. 2026. From Atom to Community: Structured and Evolving Agent Memory for User Behavior Modeling. *arXiv preprint arXiv:2601.16872* (2026).

[17] Shaohua Liu, Liang Fang, Yilong Sun, Shudong Huang, Qingsong Luo, Shaoxin Liu, Xiaoyang Chen, Dongqiang Liu, Chuangang Ma, Zhenzhen Chai, Henghuan Wang, Shijie Quan, Changyuan Cui, Zhangbin Zhu, Peng Chen, Wei Xu, Lei Xiao, Haijie Gu, and Jie Jiang. 2026. NOVA: A Verification-Aware Agent Harness for Architecture Evolution in Industrial Recommender Systems. arXiv:2606.27243 [cs.IR] https://arxiv.org/abs/2606.27243

[18] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G. Azzolini, Dmytro Dzhulgakov, Andrey Mallevich, Ilia Cherniavskii, Yinghai Lu, Raghuraman Krishnamoorthi, Ansha Yu, Volodymyr Kondratenko, Stephanie Pereira, Xianjie Chen, Wenlin Chen, Vijay Rao, Bill Jia, Liang Xiong, and Misha Smelyanskiy. 2019. Deep Learning Recommendation Model for Personalization and Recommendation Systems. arXiv:1906.00091 [cs.IR] https://arxiv.org/abs/1906.00091

[19] Minh-Duc Nguyen, Hai-Dang Kieu, and Dung D. Le. 2026. AMEM4Rec: Leveraging Cross-User Similarity for Memory Evolution in Agentic LLM Recommenders. *arXiv preprint arXiv:2602.08837* (2026).

[20] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[21] Kesha Ou, Chenghao Wu, Xiaolei Wang, Bowen Zheng, Wayne Xin Zhao, Weitao Li, Long Zhang, Sheng Chen, and Ji-Rong Wen. 2026. Deep Research for Recommender Systems. *arXiv preprint arXiv:2603.07605* (2026).

[22] Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2023. MemGPT: Towards LLMs as Operating Systems. *arXiv preprint arXiv:2310.08560* (2023).

[23] Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th annual acm symposium on user interface software and technology*. 1–22.

[24] Qiyao Peng, Hongtao Liu, Hua Huang, Qing Yang, and Minglai Shao. 2025. A Survey on LLM-powered Agents for Recommender Systems. arXiv:2502.10050 [cs.IR] https://arxiv.org/abs/2502.10050

[25] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, Vol. 2024. 9695–9717.

[26] Xiang Shen, Yuhang Zhou, Yifan Wu, Zhuokai Zhao, Siyu Lin, Lei Huang, Qianqian Zhong, Lizhu Zhang, Benyu Zhang, Xiangjun Fan, and Hong Yan. 2026. Agentic Recommender System with Hierarchical Belief-State Memory. arXiv:2605.14401 [cs.CL] https://arxiv.org/abs/2605.14401

[27] Yunxiao Shi, Wujiang Xu, Zeqi Zhang, Xing Zi, Qiang Wu, and Min Xu. 2025. PersonaX: A Recommendation Agent Oriented User Modeling Framework for Long Behavior Sequence. In *Findings of the Association for Computational Linguistics (ACL)*.

[28] Theodore Sumers, Shunyu Yao, Karthik R Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *Transactions on Machine*

