SR-Agent: An Experience-Driven Agentic Framework for Post-Ranking Strategy Refinement in E-Commerce Recommendation

Hanchen Yang[1,†], Kaiwen Yang[1,†], Junpeng Zhuang[1,†], Yang He[1], Keting Cen[1], Bochao Liu[1,*], Zhongbo Sun[1,*], An Liu[1], Zhongteng Han[1] and Chenyi Lei[1]

[1]*Kuaishou Technology, Beijing, China*

Abstract

User experience is a first-class objective in industrial e-commerce recommender systems (RS). *Post-ranking strategies*, which govern diversity, similarity, and exposure over a ranked list, are widely deployed in industrial RS for their simplicity and low serving cost. However, as the online recommendation environment evolves continuously, these statically configured strategies gradually become stale, thereby degrading the user experience. Refining them typically relies on manual inspection, diagnosis, and updates, making it slow, costly, and difficult to scale or reuse. Although recent LLM-based agents (e.g., RecUserSim, SimUSER, and Self-EvolveRec) offer promising directions, none of them close the full loop of automated, self-evolving strategy refinement. To bridge this gap, we introduce **SR-Agent**, which, to the best of our knowledge, is the first agentic framework deployed to refine post-ranking strategies in industrial RS. SR-Agent unifies three components: (i) a **UserSim** agent that applies inspection skills to surface user-perceived bad cases; (ii) an **Analysis** agent that consolidates recurring bad cases into structured, reusable diagnoses; and (iii) a constrained **Strategy Refinement Harness** that maps diagnoses to typed and bounded actions, gated by a four-stage reward pipeline with reversible rollback. Deployed on the Kuaishou e-commerce platform, SR-Agent continuously runs this refinement loop and, in a one-month online A/B test, increases order volume by 0.71%, browsing depth by 0.34%, and clicked-category diversity by 0.48%, while markedly shortening the refinement cycle and lowering operational cost.

Keywords

E-commerce recommendation, multi-agent systems, user experience, large language models

1. Introduction

Improving user experience has always been a primary goal of e-commerce recommender systems (RS) [1, 2]. Beyond accurately identifying items that users are likely to click or purchase, an RS should organize its recommendations to provide a coherent and satisfying experience throughout the shopping journey [3, 4]. However, optimizing pointwise relevance alone does not necessarily lead to a high-quality recommendation experience. For example, an RS may recommend multiple visually similar products because each item has a high predicted probability of being clicked or purchased. Despite their individual relevance, presenting these items together offers users limited alternatives and impedes efficient interest exploration. Such experience-level deficiencies can reduce user satisfaction and engagement, ultimately affecting purchase conversion and long-term retention [5, 6].

Over the past decades, RS research has increasingly moved beyond predictive accuracy toward broader experience objectives such as diversity, novelty, and reduced repetition. Existing studies address these objectives from multiple angles [7, 8, 9, 10]: *re-ranking* methods (e.g., MMR, DPP) post-process candidate lists to trade off relevance against diversity or novelty; *regularized training objectives* bake experience signals into the loss via multi-task learning; and *reinforcement-learning-based policies* cast recommendation as sequential decision-making to optimize long-horizon rewards such as fatigue or return rate. Beyond these learning-based paradigms, a large family of rule-based *post-ranking strategies* has proven remarkably effective in practice, and has become a dominant mechanism in large-scale

GenAIECommerce'26: The Third Workshop on Agentic and Generative AI for E-Commerce, co-located with RecSys, September 28, 2026, Minneapolis, MN, USA

*Corresponding authors.

†Equal contribution.

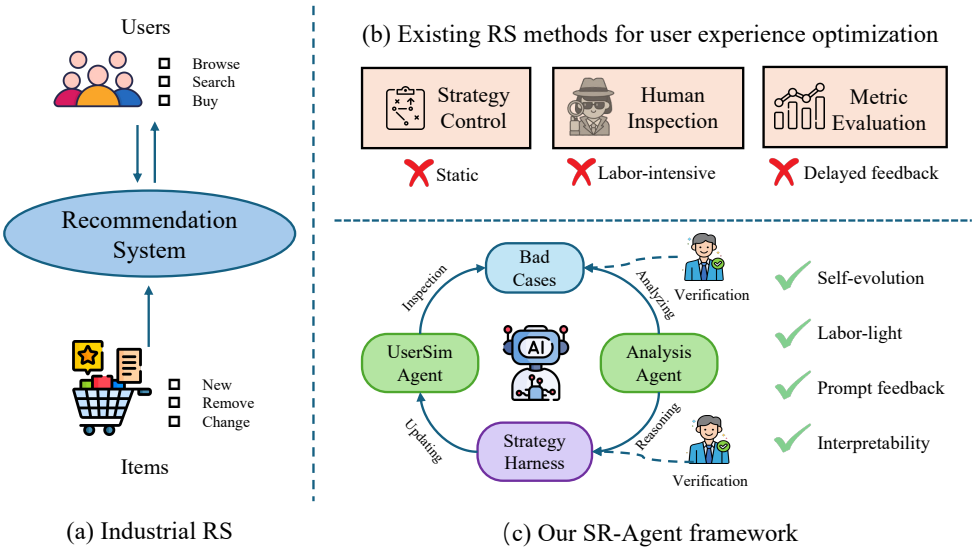

Figure 1: Overview of the motivation and proposed SR-Agent framework. Existing maintenance of post-ranking strategies relies on static configurations, manual inspection, and delayed metric feedback. Our agentic framework closes the loop from bad-case discovery and diagnosis to bounded strategy updates, offline verification, and controlled online validation.

industrial RS. Their low serving cost, operational flexibility, interpretability, and seamless compatibility with existing pipelines make them particularly well suited to large-scale deployment. Despite these advantages, two significant real-world challenges in industrial RS still need to be addressed.

Challenge 1: Static post-ranking strategies degrade as the recommendation environment evolves, leading to recurring experience bad cases. Industrial recommenders commonly employ post-ranking strategies to control category, shop, brand, price-range, item-similarity, and exposure patterns. These strategies are typically configured using predefined item attributes, item relations, and fixed thresholds derived from historical observations [11]. However, the environment in which they operate evolves continuously. As shown in Fig. 1, on the item side, products are frequently introduced, removed, repriced, relabeled, or updated in their titles and visual content, causing previously established attributes, relations, and thresholds to become stale. On the user side, preferences for product comparison, diversity, and repeated exposure vary with evolving interests, shopping intent, session state, and recent interactions [12]. As static configurations gradually become misaligned with the current environment, they may incorrectly suppress useful alternatives or fail to remove redundant and functionally substitutable items. Consequently, their effectiveness degrades over time, giving rise to recurring experience bad cases that require continual strategy refinement.

