EvoRec: A Unified Self-Evolving Agentic Recommender with Dynamic Reflection Trees

Anonymous Author(s)

Abstract

Conventional recommender systems follow cascaded retrieval, pre-ranking, and ranking pipelines with predefined models and offline-optimized configurations, limiting their ability to adapt to evolving user needs and observed outcomes. Recent agentic recommenders introduce reasoning and tool use but largely focus on isolated stages, lacking both end-to-end decision control and systematic reflection on recommendation failures. To address these limitations, we propose **EvoRec**, a unified agentic recommendation framework with dynamic reflection that integrates end-to-end autonomous recommendation with reflection-driven self-evolution. During recommendation, a **Need Agent** infers structured needs from long-term, short-term, and working memory, a **Planning Agent** dynamically allocates retrieval budgets between Behavior Continuation and Need Satisfaction while selecting retrieval tools and ranking-fusion weights, and an **Acting Executor** deterministically executes the resulting plan and records the complete decision and execution trace. After observing user feedback, a **Dynamic Reflection Tree** localizes failures to need inference, recommendation planning, or prior repair strategies, generates personalized repairs, and verifies their effectiveness through replay. Verified repairs are incorporated into working memory to guide subsequent decisions and further abstracted into reusable strategies shared across users. EvoRec thereby establishes a closed loop of Need Understanding → Planning → Execution → Feedback → Reflection → Repair, enabling continuous self-evolution without updating agent parameters. Experiments on a Kuaishou local-services advertising dataset show that EvoRec improves Hit@50 by 15.7% over the strongest baseline. Controlled ablations validate both adaptive agentic decision-making and hierarchical reflection, with the latter improving Hit@50 by 13.1% for users with prior repair experience. A seven-day online A/B test shows that nearline-deployed EvoRec increases advertising revenue by 4.6% and CTCVR by 3.5%, with a 14.5% revenue gain among high-intent users. These results establish EvoRec as a practical, industrial-scale, unified agentic recommender capable of reflection-driven self-evolution.

CCS Concepts

• **Information systems** → **Recommender systems**; • **Computing methodologies** → *Intelligent agents*; *Multi-agent planning*.

Keywords

Agentic Recommendation, Large Language Models, Dynamic Reflection Trees, Self-Evolving Recommender Systems

1 Introduction

Recommender systems play a central role in modern Web platforms, connecting users with relevant content, products, and services. Traditional recommenders typically adopt multi-stage architectures, in which predefined retrieval strategies generate candidates and ranking models predict user preferences [4, 11, 12, 31]. Despite their success, these systems largely rely on predesigned workflows, limiting their ability to autonomously adapt recommendation decisions to evolving user states and observed outcomes. Recent advances in large language models (LLMs) open new opportunities for recommendation. Representative studies such as OneReason [24] and OneRec-Think [17] incorporate reasoning into recommendation models, enhancing preference understanding and item prediction. However, stronger reasoning within a model does not by itself give the system autonomous control over the entire recommendation process. Leveraging the semantic understanding, reasoning, and tool-use capabilities of LLMs to move recommendation from passive preference prediction toward autonomous decision-making [29] has become a promising research direction.

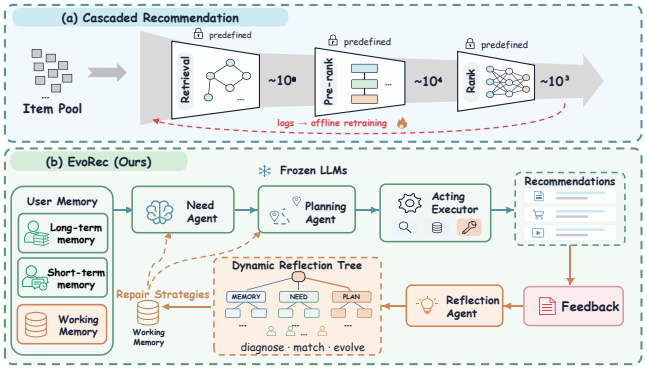


Figure 1: Conventional cascaded recommendation vs. EvoRec. (a) Conventional recommenders rely on predefined pipelines and offline parameter updates. (b) EvoRec integrates autonomous recommendation with reflection-driven self-evolution, using a Dynamic Reflection Tree to diagnose errors and provide personalized repair strategies without updating agent parameters.

Following this direction, recent studies have explored agentic recommendation. InteRecAgent [9] and ToolRec [40] enable agents to invoke conventional recommendation models as tools, while MACRec [26] coordinates multiple specialized agents for recommendation. ChainRec [14] and RecThinker [33] further explore dynamic tool orchestration and active information acquisition. Industrial systems such as RecGPT [30], DREAM [32], and AgentX [13] also bring agents into production recommendation pipelines.

However, two limitations remain. First, these studies mainly address different aspects, ranging from reasoning and tool coordination to pipeline control and algorithm automation, rather than letting agents make actual recommendation decisions end to end, which requires inferring latent needs from dynamic user states, formulating recommendation plans, and coordinating heterogeneous tools accordingly. Second, recommendation feedback is typically used to update model parameters or as interaction memory [15, 30],

without identifying which decision caused an undesirable outcome. Since a missed conversion may stem from an unrecognized need, an inappropriate plan, or a previously learned strategy, and similar error mechanisms recur across users while effective repairs depend on individual users, such feedback is difficult to turn into reusable experience for subsequent recommendation decisions.