Learning Research (2023).
[29] Lei Wang, Jingsen Zhang, Hao Yang, Zhiyuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, Jun Xu, Zhicheng Dou, Jun Wang, and Ji-Rong Wen. 2023. User Behavior Simulation with Large Language Model based Agents. *arXiv preprint arXiv:2306.02552* (2023).
[30] Yancheng Wang, Ziyan Jiang, Zheng Chen, Fan Yang, Yingxue Zhou, Eunah Cho, Xing Fan, Yanbin Lu, Xiaojiang Huang, and Yingbo Lu. 2024. RecMind: Large Language Model Powered Agent For Recommendation. *arXiv preprint arXiv:2308.14296* (2024).
[31] Yifan Wu, Lizhu Zhang, Yuhang Zhou, Mingyi Wang, Bo Peng, Serena Li, Xiangjun Fan, and Zhuokai Zhao. 2026. Remember When It Matters: Proactive Memory Agent for Long-Horizon Agents. *arXiv preprint arXiv:2607.08716* (2026).
[32] Yifan Wu, Zhuokai Zhao, Songlin Li, Ho Hin Lee, Jiacheng Zhu, Shirley Wu, Tianhe Yu, Serena Li, Lizhu Zhang, Xiangjun Fan, et al. 2026. SWE-Together: Evaluating Coding Agents in Interactive User Sessions. *arXiv preprint arXiv:2606.29957* (2026).
[33] Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2026. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems* 38 (2026), 17577–17604.
[34] Wujiang Xu, Yunxiao Shi, Zujie Liang, Xuying Ning, Kai Mei, Kun Wang, Xi Zhu, Min Xu, and Yongfeng Zhang. 2025. iAgent: LLM Agent as a Shield between User and Recommender Systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*.
[35] John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems* 37 (2024), 50528–50652.
[36] Zihe Ye, Jingyuan Huang, Weixin Chen, and Yongfeng Zhang. 2026. H-Mem: Hybrid Multi-Dimensional Memory Management for Long-Context Conversational Agents. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7756–7775.
[37] Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, Yinghai Lu, and Yu Shi. 2024. Actions Speak Louder than Words: Trillion-Parameter Sequential Transducers for Generative Recommendations. arXiv:2402.17152 [cs.LG] https://arxiv.org/abs/2402.17152
[38] An Zhang, Yuxin Chen, Leheng Sheng, Xiang Wang, and Tat-Seng Chua. 2024. On Generative Agents in Recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*.
[39] Junjie Zhang, Yupeng Hou, Ruobing Xie, Wenqi Sun, Julian McAuley, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2024. AgentCF: Collaborative Learning with Autonomous Language Agents for Recommender Systems. In *Proceedings of the ACM Web Conference (WWW)*.
[40] Junjie Zhang, Ruobing Xie, Yupeng Hou, Wayne Xin Zhao, Leyu Lin, and Ji-Rong Wen. 2023. Recommendation as Instruction Following: A Large Language Model Empowered Recommendation Approach. arXiv:2305.07001 [cs.IR] https://arxiv.org/abs/2305.07001
[41] Yuyue Zhao, Jiancan Wu, Xiang Wang, Wei Tang, Dingxian Wang, and Maarten de Rijke. 2024. Let Me Do It For You: Towards LLM Empowered Recommendation via Tool Learning. *arXiv preprint arXiv:2405.15114* (2024).
[42] Hailin Zhong, Hanlin Wang, Yujun Ye, Meiyi Zhang, and Shengxin Zhu. 2025. GG-Bond: Growing Graph-Based AI-Agent Society for Socially-Aware Recommender Simulation. *arXiv preprint arXiv:2505.21154* (2025).
[43] Wanjun Zhong, Lianghong Guo, Qiqi Gao, He Ye, and Yanlin Wang. 2024. MemoryBank: Enhancing Large Language Models with Long-Term Memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[44] Yuhang Zhou, Zhuokai Zhao, Ke Li, Spilios Evmorfos, Gökalp Demirci, Mingyi Wang, Qiao Liu, Qifei Wang, Serena Li, Weiwei Li, et al. 2026. LLM-Driven Reasoning for Constraint-Aware Feature Selection in Industrial Systems. *arXiv preprint arXiv:2603.24979* (2026).

A Sample Prompt

To make the harness concrete, Figure 2 gives an abstracted template of the per-cycle decision prompt—the single call in which the model turns the tool-computed context into a proposed configuration. The prompt is abstracted to its operative structure: internal metric, model, and system names are replaced with role placeholders in braces. Both case studies use the same template, differing only in what fills these placeholders (Table 3).

```
System: You are the reasoning core of an agentic harness that
continually optimizes a production recommender. Each cycle you receive
tool-computed context and propose how to reallocate a bounded budget
across the system's control units to improve engagement. A
constrained-optimizer tool then verifies your proposal and, only if it
would exceed the budget, projects it onto the nearest budget-feasible
configuration before deployment.

# Context (produced by the harness's tools)
- Analysis: for each control unit, its current {engagement metrics}
  and {operating cost}, and their change since the previous cycle.
- Budget: the operating budget {B} and each unit's feasible range of
  settings.
- Memory: your configurations from the last {m} cycles and the
  outcomes attributed to them.
- Attribution: the estimated effect of your most recent configuration.

# Task
For each control unit, judge how efficiently its current setting
converts operating budget into engagement, then propose a bounded
adjustment from that unit's feasible set: a continuous multiplier
within a fixed range, or a choice from an ordered discrete menu.
Select only from this set; do not invent actions. Keep the total
within the operating budget {B}. Give a brief rationale and a
confidence for each.

# Output (JSON)
{
  "decisions": [
    {"unit": "<id>", "adjustment": "<action from feasible set>",
     "rationale": "<why>", "confidence": "high|medium|low"}
  ],
  "assessment": "<cross-unit synthesis of what is working and
   what is not>"
}
```

Figure 2: Abstracted template of the per-cycle decision prompt. Placeholders in braces are filled at runtime by the harness's tools; the model proposes bounded per-unit adjustments, which the constrained optimizer renders budget-feasible before deployment.

Table 3: How the two case studies instantiate the template's placeholders.

Placeholder	Retrieval-budget case	Serving-capacity case
control unit	retrieval source	user segment
feasible set	bounded continuous multiplier	ordered discrete treatment menu
engagement metric	video-viewing sessions, watch time	engaged sessions, time spent
operating budget	retrieval budget	serving compute budget