Challenge 2: Human-intensive strategy refinement is slow, costly, and difficult to scale. When experience bad cases emerge, industrial systems typically rely on a multi-stage manual workflow to refine the corresponding post-ranking strategies. Product or operation specialists first inspect recommendation results and identify potential experience defects; domain experts then analyze their causes and communicate the findings to algorithm engineers; the engineers translate these case-level observations into adjustments to category mappings, thresholds, penalties, or other strategy configurations; and the updated results must be inspected and verified again before deployment. Although this workflow is well established, it requires repeated coordination across multiple roles, incurs substantial human and time costs, and leads to long refinement cycles. Moreover, the quality of bad-case diagnosis and strategy adjustment depends heavily on the experience of individual experts. The resulting knowledge is often scattered across isolated cases and difficult to preserve, reuse, and improve in subsequent iterations.

Although recent developments in large language models (LLMs) and LLM-based agents offer promising directions for automating parts of this workflow, none of them close the full loop of automated,

self-evolving strategy refinement. A number of LLM-as-a-judge methods enable scalable, reference-free evaluation of complex outputs [13, 14], while LLM-based user simulators model user preferences, feedback, and sequential behavior for offline evaluation and training [15, 16, 17, 18]. However, these approaches primarily produce evaluation signals and do not directly convert detected experience defects into executable strategy updates [14, 19, 16]. More recent self-evolving systems further use agents to propose, implement, and validate changes to recommendation models, objectives, or production code [20, 21, 22, 23]. Nevertheless, these systems primarily target recommendation decisions or broad model and code optimization. A scalable workflow that continuously connects user-perceived bad-case detection, cause diagnosis, reusable knowledge accumulation, bounded post-ranking strategy refinement, and deployment validation remains underexplored.

To address above issues, we introduce **SR-Agent**, an agentic framework that automates bad-case detection and diagnosis, accumulates reusable refinement knowledge, and supports bounded and verifiable updates to *post-ranking strategies*. Concretely, the **UserSim agent** inspects recommendation pages and session sequences through staged skills for exposure-pattern analysis, item-relation inference, and context-conditioned experience assessment, and outputs structured bad cases. The **Analysis agent** explains each bad case by identifying the underlying relation type and failure cause, dynamic item substitution, taxonomy mismatch, loose thresholds, dense session exposure, etc, and abstracts recurring failure patterns into a reusable Diagnosis Memory, while the underlying bad cases are retained in Bad-Case Memory for retesting. The **Strategy refinement harness** maps these diagnoses to a predefined strategy action space and generates candidate updates through four typed actions: relation-threshold adjustment, redundancy-suppression adjustment, exposure-limit adjustment, and authorized taxonomy correction. Candidate updates must pass bad-case retesting, traffic replay, human verification, and online A/B validation before deployment.

Our main contributions are summarized as follows:

- We formulate post-ranking strategy maintenance as an automated and self-evolving closed loop from experience inspection and diagnosis to bounded refinement and verification. By running continuously, the loop enables deployed strategies to adapt to the evolving online environment.
- We develop SR-Agent, which combines two reasoning agents with a guarded refinement harness to automate bad-case analysis, accumulate reusable knowledge, and safely update post-ranking strategies, replacing the traditionally human-intensive workflow.
- Continuous experiments on Kuaishou, a large-scale e-commerce platform, demonstrate that SR-Agent improves order volume by 0.71%, browsing depth by 0.34%, and clicked-category diversity by 0.48%, while shortening the strategy refinement cycle and lowering costs.

2. Related Work

2.1. Post-Ranking Strategies for User Experience

User experience in recommender systems extends beyond predictive accuracy. Accuracy-oriented evaluation can overlook perceived usefulness, satisfaction, choice difficulty, and trust [6, 24, 4], while diversity, novelty, serendipity, and unexpectedness provide complementary perspectives on list- and session-level quality [8, 5, 25]. To operationalize these objectives after ranking, industrial recommenders employ *post-ranking strategies*: rule-based or parameterized controls that modify the final list without changing the base ranker. Such strategies include category or shop quotas, similarity suppression, exposure caps, and algorithmic diversification based on MMR [26], topic diversification [7], DPP [27, 28], or temporal diversity [9]. Existing post-ranking strategies typically operate on predefined taxonomy labels, item representations, similarity functions, and manually configured thresholds. These signals are useful proxies for item relations, but they do not by themselves determine whether a displayed relation constitutes useful comparison, redundancy, or fatigue under a particular user and session context. Prior work primarily studies how to design or optimize a post-ranking method for producing an improved list; considerably less attention has been paid to maintaining deployed strategies after user-perceived

failures emerge in live traffic. SR-Agent addresses this operational problem by detecting bad cases, diagnosing why the current strategies missed them, and translating structured diagnoses into bounded, auditable updates, therefore refining existing post-ranking strategies.

2.2. LLM-Based User Simulation in Recommender Systems

LLM-based user simulators provide a scalable means of modeling user feedback and interactions for the evaluation and training of recommender systems. Early work such as Agent4Rec constructs generative user agents with profile, memory, and action modules to simulate page-level recommendation interactions [15]. For conversational recommendation, Yoon et al. [16] introduce an evaluation protocol covering item selection, binary and open-ended preference expression, recommendation requests, and feedback, and show that LLM simulators can still deviate systematically from human behavior. Zhang et al. [29] instead model explicit user engagement with recommended items by combining LLM-derived preference logic with a statistical model, producing simulated interactions for reinforcement-learning-based recommender training. RecUserSim improves the realism and diversity of conversational user simulation through profile, memory, bounded-rationality-inspired action, and response-refinement modules, while additionally producing explicit ratings for quantitative evaluation [30]. Moving beyond static user profiles, DyTA4Rec incorporates dynamic profile updating, temporally enhanced prompting, and adaptive aggregation to capture evolving user behavior [18]. SimUSER further integrates persona, memory, perception, and behavior modules to support recommender evaluation and simulator-guided refinement [17]. These studies primarily simulate user responses, engagement, or interaction trajectories for recommender evaluation and training. UserSim in SR-Agent instead combines user simulation with an LLM-as-a-judge mechanism: conditioned on the user's behavior history and the current recommendation context, the LLM adopts the user's perspective to assess realized lists and sessions for potential experience defects. Unlike a generic LLM judge that evaluates an output against a task-level rubric [13, 14], UserSim performs context-aware experience assessment and produces evidence-grounded bad cases rather than a single aggregate score. These assessments are quality-controlled through human audits and provide actionable inputs for subsequent strategy diagnosis and refinement.