To address these limitations, we propose **EvoRec**, a unified agentic recommendation framework with hierarchical reflection. EvoRec formulates recommendation as a decision-making process comprising need understanding, autonomous planning, and plan execution. Specifically, the **Need Agent** infers structured user needs from long- and short-term user memory; the **Planning Agent** allocates the retrieval budget between two complementary paths, *Behavior Continuation* and *Need Satisfaction*, and selects retrieval tools and ranking fusion weights; and the **Acting Executor** executes the plan through retrieval, candidate merging, and ranking fusion to produce recommendations. On this basis, we further introduce a **Dynamic Reflection Tree**. Combining subsequent user feedback with the execution trace, a Reflection Agent localizes errors to need inference, planning, or previously applied repair strategies, matches them hierarchically against existing generalized error patterns, and generates personalized repair strategies, which are provided to the agents as working memory in subsequent rounds. As experience accumulates, verified strategies are abstracted into generalized repair strategies, and the tree reorganizes its hierarchy as new error patterns emerge, forming a closed loop of continuous self-evolution over successive recommendation rounds without updating agent parameters [37].

We evaluate EvoRec on local-services advertising data from Kuaishou, a large-scale short-video platform. EvoRec improves Hit@50 by 15.7% over the strongest baseline, and reflection improves Hit@50 by 13.1% for users who received repair strategies. Controlled ablations further confirm the benefits of adaptive agentic decision-making. Deployed through nearline generation that keeps LLM inference outside the real-time request path, EvoRec increases advertising revenue by 4.6% and CTCVR by 3.5% in a seven-day online A/B test, with a 14.5% revenue gain among high-intent users.

Our main contributions are summarized as follows:

- **Unified agentic recommendation.** We propose EvoRec, which integrates need inference, recommendation planning, and deterministic tool execution into a unified decision-making process, enabling agents to balance historical interests and inferred needs for each user and to coordinate heterogeneous retrieval tools and ranking models.
- **Dynamic Reflection Tree for self-evolution.** We design a Dynamic Reflection Tree that localizes errors to decision stages, verifies personalized repairs by replay, and abstracts verified experience into cross-user strategies while evolving its hierarchy, enabling EvoRec to continuously improve from feedback without updating agent parameters.
- **Industrial deployment and validation.** We deploy EvoRec in a production advertising system through nearline generation, keeping LLM inference off the real-time request path. Offline experiments and an online A/B test show that EvoRec outperforms traditional, generative, and LLM-based or agentic recommenders, and ablation studies verify the respective contributions of adaptive decision-making and hierarchical reflection.

2 Related Work

2.1 Large Language Models for Recommendation

Large language models (LLMs) bring semantic knowledge and richer user modeling to recommender systems. P5 [7] formulates diverse recommendation tasks as conditional text generation. TALLRec [1] and LLaRA [16] adapt pretrained LLMs through recommendation instruction tuning and through the fusion of textual and behavioral representations, respectively. TIGER [20] represents items as semantic identifiers for autoregressive generation. OneRec [5] unifies retrieval and ranking and improves generated recommendations through preference alignment. OpenOneRec [41] observes that next-token prediction alone fits the distribution of items exposed by an existing recommender, and therefore uses reinforcement learning to further align recommendations with user preferences. OneRec-Think [17] incorporates explicit reasoning into recommendation. OneReason [24] combines item semantic alignment, supervised reasoning over user interests, and reinforcement learning to infer latent interests from behavioral evidence. OneLatent [10] internalizes multi-view reasoning supervision in latent tokens, reducing the cost of explicit chain-of-thought generation while improving recommendation accuracy. These and related reasoning-enhanced recommenders [11, 34] primarily optimize recommendation outputs, with reasoning serving item prediction. Output optimization alone does not specify how user needs guide action selection or how execution feedback informs strategy revision. Agentic recommendation explicitly links analysis, planning, execution, and feedback, extending LLM reasoning from item prediction to planning, executing, and revising recommendation decisions.

2.2 Agentic Recommendation

Recent advances in LLMs have accelerated the development of agentic recommendation, enabling recommender systems to incorporate autonomous reasoning, planning, and tool use. Early frameworks such as InteRecAgent [9] and MACRec [26] integrate LLM-based agents with recommendation tools and multi-agent collaboration. Subsequent studies further explore adaptive tool orchestration (ChainRec) [14], active information acquisition (RecThinker) [33], and feedback-driven reasoning optimization (AgenticRec) [15]. AgentDR [28] combines LLM-based agents with conventional recommendation models, while Cao et al. [2] jointly model need prediction and service recommendation. Industrial research has further extended agentic recommendation to large-scale applications. RecGPT [30] explores intent-driven recommendation with dynamically evolving user memory; DREAM [32] introduces an Agentic Meta-Control mechanism based on dual-loop optimization; and AgentX [13] automates recommendation algorithm development through agent-driven experimentation and iterative refinement. Beyond recommendation, self-reflective agents such as Reflexion [23] and ExpeL [39] improve task performance by storing verbal reflections or extracted insights from past trials.

Despite these advances, existing agentic recommenders either improve decisions through parameter updates or maintain feedback as global or user-level memory, without localizing errors to specific decision stages or accumulating reusable repair experience across users. General-purpose reflective agents, in turn, operate on

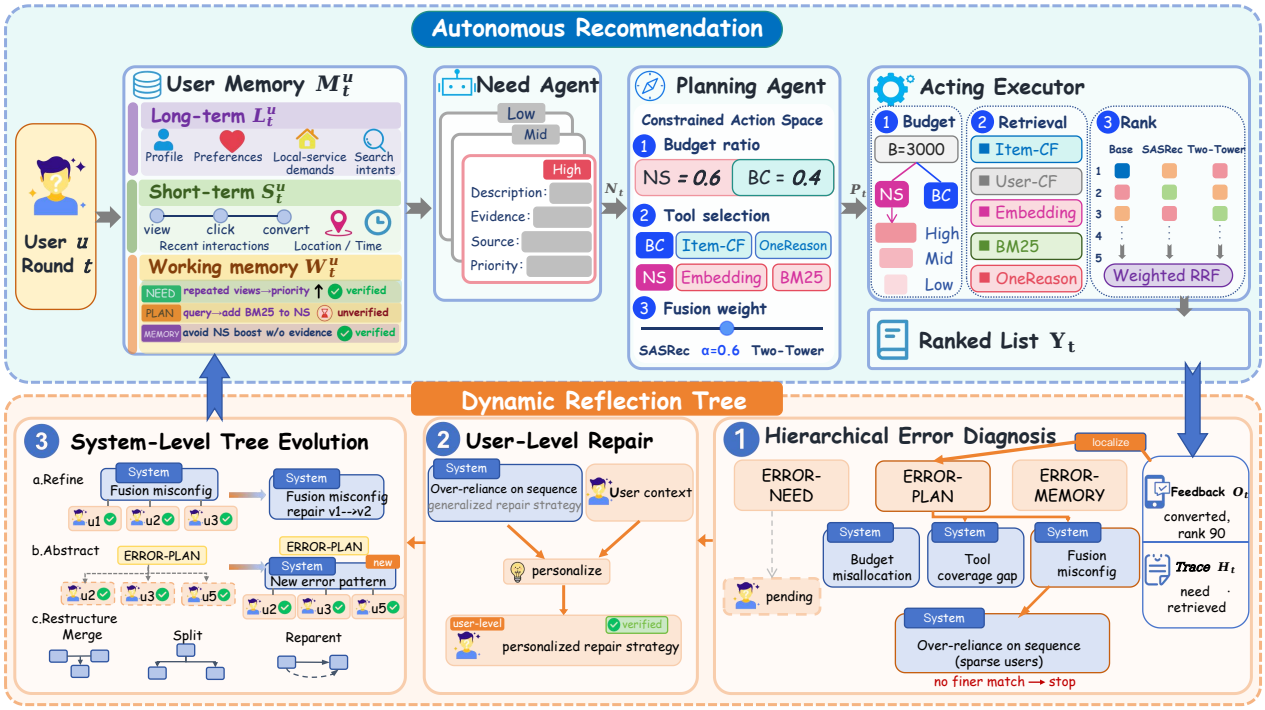


Figure 2: Overview of EvoRec. The upper part illustrates autonomous recommendation through need inference, adaptive planning, and deterministic execution. The lower part presents the Dynamic Reflection Tree, which performs hierarchical error diagnosis, personalized repair, and system-level evolution. Repair strategies are fed back through working memory, enabling continuous self-evolution without parameter updates.

tasks with explicit success signals rather than sparse and delayed recommendation feedback. To bridge this gap, we propose EvoRec, a unified agentic recommendation framework that integrates need inference, planning, and tool-based execution. Furthermore, we introduce a Dynamic Reflection Tree that localizes errors to decision stages and organizes repair experience at the user and system levels, transforming recommendation outcomes into actionable experience that guides future recommendation decisions.

3 Methodology

3.1 Problem Formulation and Overview

Traditional recommender systems rely on predefined retrieval and ranking pipelines to predict user preferences from historical interactions [4]. In contrast, EvoRec formulates recommendation as a multi-stage decision-making process that adapts across recommendation rounds. At round t, user u's memory is $M_t^u = (L_t^u, S_t^u, W_t^u)$, comprising the long-term memory L_t^u, the short-term memory S_t^u, and the working memory W_t^u containing repair strategies derived from reflection. Given M_t^u, the item catalog $\mathcal{I}$, and the recommendation tools $\mathcal{F}$, EvoRec produces a ranked list Y_t:

$$\begin{aligned} N_t &= f_{\text{need}}(M_t^u), \\ P_t &= f_{\text{plan}}(M_t^u, N_t), \\ Y_t &= f_{\text{act}}(P_t, \mathcal{F}, \mathcal{I}), \end{aligned} \tag{1}$$

where the Need Agent infers user needs N_t, the Planning Agent produces a recommendation plan P_t, and the Acting Executor deterministically executes the plan. Agent parameters remain fixed: while L_t^u and S_t^u are updated with new interactions, reflection-derived experiences influence subsequent decisions through W_t^u.

After recommendation, the Reflection Agent uses the decision and execution trace H_t and subsequently observed user feedback O_t to update the Dynamic Reflection Tree:

$$\mathcal{T}_{t+1} = \mathcal{R}(\mathcal{T}_t, H_t, O_t). \tag{2}$$

The user-specific repair strategies derived from the updated tree form the working memory W_{t+1}^u for the next round. EvoRec thus comprises **Autonomous Recommendation** (Section 3.2) and **Reflection-driven Self-Evolution** (Section 3.3), as illustrated in Figure 2.

3.2 Unified Agentic Recommendation Framework

EvoRec organizes autonomous recommendation decision-making into three stages: need inference, planning, and execution, performed by the Need Agent, Planning Agent, and Acting Executor, respectively.

Need Agent: User Need Inference. The long-term memory L_t^u summarizes the user's profile, preferences, local-service demands, and search intents, while the short-term memory S_t^u [25] records recent ad interactions with their action types, timestamps, and content descriptions, together with the spatiotemporal context of the request. Combining these memories with the need-related repair strategies in the working memory W_t^u (Section 3.3), the Need Agent infers user needs $N_t = \{n_1, \ldots, n_m\}$ following Eq. (1). Each need is

represented as a structured record comprising a natural-language description, supporting behavioral evidence, a priority level, and a source type.