2.3. Agentic Recommender Systems

LLM-based agents have extended recommender systems from one-shot prediction toward interactive and reasoning-intensive recommendation. Existing studies employ agent capabilities such as memory, planning, tool use, and multi-agent coordination for preference elicitation, conversational recommendation, user–item reasoning, and explanation [31, 32, 33, 34, 35]. For instance, Self-EvolveRec integrates user simulation with model diagnosis to provide directional feedback for iterative recommendation-model evolution [20]. Industrial systems further employ LLM agents as machine-learning engineers to optimize model architectures, learning algorithms, and reward functions through offline and online feedback loops [21], while AgentX automates the broader research cycle from hypothesis generation and code modification to online experimentation and knowledge accumulation [22]. These systems demonstrate the potential of agents to close the loop of recommender optimization, but mainly focus on evolving recommendation models or automating the model-development workflow [36]. In contrast, SR-Agent deliberately narrows self-evolution to a typed, bounded strategy layer, trading expressive power for auditability, fast validation, and low-risk deployment in always-on production maintenance.

3. Method

3.1. Preliminaries and Problem Setup

We consider a production recommender in which a base ranker produces, for a user u under serving context c, a scored candidate pool that is processed by a set of post-ranking strategies, jointly represented by π_S, before display. π_S is parameterized by a strategy state $S \in \mathcal{S}$ that encodes diversity thresholds,

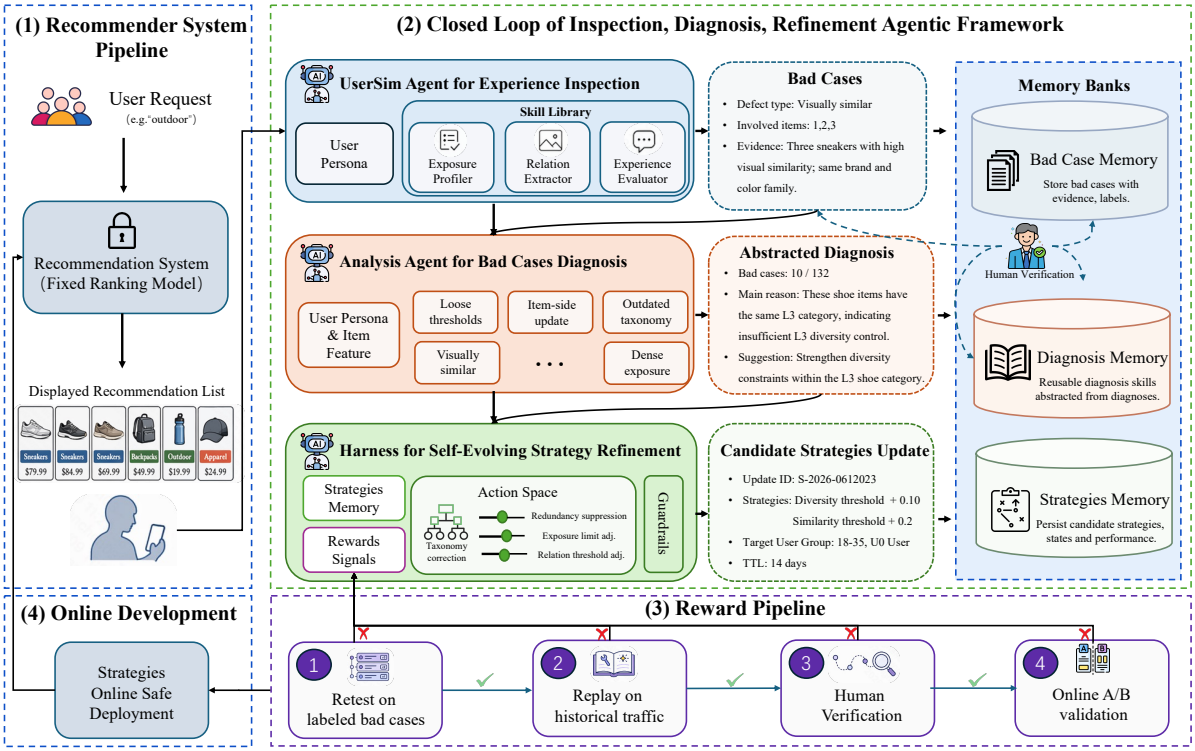

Figure 2: Overview of the proposed agentic framework. The base recommender produces displayed recommendation lists with a fixed ranking model. The agentic loop then performs user-perspective inspection, bad-case diagnosis, bounded strategy refinement, memory update, reward validation, and safe online deployment.

similarity suppressions, item/relation penalties, exposure caps, and item groupings. The displayed list $L = \pi_S(u, c) = (i_1, \ldots, i_K)$ is what the user actually observes.

SR-Agent leaves the base ranker frozen and refines only π_S. Given traffic $\mathcal{T} = \{(u, L, c)\}$, deployed state S, logged history $\mathcal{D}_{\text{hist}}$, and a fixed typed action space $\mathcal{A}$, the loop produces a sequence of candidate updates $\{a_t\} \subset \mathcal{A}$ that must pass a guarded admission pipeline before being applied to S. SR-Agent retrieves accumulated experience from three memories to inform subsequent inspection, diagnosis, and strategy refinement.