Planning Agent: Adaptive Recommendation Planning. Given the inferred needs N_t and the user memory M_t^u, the Planning Agent generates a structured recommendation plan P_t following Eq. (1). To balance historical preferences with inferred user needs, EvoRec adopts two complementary recommendation paths: *Behavior Continuation*, which focuses on historical interests, and *Need Satisfaction*, which targets the inferred user needs. Guided by plan-related repair strategies from W_t^u (Section 3.3), the Planning Agent selects retrieval tools for each path based on the inferred needs and tool capabilities. It also determines the retrieval budget ratio w^{NS} assigned to Need Satisfaction, with the remainder assigned to Behavior Continuation, and the fusion weight α of the sequential ranker SASRec relative to the semantic ranker Two-Tower. These choices are informed by the richness and stability of the user's behavioral history and by the clarity and priority of the inferred needs. Both ratios are selected from $\{0.1, 0.2, \ldots, 0.9\}$ to constrain the planning action space; when no need is inferred, w^{NS} is set to 0 and the entire budget is assigned to Behavior Continuation.

Acting Executor: Deterministic Plan Execution. The Acting Executor carries out P_t through a unified callable interface to retrieval tools and rankers. The retrieval tools cover complementary signals: Item-CF [22] and User-CF [6] exploit item associations and similar-user preferences, embedding-based retrieval and BM25 [21] match items with user interests and inferred needs at the semantic and lexical levels, and OneReason [24] retrieves items through reasoning over user interests. Additional retrieval methods [35, 36] could be integrated as tools through adapters that map their outputs to ranked item candidates. The executor assigns budgets $w^{\mathrm{NS}}B$ and $(1 - w^{\mathrm{NS}})B$ to Need Satisfaction and Behavior Continuation, where B is the total retrieval budget, and further allocates $w^{\mathrm{NS}}B\,\rho(n_k)/\sum_l \rho(n_l)$ to each inferred need n_k, where $\rho(\cdot)$ maps the high, medium, and low priority levels to fixed weights. It then invokes the selected tools accordingly; when the tools return fewer candidates than their budgets, a predefined backfill procedure fills the remaining budget. The retrieved lists are merged by snake-order interleaving, which preserves the internal ranking of each tool, and deduplicated to form the candidate set C_t and its Base ranking Y_t^{base}. The rankers SASRec [12] and Two-Tower [31] then rank C_t, and the three rankings are combined by weighted reciprocal rank fusion (RRF) [3]:

$$s_t(i) = \sum_{m \in \{\mathrm{base,seq,tower}\}} \frac{w_{t,m}}{c + r_{t,m}(i)}, \qquad i \in C_t, \tag{3}$$

where base, seq, and tower denote the Base, SASRec, and Two-Tower rankings, respectively; $r_{t,m}(i)$ is the rank of item i in ranking m, and c is a rank-smoothing constant. The Base ranking receives a fixed weight, while the plan splits a unit weight between the two learned rankers, with $w_{t,\mathrm{seq}} = \alpha$ and $w_{t,\mathrm{tower}} = 1 - \alpha$. Sorting candidates in descending order of $s_t(i)$ yields the final list Y_t. Throughout execution, the executor records the inferred needs, plan, tool calls, budgets, candidate provenance, and final rankings, forming the trace H_t for reflection.

3.3 Dynamic Reflection Tree for Self-Evolution

Each recommendation round yields an outcome, yet final feedback alone neither reveals where a decision went wrong nor makes past experience reusable. We therefore introduce a Dynamic Reflection Tree that organizes reflective experience at the system and user levels. Formally, the tree is denoted as $\mathcal{T}_t = (\mathcal{V}_t^{\mathrm{sys}} \cup \mathcal{V}_t^{\mathrm{usr}}, \mathcal{E}_t)$. Each system-level node $v = (d_v, r_v)$ consists of an error description d_v and a generalized repair strategy r_v; parent–child relations among system-level nodes form an error hierarchy whose top level comprises `ERROR-NEED` for need-inference errors, `ERROR-PLAN` for planning errors, and `ERROR-MEMORY` for errors induced by repair strategies in the working memory. Each user-level node $e = (u_e, d_e, r_e, \sigma_e)$ consists of the user u_e, a description of the diagnostic evidence d_e, a personalized repair strategy r_e, and a verification status $\sigma_e \in \{\textit{unverified}, \textit{verified}, \textit{retracted}\}$, and is linked to its matched system-level nodes. The Reflection Agent diagnoses errors and generates personalized repair strategies, and, as user-level experiences accumulate, continuously refines generalized repair strategies and evolves the tree structure.

Hierarchical Error Diagnosis. Given the trace H_t and subsequent feedback O_t, the Reflection Agent extracts structured diagnostic signals for each feedback-associated item, including evidence for the feedback-indicated need in the long- and short-term memory, whether and with what priority the Need Agent inferred this need, retrieval hits of the item across paths and tools, its rank in Y_t, and the working-memory nodes provided to each agent. Based on these signals, the Reflection Agent localizes potential errors to the three top-level nodes of the tree. If the long- or short-term memory contains evidence for the feedback-indicated need but the Need Agent fails to infer it or assigns an inappropriate priority, the error is localized to `ERROR-NEED`; if such evidence is absent, need inference is not held responsible. If the inferred needs are adequate but the item is not retrieved or falls outside the top-K positions of Y_t, and the Reflection Agent attributes this outcome to the budget ratio, tool selection, or fusion weight based on the per-path and per-tool retrieval results in the trace, the error is localized to `ERROR-PLAN`; such attributions are subsequently checked by replay verification. If re-executing the decision without the working-memory nodes provided to the corresponding agent eliminates a need or plan error, indicating that previously generated repair strategies misled the decision, the error is localized to `ERROR-MEMORY` instead of `ERROR-NEED` or `ERROR-PLAN`. Multiple errors with independent supporting evidence may be identified, while cases satisfying none of these conditions are not attributed.