3.2. Framework Overview

Figure 2 illustrates the data flow of SR-Agent. The left part shows the production recommender pipeline: a user request is processed by the existing recommender system, whose ranking model remains fixed, to produce a displayed recommendation list. SR-Agent operates on this list without replacing the base recommender. The middle part presents the closed agentic loop. Conditioned on the user persona, the UserSim agent inspects the displayed list using rule-based, visual, and intent-reasoning skills and produces structured bad cases. The Analysis agent abstracts these cases into diagnoses that describe the underlying failure causes and corresponding refinement directions. Based on the diagnoses, current strategy states, and historical reward signals, the strategy-refinement harness generates bounded candidate strategy updates. The right part maintains the resulting bad cases, diagnoses, and strategy records in three memory banks. Finally, the bottom reward pipeline evaluates each candidate through labeled bad-case retesting, historical-traffic replay, human verification, and online A/B testing before safe deployment. Validation outcomes are fed back into the framework to guide subsequent refinements.

3.3. UserSim Agent

Aggregate performance metrics may fail to capture list- and session-level experience defects: individually relevant items can still form a repetitive, homogeneous, or fatiguing display. UserSim therefore approximates how a user perceives the displayed results, rather than simulating a complete trajectory of future user actions. Given a user u, a displayed list L, and serving context c, it constructs an inspection context from the user's recent behaviors and exposure history, together with item attributes, category hierarchy, product images and titles, shop, and price information.

UserSim examines this context through three complementary inspection skills that are invoked in a staged manner:

- The **Exposure Profiler skill** identifies observable list- and session-level patterns, including repeated exposure and excessive concentration over categories, shops, and price bands. It characterizes what the user repeatedly encounters.
- The **Relation Extractor skill** identifies functional relations among the displayed items, including relations not captured by existing taxonomy labels. It provides evidence of why a group of items may be perceived as similar, substitutable, or complementary.
- The **Experience Evaluator skill** integrates the preceding evidence with the user's current intent, recent behavior, and exposure history. It determines whether the observed recommendation list support useful comparison or instead constitute redundancy, homogeneity, or browsing fatigue.

Formally, the *Exposure Profiler* and *Relation Extractor* produce exposure evidence e_b^{exp} and relation evidence e_b^{rel}, respectively. The *Experience Evaluator* then produces the defect type y, the involved items $I_b \subseteq L$, context-conditioned evidence e_b^{ctx}, and confidence q_b:

$$(y,\, I_b,\, e_b^{\mathrm{ctx}},\, q_b) = \mathrm{ExpEval}\big(u,\, c,\, e_b^{\mathrm{exp}},\, e_b^{\mathrm{rel}}\big), \tag{1}$$

where $\mathrm{ExpEval}$ denotes the Experience Evaluator. It integrates the exposure and relation evidence with the user's current intent u, recent behavior, and exposure history c to yield the final experience judgment. The resulting bad case is stored as

$$b = \big(y,\, I_b,\, e_b,\, q_b\big), \qquad e_b = \{e_b^{\mathrm{exp}},\, e_b^{\mathrm{rel}},\, e_b^{\mathrm{ee}}\}. \tag{2}$$

3.4. Analysis Agent

UserSim decides *whether* a displayed result is an experience bad case; the Analysis agent decides *why the deployed post-ranking strategies let it through, and where to intervene.* The separation matters because a single visible symptom—say, three visually similar shoes in the top-6—can arise from very different operational causes: missing relation signals, stale item groupings, miscalibrated thresholds, or loose exposure control. Conflating them leads to the wrong fix.

Given a bad-case cluster $B \subseteq \mathcal{M}_b$ (grouped by defect type, involved items or categories, evidence patterns, and serving scenario) and the deployed strategy state S, the Analysis agent runs a structured chain-of-thought that narrows the diagnosis in three stages, each conditioned on the previous:

- **What do the bad cases have in common?** Consolidate the per-case evidence produced by UserSim across the cluster B and extract the shared patterns, the recurring item-relation type z (visual or semantic similarity, functional substitution, complementarity, or repeated exposure), together with the co-occurring item, category, and context signals. This step turns a bag of noisy per-case observations into a single, cluster-level hypothesis about the underlying item-level phenomenon, and filters out one-off noise before any causal reasoning begins.
- **Why do the current strategies miss it?** Contrast the shared pattern under z against the deployed strategy state S to attribute an operational cause h: missing signal coverage, stale grouping, threshold miscalibration, insufficient exposure control, or a mismatch between strategy tolerance and user context.

- **How should we fix it?** Given (z, h), infer the smallest defensible scope s, an item group, category subtree, serving scenario, or user segment, and a high-level refinement direction v (tighten or relax a constraint, adjust a threshold or penalty, revise an authorized grouping).

Each diagnosis is recorded as

$$d = (z, h, s, v, e_d), \tag{3}$$

where (z, h, s, v) carry the outputs of the three CoT stages and e_d bundles the supporting bad cases, consolidated evidence, and the relevant slice of S for downstream retracing. In the continuously running production loop, human reviewers randomly audit approximately 1% of the bad cases produced by UserSim and 1% of the diagnoses produced by the Analysis agent. This sampled quality-control audit is distinct from the guarded validation required for candidate strategy updates before deployment.

3.5. Strategy Refinement Harness

The strategy-refinement harness provides a constrained interface between diagnosis and online deployment. Its action space, output schema, scope permissions, and deployment guardrails are fixed and cannot be modified by the LLM. Its adaptive state is Strategy Memory $\mathcal{M}_s$, which records previous proposals and their validation outcomes.

Typed action space. Before the optimization loop, strategy owners configure and version the typed action space $\mathcal{A}$ summarized in Table 1. The LLM cannot emit serving code, alter model objectives, target unauthorized scopes, or invoke operations outside $\mathcal{A}$.

The action types address different user-experience failures. A *relation-threshold* action corrects overly permissive or restrictive identification of similar or substitutable items. A *redundancy-suppression* action adjusts how strongly identified relations affect the displayed list. An *exposure-limit* action controls excessive concentration over dimensions such as category, shop, and price band. Finally, a *taxonomy correction* repairs authorized taxonomy mappings that prevent functionally equivalent items from being treated consistently.

Figure 2 uses serving-side shorthand for these schema-level actions: threshold updates correspond to relation-threshold adjustments; item-side penalties and dispersion-strength changes correspond to redundancy-suppression adjustments; segment-tolerance changes are implemented through exposure-limit configurations over authorized scopes; and taxonomy changes correspond to taxonomy corrections. The candidate card in the figure shows resulting configured strategy values, whereas Table 1 specifies the admissible shift applied by any single candidate update.

Table 1
Typed action space $\mathcal{A}$. Parameter bounds, target whitelists.