The Reflection Agent then invokes the Tree Tool to perform hierarchical matching over system-level nodes. The Tree Tool first returns the immediate children of each identified top-level node in a batch, and the Reflection Agent jointly evaluates these candidates against the diagnostic evidence to determine whether they match the current error. For each selected node, the Tree Tool returns its immediate children, allowing the Reflection Agent to determine whether a more specific match exists. This process iterates along the selected branches until no more specific node matches, yielding a set of matched system-level nodes; an error with no matching child under its top-level node is treated as unmatched.

User-Level Repair and Adaptation. For matched errors, the Reflection Agent combines the generalized repair strategies of the matched system-level nodes with the current user context to generate a personalized repair strategy, and records the diagnostic evidence and the strategy as a user-level error node linked to these system-level nodes. For unmatched errors, it generates a personalized repair strategy directly from the diagnostic evidence and user context, and links the resulting user-level node to the corresponding top-level node as a pending node awaiting abstraction. Repair strategies specify the condition, direction, and approximate magnitude of adjustments to need inference and planning rather than which items to recommend, preventing repairs from degenerating into memorization of specific feedback items and allowing them to generalize to subsequent recommendations.

To verify a personalized repair strategy, the Reflection Agent injects it into the original recommendation context, re-executes the pipeline, and checks whether the diagnosed error is mitigated with respect to the observed feedback [42]. The node is marked as *verified* if the error is mitigated and *unverified* otherwise. Node status is continuously updated with subsequent feedback: a strategy implicated in `ERROR-MEMORY` is marked as *retracted*, while an unverified node is promoted to *verified* once its strategy mitigates a recurrence of the diagnosed error in a later round.

Personalized repair strategies are applied to subsequent recommendations through working memory rather than by modifying the execution pipeline [19, 23]. The working memory of user u consists of the user's non-retracted user-level nodes, $W_t^u = \{e \in \mathcal{V}_t^{\text{usr}} \mid u_e = u,\ \sigma_e \neq \textit{retracted}\}$. In subsequent recommendations, the repair strategies in W_t^u are inserted into the prompts of the Need Agent or the Planning Agent according to their top-level category: strategies under `ERROR-NEED` direct the Need Agent to recover missed needs and revise need priorities; strategies under `ERROR-PLAN` direct the Planning Agent to adjust w^{NS}, tool selection, and α, with the adjusted ratios still selected from their grids; and strategies under `ERROR-MEMORY` are provided to the agent whose earlier strategy was retracted, cautioning it against similar adjustments. Consequently, the self-evolution of EvoRec takes place entirely at inference time, without updating the parameters of any agent.

System-Level Tree Evolution. As verified user-level nodes accumulate, the Reflection Agent consolidates them into system-level knowledge. For nodes linked to existing system-level nodes, it extracts patterns shared across users to refine the corresponding generalized repair strategies. For pending nodes, once a sufficient number of verified ones has accumulated under the same top-level node, the Reflection Agent analyzes them collectively and abstracts them into new system-level nodes with generalized repair strategies under that top-level node. As error patterns expand, the Reflection Agent further reorganizes the hierarchy: *Merge* consolidates redundant nodes describing the same error mechanism, *Split* separates a node covering distinct mechanisms into more specific children, and *Reparent* corrects inappropriate parent–child relations. Each structural operation is checked for error-mechanism consistency and hierarchical validity before being applied.

3.4 Industrial Deployment

To integrate EvoRec into the existing recommendation pipeline, we adopt nearline generation and online consumption, keeping LLM inference outside the real-time request path. As shown in Figure 3, the nearline user pipeline performs need inference and resource planning through Need and Plan, respectively, using user memory and recent behavior. Retrieval queries are encoded in advance, and the resulting needs, plans, and query vectors are stored for reuse by Act. Because user needs and item supply evolve at different rates, user computation and item updates run independently, as depicted by the two branches in the lower part of the figure. The item branch preprocesses and encodes new or updated items to maintain the retrieval index. Act allocates retrieval resources according to the stored plan and uses the saved needs and query vectors to retrieve candidates from the current index, followed by deduplication, fusion, and ordering. Provided that the user-side results remain applicable and the query and item encodings remain compatible, an index update can be followed by another Act execution to incorporate newly available items without repeating need inference, planning, or query encoding.

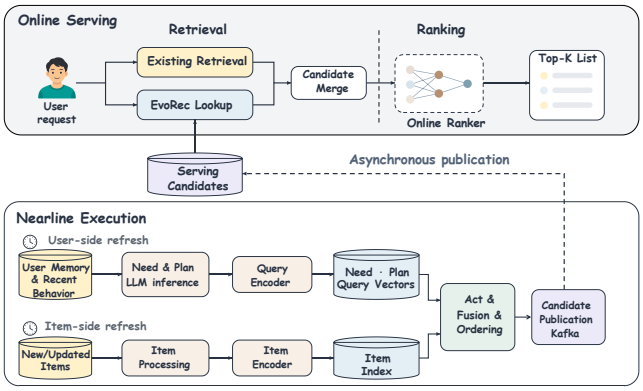


Figure 3: The online deployment architecture of EvoRec.