Action type	Parameters	Admissible domain
Relation threshold adjustment	Relation r; threshold shift $\Delta\tau_r$	$r \in \{\textsc{visual}, \textsc{semantic}, \textsc{substitute}\}$; $\Delta\tau_r \in [-0.15, +0.15]$, step 0.01
Redundancy suppression adjustment	Group g; strength shift $\Delta\lambda$	$g \in \mathcal{G}_{\text{auth}}$; $\Delta\lambda \in [-0.50, +0.50]$, step 0.05
Exposure limit adjustment	Dimension d; horizon h; limit k	$d \in \{\textsc{category}, \textsc{shop}, \textsc{price-band}\}$; $(h, k) \in \mathcal{C}_{\text{cfg}}$
Taxonomy correction	Operation o; source node(s) n; target group g	$o \in \{\textsc{merge}, \textsc{split}, \textsc{reassign}\}$; $(n, g) \in \mathcal{C}_{\text{tax}}^{\text{auth}}$

Candidate generation and validation. For each diagnosis d, the harness retrieves relevant precedents

$$H = \text{Retrieve}(d, \mathcal{M}_s)$$

according to diagnosis, scope, and historical outcome. It then generates a small candidate set

$$C = G(d, S, H; \mathcal{A}) \subseteq \mathcal{A}, \qquad a = (\mathrm{id}, s, t, p, E, T), \tag{4}$$

where s is the authorized scope, t the action type, p its bounded parameters, E the supporting evidence, and T the time-to-live.

A deterministic validator rejects candidates with invalid parameters, unauthorized scopes, unsupported operations, or incompatible strategy versions. Compatible parameter updates may be composed and clipped to their authorized bounds; conflicting updates and taxonomy operations are escalated for strategy-owner review.

Guarded evaluation and memory update. Each valid candidate passes four stages. First, the harness invokes the pre-registered strategy script associated with the candidate action on diagnosis-matched cases from $\mathcal{M}_b$. The LLM supplies only bounded parameters and cannot generate or modify the script. This bad-case retest measures whether the targeted defect is corrected. Second, the same script is replayed on the larger historical sample $\mathcal{D}_{\mathrm{hist}}$ to estimate population-level coverage, list changes, and unintended regressions. Thus, bad-case retesting measures targeted effectiveness, whereas historical traffic replay measures generality and safety beyond the selected cases. Candidates that pass both offline stages proceed to strategy-owner review and controlled online A/B testing. Their outcomes are recorded as

$$r_t = \left(r_t^{\mathrm{case}}, r_t^{\mathrm{replay}}, r_t^{\mathrm{human}}, r_t^{\mathrm{online}}\right). \tag{5}$$

Failure at any stage stops the candidate, while an online guardrail violation triggers rollback. The complete outcome is stored in Strategy Memory:

$$\mathcal{M}_s^{t+1} = \mathcal{M}_s^t \cup \{(d_t, a_t, r_t, o_t)\}, \tag{6}$$

where $o_t \in \{\mathrm{invalid}, \mathrm{rejected}, \mathrm{deployed}, \mathrm{rolled\text{-}back}\}$.

Algorithm 1 summarizes the refinement loop.

4. Experiments

To evaluate SR-Agent in a real-world industrial setting, we deployed it on Kuaishou's e-commerce platform, a large-scale, content-driven platform serving tens of millions of daily active users. To isolate the effect of SR-Agent, the control group retained the existing manually configured post-ranking strategies, whereas the treatment group kept the base ranking model unchanged and allowed SR-Agent to refine only predefined, bounded strategies. Within this deployment, we conducted a one-month online A/B test, with 2.5% of the traffic assigned to the treatment group. During the test period, the deployment covered approximately 60 million users and 70 million items. Our evaluation addresses the following research questions:

- **RQ1**: Does SR-Agent improve online user experience and business metrics?
- **RQ2**: To what extent are the bad cases identified by UserSim validated by human reviewers?
- **RQ3**: How much does SR-Agent reduce case inspection and end-to-end strategy iteration time compared with the legacy manual workflow?
- **RQ4**: How do user-experience metrics evolve over successive iterations of the SR-Agent optimization loop?

4.1. RQ1: Online A/B Test

Table 2 summarizes the one-month online A/B test in terms of user experience and order-related performance. SR-Agent improves all reported metrics. It increases the number of clicked categories, browsing depth, and clicks, while also improving order volume and gross merchandise volume (GMV). These results demonstrate that SR-Agent enhances the browsing experience while delivering positive business outcomes.

Algorithm 1 SR-Agent strategy-refinement loop

Require: traffic $\mathcal{T}$, state S, memories $\mathcal{M}_b, \mathcal{M}_d, \mathcal{M}_s$, history $\mathcal{D}_{\text{hist}}$, action space $\mathcal{A}$

$\mathcal{B}_{\text{new}} \leftarrow \varnothing$
for $(u, L, c) \in \mathcal{T}$ **do**
  $b \leftarrow \textsc{UserSim}(u, L, c)$
  if b is valid and sufficiently confident **then**
    append b to $\mathcal{M}_b$ and $\mathcal{B}_{\text{new}}$
  end if
end for
for cluster $B \in \textsc{ClusterBadCases}(\mathcal{B}_{\text{new}})$ **do**
  $d \leftarrow \textsc{Analysis}(B, \mathcal{M}_d)$
  if d satisfies the diagnosis contract **then**
    append d to $\mathcal{M}_d$
    $H \leftarrow \textsc{RetrievePrecedents}(d, \mathcal{M}_s)$
    $C \leftarrow \textsc{Harness}(d, S, H, \mathcal{A})$
    for $a \in C$ **do**
      $a' \leftarrow \textsc{ValidateAndResolve}(a, S, \mathcal{A})$
      if $a' = \varnothing$ **then**
        record $(d, a, \varnothing, \text{invalid})$ in $\mathcal{M}_s$
      else
        $(r, o) \leftarrow \textsc{GuardedEvaluate}(a', \mathcal{M}_b, \mathcal{D}_{\text{hist}})$
        if $o = \text{deployed}$ **then**
          $S \leftarrow \textsc{Deploy}(S, a')$
        end if
        record (d, a', r, o) in $\mathcal{M}_s$
      end if
    end for
  end if
end for
return updated $S, \mathcal{M}_b, \mathcal{M}_d, \mathcal{M}_s$

Table 2
One-month online A/B results of SR-Agent. Values are relative changes compared with the control group.