The resulting candidate lists are published asynchronously through Kafka for each user, connecting nearline execution to online serving in the upper part of Figure 3. Upon request, the serving system retrieves the user's precomputed candidates and merges them with candidates from existing retrieval routes. While Act orders candidates for nearline publication, the online ranker uses the current request context to produce the final recommendation list. LLM decisions therefore influence online recommendations through precomputed candidates, without requiring synchronous need inference, resource planning, or query encoding during request processing.

Reflection is restricted to high-intent users with advertising inquiry or search intent, accounting for 11.3% of active users. Their richer feedback and clearer needs enable more reliable and cost-effective reflection. Reflection runs offline and updates user-level experience upon new clicks or conversions. The working memory is stored for nearline Need and Plan upon their next refresh.

4 Experiments

4.1 Experimental Setup

Dataset. We construct a proprietary dataset from logs of local-services recommendation ads on Kuaishou, a large-scale short-video platform, comprising user and item profiles and four interaction types: impressions, 3-second views, clicks, and conversions. Given all information available by the end of day D, the task is to rank candidate items for each user, with items converted by the user

Table 1: Performance comparison of recommendation methods. Hit@K is reported as a percentage.

Method	Hit@K ↑						NDCG@K ↑			ILD@K ↑	
	@1	@5	@10	@20	@50	@100	@5	@10	@20	@50	@100
SASRec	1.35	4.76	7.13	10.77	16.68	22.06	0.0299	0.0373	0.0462	0.5781	0.5899
UltraGCN	0.91	3.19	4.78	8.04	13.32	17.67	0.0201	0.0250	0.0330	0.6369	0.6441
TIGER	1.32	4.73	7.09	10.86	16.97	22.85	0.0293	0.0371	0.0463	0.4231	0.4322
OneRec	1.42	5.01	7.51	11.61	18.26	24.82	0.0315	0.0393	0.0493	0.5181	0.5499
LLMRank	0.92	3.86	6.41	10.12	17.34	25.72	0.0241	0.0329	0.0428	0.5376	0.5412
MACRec	1.01	4.12	6.79	10.86	18.21	25.72	0.0262	0.0351	0.0461	0.5358	0.5412
ToolRec	1.73	6.21	9.38	13.84	20.47	26.61	0.0398	0.0497	0.0602	0.5287	0.5396
InteRecAgent	1.87	6.53	9.76	14.29	21.16	27.42	0.0415	0.0521	0.0631	0.5302	0.5411
EvoRec w/o Reflection	2.33	7.52	11.37	16.23	23.90	30.38	0.0479	0.0600	0.0720	0.5339	0.5438
EvoRec	**2.37**	**7.66**	**11.61**	**16.62**	**24.48**	**31.06**	**0.0488**	**0.0612**	**0.0736**	0.5364	0.5464

on day $D+1$ serving as ground truth. The evaluation set consists of 28,227 users, all sharing a candidate pool of 355,163 items from the snapshot on day D. The dataset is anonymized, with de-identified user IDs and no personally identifiable information retained.

Evaluation Protocol. EvoRec is evaluated after seven rounds of reflection over the preceding week, starting from a tree with only its three top-level nodes and empty working memory. On each day $d \in \{D-6, \ldots, D\}$, EvoRec generates recommendations using user memory available by the end of day $d-1$, and after the day ends, clicks and conversions from evaluation users drive reflection. On day $D+1$, EvoRec uses the working memory accumulated through day D, and all methods use only information available by the end of day D and are evaluated against conversions observed on day $D+1$. Section 4.5 extends this protocol to subsequent days to examine how reflection evolves. Detailed evaluation metrics are provided in Appendix A.4.

Baselines. We compare EvoRec with three groups of representative baselines: (1) *traditional recommenders*, including SASRec [12] and UltraGCN [18]; (2) *generative recommenders*, including TIGER [20] and OneRec [5]; and (3) *LLM-based and agentic recommenders*, including LLMRank [8], MACRec [26], ToolRec [40], and InteRecAgent [9], all using Qwen3-32B [27]. LLMRank and MACRec, which are designed to rank given candidates, rerank the top-100 candidates of the Base ranking Y_t^{base} produced by EvoRec's retrieval. ToolRec and InteRecAgent, which natively invoke retrieval tools, are given the same user memory, retrieval tools, rankers, and retrieval budget as EvoRec, so that they differ from EvoRec only in how recommendation decisions are made. Detailed baseline descriptions are provided in Appendix A.3.

Implementation Details. The Need Agent and Planning Agent use Qwen3-4B [27], while the Reflection Agent uses Qwen3-32B [27]. The total retrieval budget is set to 3,000 candidates per user. Further implementation details are provided in Appendix A.

4.2 Main Results

As shown in Table 1, EvoRec achieves the best results across all accuracy metrics, improving Hit@10 by 19.0%, Hit@50 by 15.7%, and NDCG@10 by 17.5% over InteRecAgent, the strongest baseline. Even without reflection, EvoRec surpasses InteRecAgent by 12.9% in Hit@50 despite identical tools, rankers, and retrieval budget and a smaller decision model (Qwen3-4B vs. Qwen3-32B), indicating that its gains stem from the decision mechanism rather than tool aggregation; in contrast, LLMRank and MACRec, which only rerank the top-100 Base candidates, fall below the Base ranking itself (Hit@10 of 8.97%, Table 4). Reflection further improves Hit@50 by 2.4%, with gains concentrated on the 18.5% of users who received working memory, on whom Hit@50 increases by 13.1% (Table 5). EvoRec also maintains diversity comparable to OneRec and the LLM-based baselines, while SASRec and UltraGCN attain higher ILD at the cost of substantially lower accuracy.