Metric group	Metric	Relative change
User experience	Clicked categories per user (1-day)	$+0.425\%$
User experience	Clicked categories per user (7-day)	$+0.482\%$
User experience	Browsing depth	$+0.342\%$
User experience	Number of clicks	$+0.554\%$
Order-related	Order volume	$+0.707\%$
Order-related	Gross merchandise volume (GMV)	$+0.461\%$

4.2. RQ2: Effectiveness of Bad-Case Detection

We evaluate bad-case detection on 500 recommendation cases sampled from production traffic. As shown in Table 3,Manual inspection identifies 36 bad cases. UserSim without user intent detects 114 cases with a 91% validation rate, while UserSim with user intent detects 142 cases with a 95% validation rate. Compared with manual inspection, although UserSim produces a small number of false positives, it identifies substantially more valid bad cases than manual inspection, demonstrating broader coverage and greater effectiveness.

Table 3
Bad-case detection results on 500 recommendation cases.

Method	Detected cases	Validation rate
Manual inspection	36	100%
UserSim without user intent	114	91%
UserSim with user intent	142	95%

4.3. RQ3: Human-in-the-Loop Operational Efficiency

We compare SR-Agent with the legacy workflow in terms of model usage, stage-level time, and end-to-end iteration time. All LLM-based components use Claude Sonnet 4.6, and one complete loop consumes 36 Million aggregate tokens across all agent calls. Importantly, SR-Agent does not remove human participation: it automates large-scale inspection, diagnosis, and candidate generation, while human reviewers randomly audit approximately 1% of generated bad cases and diagnoses and review candidate strategy updates before online deployment.

Table 4
Operational efficiency of one strategy-refinement loop. Stage-level results report automated runtime and human effort separately; end-to-end iteration denotes elapsed wall-clock time.

Dimension	Legacy workflow	SR-Agent (automation + human)
LLM and capacity	No LLM	Claude Sonnet 4.6
Average Tokens per loop	0	36 Million
Inspection of 500 cases	7 days	50 min automation + 1 reviewer-day
Analysis of 500 bad cases	2 days	90 min automation + 1 reviewer-day
Strategy update	2 weeks	3 h automation + 1 reviewer-day
End-to-end iteration	2–3 weeks	3–5 elapsed days

4.4. RQ4: Bad-Case Resolution and Online Trend

We track how SR-Agent improves over ten successive strategy updates, each produced by one invocation of the refinement harness on a single bounded strategy. For update t, we report the *bad-case resolution rate*: the percentage of targeted bad cases whose defect disappears in post-update retesting while passing the replay guardrails. In parallel, we monitor the one-month online rollout (June 15–July 15, 2026), measuring the relative change in number of clicks and browsing depth against the baseline.

As shown in Figure 3(a), the resolution rate climbs from 49.2% at the first update to 83.1% at the tenth (+33.9 pp), with diminishing per-update gains as common failures are fixed early and later updates target rarer, harder cases. Figure 3(b) shows a matching online trend: despite short-term fluctuation from traffic composition and user behavior, both metrics rise overall, reaching +0.55% clicks and +0.34% browsing depth by the end of the rollout. Together, these results indicate that the harness's growing ability to resolve bad cases translates into a progressively better online browsing experience.

4.5. Case Study

We further examine two representative production cases, shown in Figure 4, to illustrate how SR-Agent connects user-perceived defects to actionable strategy diagnoses.

Case 1: Cross-category repetition of functionally equivalent items. In the recommendation list, three nearly identical handheld massage products appear at positions 7, 12, and 13. Although they address the same user need, they are assigned to different catalog branches—massage hammers, manual massagers, and fitness massage tools. Consequently, the deployed category-based post-ranking strategy fails to recognize them as redundant exposures. By jointly considering item semantics, functional relations, and list context, UserSim identifies both repeated and locally clustered exposure. The Analysis

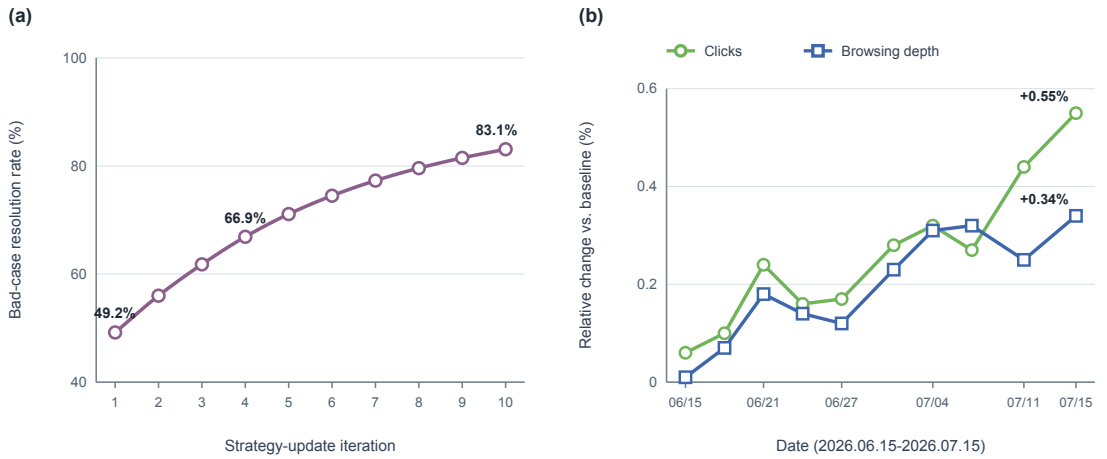

Figure 3: Results over ten strategy updates. (a) Per-update bad-case resolution rate of the refinement harness. (b) Relative change in number of clicks and browsing depth during the one-month online rollout, versus the pre-rollout baseline.