Table 2: Agentic vs. static decisions with the same available tools, retrieval budget, and ranking models. Hit@K in %. Δ: relative change in Hit@50 w.r.t. EvoRec w/o Reflection.

Variant	Hit@K ↑			NDCG@K ↑		Δ
	@5	@10	@50	@5	@10	
Static Ensemble	6.85	9.42	20.69	0.0428	0.0516	−13.4%
w/o Need Agent	7.08	10.71	22.18	0.0452	0.0566	−7.2%
w/o Planning Agent	7.27	11.03	23.16	0.0465	0.0584	−3.1%
EvoRec w/o Reflection	**7.52**	**11.37**	**23.90**	**0.0479**	**0.0600**	—

4.3 Ablation Study

Agentic vs. Static Decisions. Table 2 tests whether EvoRec's gains stem from adaptive decisions rather than tool aggregation. All variants have access to the same tools, total retrieval budget, and ranking models, with reflection disabled throughout. *w/o Need Agent* removes need inference and the Need Satisfaction path, while the Planning Agent still adapts tool budgets and fusion weights within Behavior Continuation. *w/o Planning Agent* retains the inferred needs and both paths but fixes path weights, tool budgets, and fusion weights. *Static Ensemble* removes both need inference and adaptive planning, using only Behavior Continuation with fixed budgets and fusion weights. Fixed configurations follow the average allocations under EvoRec. Static Ensemble reduces Hit@50 from 23.90% to 20.69% (−13.4%), showing that EvoRec's gains cannot be explained by tool aggregation alone. Removing the Need

Table 3: Ablation of retrieval tools on retrieval-only results. Hit@K in %; Hit@3000 equals the coverage of the candidate set. Δ: relative change in Hit@50 w.r.t. all tools.

Configuration	Hit@K ↑				Δ
	@50	@100	@500	@3000	
All tools	**20.02**	**25.72**	**37.68**	**52.55**	—
w/o Item-CF	17.59	22.05	32.15	46.94	−12.1%
w/o User-CF	18.01	23.85	35.07	51.85	−10.0%
w/o Embedding	19.18	24.47	35.49	48.26	−4.2%
w/o BM25	19.63	25.18	36.77	51.09	−1.9%
w/o OneReason	19.41	24.79	36.58	51.31	−3.0%

Table 4: Comparison of ranking strategies. Base: snake-order merge of retrieved candidates; +: RRF fusion with Base; TT: Two-Tower; fixed/agentic: balanced preset for all users vs. per-user presets. Hit@K in %. Δ: relative change in Hit@50 w.r.t. the agentic fusion.

Strategy	Hit@K ↑			NDCG@K ↑		Δ
	@5	@10	@50	@5	@10	
Base	5.77	8.97	20.02	0.0365	0.0465	−16.2%
SASRec	4.79	7.38	17.13	0.0302	0.0383	−28.3%
Two-Tower	4.16	6.18	13.82	0.0262	0.0325	−42.2%
+ SASRec	6.22	9.85	21.51	0.0389	0.0503	−10.0%
+ SASRec + TT (fixed)	7.26	11.10	23.28	0.0461	0.0582	−2.6%
+ SASRec + TT (agentic)	**7.52**	**11.37**	**23.90**	**0.0479**	**0.0600**	—

Agent reduces Hit@50 by 7.2%, more than at Hit@10 (5.8%), suggesting that inferred needs help retrieve items less accessible from behavioral history, while removing the Planning Agent yields a smaller decline (3.1%), indicating that adaptive allocation remains beneficial even with inferred needs.

Retrieval Tool Collaboration. Table 3 evaluates each retrieval tool without downstream ranking. Removing any tool degrades retrieval at all cutoffs, most notably Item-CF and User-CF (−12.1% and −10.0% in Hit@50), while OneReason and BM25 contribute smaller but non-redundant gains (−3.0% and −1.9%). Removing embedding-based retrieval reduces Hit@50 by 4.2% but Hit@3000 by 8.2%, indicating that it mainly extends coverage at larger depths. These results show that the tools provide complementary signals, supporting the Planning Agent's coordination across them.

Ranking Signal Fusion. Table 4 compares ranking strategies over the same retrieved candidates. Although SASRec and Two-Tower individually underperform Base, fusing them with Base raises Hit@50 from 20.02% to 23.28% under fixed weights, showing that even weaker standalone rankers provide complementary evidence. Letting the Planning Agent set fusion weights per user further increases Hit@50 to 23.90%, indicating that adapting the relative contributions of rankers to individual users improves ranking.

Reflection Mechanism. Table 5 evaluates the Dynamic Reflection Tree on the 5,216 users (18.5% of evaluation users) who received working memory during seven rounds of reflection, with each variant modifying a single component. On these users, EvoRec

Table 5: Ablation of the Dynamic Reflection Tree on 5,216 users (18.5% of evaluation users) who received working memory during seven rounds of reflection, evaluated on day $D+1$; for the remaining users, all variants yield identical recommendations. Hit@K in %. Δ: relative change in Hit@50 w.r.t. EvoRec.