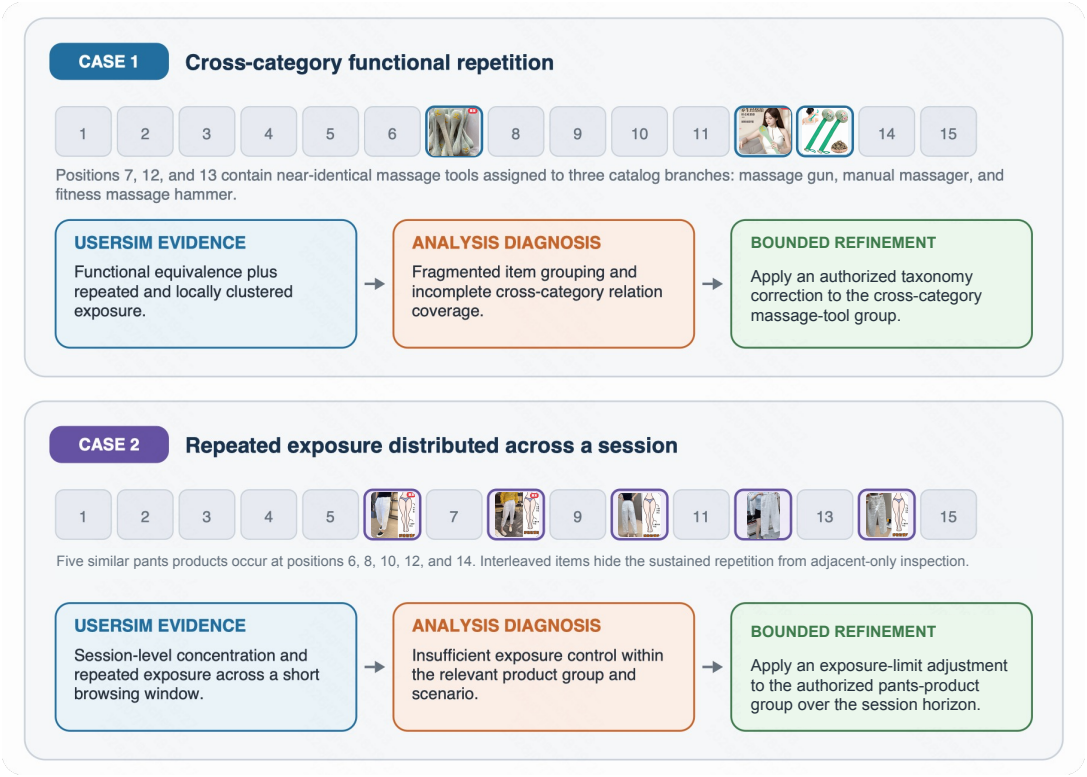

Figure 4: Two cases processed by SR-Agent. The first requires cross-category item-relation inference, whereas the second requires session-level exposure analysis. In both cases, UserSim evidence is converted into an actionable diagnosis and mapped to a bounded post-ranking strategy refinement.

agent attributes the defect to incomplete relation coverage caused by fragmented item groupings and localizes its scope to the corresponding cross-category massage-tool group. Based on this diagnosis, the harness can generate an authorized taxonomy-correction candidate that merges or reassigns the involved catalog nodes to the cross-category massage-tool group.

Case 2: Repeated exposure distributed across a session. In another item session, five highly similar pants products appear at positions 6, 8, 10, 12, and 14. Because the items are interleaved with unrelated products, no single adjacent pair fully captures the user-facing repetition, and the online strategies with window 2 are not activated. By aggregating exposure evidence over the session, UserSim

identifies a sustained concentration pattern that would be missed by purely local inspection. The Analysis agent attributes the defect to insufficient exposure control within the relevant product group and recommends tightening the exposure limit for the authorized pants-product group over a configured session horizon. The harness expresses this direction as an exposure-limit adjustment and submits the resulting candidate to the guarded validation pipeline.

5. Conclusion

We presented SR-Agent, an agentic framework that closes the loop from user-perceived bad-case detection to bounded, reward-validated refinement of post-ranking strategies in industrial e-commerce recommendation. A UserSim agent surfaces evidence-grounded bad cases, an Analysis agent consolidates them into reusable diagnoses, and a constrained refinement harness turns each diagnosis into typed, bounded actions that are admitted only after a four-stage reward pipeline with reversible rollback. This decomposition keeps every online change interpretable, auditable, and safe to revert, which is what makes continuous self-refinement acceptable in production. Deployed on Kuaishou, SR-Agent delivers on both axes we set out to improve. On effectiveness, a one-month online A/B test yields consistent gains in clicked-category diversity, browsing depth, clicks, and order volume. On efficiency, SR-Agent significantly compresses the end-to-end strategy-refinement cycle that previously required cross-role manual work, enabling faster iteration while retaining lightweight human audits. In the future, we plan to generalize the agentic loop to other components of industrial RS, such as cold-start boosting and multi-channel retrieval quotas.

Declaration on Generative AI

During the preparation of this work, the authors only used GenAI for language polishing, including grammar correction and improvements to clarity and fluency. The authors reviewed and edited all suggested revisions and take full responsibility for the content of this paper.

References

[1] Q. Peng, H. Liu, H. Huang, Q. Yang, M. Shao, A survey on llm-powered agents for recommender systems, arXiv preprint arXiv:2502.10050 (2025).

[2] I. Adaji, Towards improving e-commerce users experience using personalization & persuasive technology, in: Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization, 2017, pp. 318–321.

[3] X. Song, User-centric internal tools in e-commerce: Enhancing operational efficiency through ai integration, in: Proceedings of the 2025 International Conference on Economic Management and Big Data Application, 2025, pp. 702–706.

[4] P. Pu, L. Chen, R. Hu, Evaluating recommender systems from the user's perspective: survey of the state of the art, User Modeling and User-Adapted Interaction 22 (2012) 317–355.

[5] T. Duricic, D. Kowald, E. Lacic, E. Lex, Beyond-accuracy: a review on diversity, serendipity, and fairness in recommender systems based on graph neural networks, Frontiers in big data 6 (2023) 1251072.

[6] S. M. McNee, J. Riedl, J. A. Konstan, Being accurate is not enough: how accuracy metrics have hurt recommender systems, in: CHI'06 extended abstracts on Human factors in computing systems, 2006, pp. 1097–1101.

[7] C.-N. Ziegler, S. M. McNee, J. A. Konstan, G. Lausen, Improving recommendation lists through topic diversification, in: Proceedings of the 14th international conference on World Wide Web, 2005, pp. 22–32.

[8] M. Kunaver, T. Požrl, Diversity in recommender systems–a survey, Knowledge-based systems 123 (2017) 154–162.

[9] N. Lathia, S. Hailes, L. Capra, X. Amatriain, Temporal diversity in recommender systems, in: Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval, 2010, pp. 210–217.
[10] L. Chen, G. Zhang, E. Zhou, Fast greedy map inference for determinantal point process to improve recommendation diversity, Advances in neural information processing systems 31 (2018).
[11] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, E. H. Chi, Top-k off-policy correction for a reinforce recommender system, in: Proceedings of the twelfth ACM international conference on web search and data mining, 2019, pp. 456–464.
[12] S. Wang, L. Cao, Y. Wang, Q. Z. Sheng, M. A. Orgun, D. Lian, A survey on session-based recommender systems, ACM Computing Surveys (CSUR) 54 (2021) 1–38.
[13] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, C. Zhu, G-eval: Nlg evaluation using gpt-4 with better human alignment, in: Proceedings of the 2023 conference on empirical methods in natural language processing, 2023, pp. 2511–2522.
[14] L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. Xing, et al., Judging llm-as-a-judge with mt-bench and chatbot arena, Advances in neural information processing systems 36 (2023) 46595–46623.
[15] A. Zhang, Y. Chen, L. Sheng, X. Wang, T.-S. Chua, On generative agents in recommendation, in: Proceedings of the 47th international ACM SIGIR conference on research and development in Information Retrieval, 2024, pp. 1807–1817.
[16] S.-e. Yoon, Z. He, J. Echterhoff, J. McAuley, Evaluating large language models as generative user simulators for conversational recommendation, in: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2024, pp. 1490–1504.
[17] N. Bougie, N. Watanabe, 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 6: Industry Track), 2025, pp. 43–60.
[18] X. Wanyan, D. Hettiachchi, C. Ma, Z. Xu, J. Chan, Temporal-aware user behaviour simulation with large language models for recommender systems, in: Proceedings of the 34th ACM International Conference on Information and Knowledge Management, 2025, pp. 5335–5339.
[19] L. Shi, C. Ma, W. Liang, X. Diao, W. Ma, S. Vosoughi, Judging the judges: A systematic study of position bias in llm-as-a-judge, in: Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics, 2025, pp. 292–314.
[20] S. Kim, S. Park, H. Kang, W. Kim, J. Seo, Y. In, K. Yoon, C. Park, Self-evolverec: Self-evolving recommender systems with llm-based directional feedback, arXiv preprint arXiv:2602.12612 (2026).
[21] H. Wang, Y. Wu, D. Chang, L. Wei, L. Heldt, Self-evolving recommendation system: End-to-end autonomous model optimization with llm agents, arXiv preprint arXiv:2602.10226 (2026).
[22] C. Lao, F. Pan, G. Ma, H. Li, H. Lin, J. Shi, K. Zhao, K. Gai, M. Zhou, Q. Zhou, et al., Agentx: Towards agent-driven self-iteration of industrial recommender systems, arXiv preprint arXiv:2606.26859 (2026).
[23] J. Mao, L. Li, Y. Gao, Z. Peng, S. He, C. Zhang, S. Qin, S. Khalid, Q. Lin, S. Rajmohan, et al., Stepfly: Agentic troubleshooting guide automation for incident diagnosis, Proceedings of the ACM on Software Engineering 3 (2026) 3070–3092.
[24] P. Pu, L. Chen, R. Hu, A user-centric evaluation framework for recommender systems, in: Proceedings of the fifth ACM conference on Recommender systems, 2011, pp. 157–164.
[25] P. Adamopoulos, A. Tuzhilin, On unexpectedness in recommender systems: Or how to better expect the unexpected, ACM Transactions on Intelligent Systems and Technology (TIST) 5 (2014) 1–32.
[26] J. Carbonell, J. Goldstein, The use of mmr, diversity-based reranking for reordering documents and producing summaries, in: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, 1998, pp. 335–336.
[27] Y. Liu, C. Walder, L. Xie, Determinantal point process likelihoods for sequential recommendation,

in: Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval, 2022, pp. 1653–1663.

[28] M. Wilhelm, A. Ramanathan, A. Bonomo, S. Jain, E. H. Chi, J. Gillenwater, Practical diversified recommendations on youtube with determinantal point processes, in: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, 2018, pp. 2165–2173.

[29] Z. Zhang, S. Liu, Z. Liu, R. Zhong, Q. Cai, X. Zhao, C. Zhang, Q. Liu, P. Jiang, Llm-powered user simulator for recommender system, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 39, 2025, pp. 13339–13347.

[30] L. Chen, Q. Dai, Z. Zhang, X. Feng, M. Zhang, P. Tang, X. Chen, Y. Zhu, Z. Dong, Recusersim: A realistic and diverse user simulator for evaluating conversational recommender systems, in: Companion Proceedings of the ACM on Web Conference 2025, 2025, pp. 133–142.

[31] X. Huang, J. Lian, Y. Lei, J. Yao, D. Lian, X. Xie, Recommender ai agent: Integrating large language models for interactive recommendations, ACM Transactions on Information Systems 43 (2025) 1–33.

[32] J. Zhang, Y. Hou, R. Xie, W. Sun, J. McAuley, W. X. Zhao, L. Lin, J.-R. Wen, Agentcf: Collaborative learning with autonomous language agents for recommender systems, in: Proceedings of the ACM Web Conference 2024, 2024, pp. 3679–3689.

[33] Y. Wang, Z. Jiang, Z. Chen, F. Yang, Y. Zhou, E. Cho, X. Fan, Y. Lu, X. Huang, Y. Yang, Recmind: Large language model powered agent for recommendation, in: Findings of the Association for Computational Linguistics: NAACL 2024, 2024, pp. 4351–4364.

[34] B. Ma, H. Li, Z. Hu, X. Gui, L. Liu, S. Lau, Agentrec: Next-generation llm-powered multi-agent collaborative recommendation with adaptive intelligence, arXiv preprint arXiv:2510.01609 (2025).

[35] J. Park, M. Kim, Madrec: A multi-aspect driven llm agent for explainable and adaptive recommendation, arXiv preprint arXiv:2510.13371 (2025).

[36] X. Lin, Y. Deldjoo, S. Dai, H. Bao, X. Ye, F. Nazary, W. Wang, T. Di Noia, J. Xu, T.-S. Chua, Autonomous information seeking: A roadmap for agentic recommender systems, arXiv preprint arXiv:2607.04433 (2026).