Variant	Hit@K ↑			NDCG@K ↑		Δ
	@5	@10	@50	@5	@10	
EvoRec	**8.38**	**13.25**	**27.42**	**0.0529**	**0.0682**	—
w/o Reflection	7.61	11.94	24.25	0.0481	0.0617	−11.6%
Flat Reflection	7.96	12.48	25.61	0.0502	0.0645	−6.6%
w/o Personalization	7.86	12.35	25.25	0.0496	0.0636	−7.9%
w/o Tree Evolution	8.15	12.90	26.57	0.0516	0.0664	−3.1%
w/o ERROR-MEMORY	8.26	13.04	26.96	0.0522	0.0672	−1.7%
w/o Verification	8.03	12.65	25.96	0.0507	0.0652	−5.3%

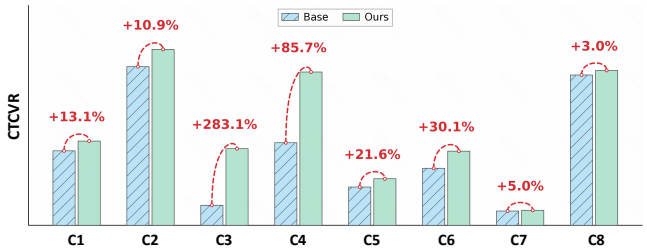


Figure 4: Online CTCVR across eight industry categories, anonymized as C1–C8 and ordered by descending revenue.

improves Hit@50 from 24.25% to 27.42% and NDCG@10 from 0.0617 to 0.0682 over w/o Reflection, corresponding to relative gains of 13.1% and 10.5%, which translate into the 2.4% gain in Hit@50 on all users in Table 1. Regarding the knowledge structure, w/o Personalization applies generalized strategies without user-specific instantiation, reducing Hit@50 by 7.9%, while Flat Reflection generates repair strategies without system-level nodes, causing a 6.6% decline, highlighting the contributions of both shared error knowledge and personalization. Regarding reliability, w/o Verification admits unverified strategies into system-level abstraction, reducing Hit@50 by 5.3%, indicating that verification is necessary for reliable generalization. Disabling tree evolution (while retaining node creation) and retraction via ERROR-MEMORY yield smaller declines of 3.1% and 1.7%, respectively; their cumulative effects are further examined across days in Section 4.5.

4.4 Online Evaluation

We conducted a seven-day online A/B test from August 28 to September 3, 2026, for lead-generation ads in Kuaishou's local-services recommendation system. Users were randomly assigned via user-ID hashing, with the control and treatment groups each accounting for approximately 5% of eligible traffic. Here, high-intent users are defined as those with a non-empty advertising inquiry intent field or search intent field. These fields are derived from users' inquiry messages in advertising contexts and their search queries, respectively. We report only relative changes from the control group to protect commercially sensitive information. Revenue denotes advertising revenue billed by the platform during the experiment.

Table 6: Online A/B results within the deployment scope.

Metric	All users	High-intent users
Impressions	+0.8%	—
Cost	+3.6%	**+5.7%**
Revenue	+4.6%	**+14.5%**
CTCVR	+3.5%	**+5.6%**

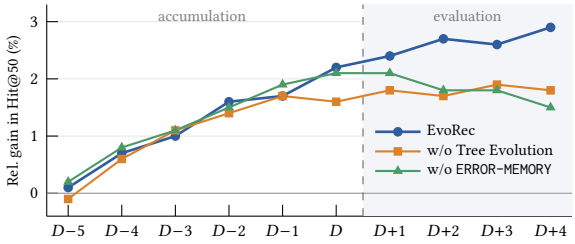


Figure 5: Daily relative Hit@50 improvement over w/o Reflection on all users. The dashed line separates the accumulation and evaluation periods.

As shown in Table 6, EvoRec increases impressions by 0.8%, advertising revenue by 4.6%, and CTCVR by 3.5% within its deployment scope. Among high-intent users, advertising revenue increases by 14.5% and CTCVR improves by 5.6%, compared with a 5.7% increase in cost. These results show improvements in both revenue and CTCVR, with larger relative gains among high-intent users. For high-intent users, inquiry messages or search queries provide explicit evidence of user needs. The larger gains in this cohort suggest that EvoRec's autonomous recommendation process can translate such evidence into effective recommendation actions through need inference, planning, and tool execution.

To examine performance across business categories, we analyze the eight top-level industry categories with the highest revenue, ordered by revenue and anonymized as C1–C8. As shown in Figure 4, C1 achieves a 13.1% relative improvement in CTCVR, while C3 and C4 improve by 283.1% and 85.7%, respectively. These results show positive effects across multiple core business categories. EvoRec is integrated into the existing recommendation pipeline as an independent retrieval channel, without additional training of the downstream pre-ranking and ranking models. The online improvements in C1 and other categories under this deployment setting suggest that candidates retrieved through explicit user-need reasoning can complement existing ranking capabilities.

4.5 Further Analysis

Evolution over Rounds. Since tree evolution and retraction mainly govern the long-term quality of reflective experience, we track their effects across days in Figure 5. During the first days, gains remain close to zero and occasionally slightly negative, since the tree starts with only its top-level nodes and most repair strategies are not yet verified; the three variants therefore perform similarly. As experience accumulates, the gain of EvoRec on all users rises to 2.4% on day $D+1$ and reaches 2.9% by day $D+4$, as more users accumulate working memory; among users with working memory on day $D+1$, the gain increases from 10.2% for those with one repair strategy to 18.5% for those with two or more. In contrast, the gain of w/o Tree Evolution plateaus at around 1.8%, and that of w/o ERROR-MEMORY declines from 2.1% on day D to 1.5% on day $D+4$ as misleading strategies accumulate, indicating that both mechanisms become more important as reflection proceeds.

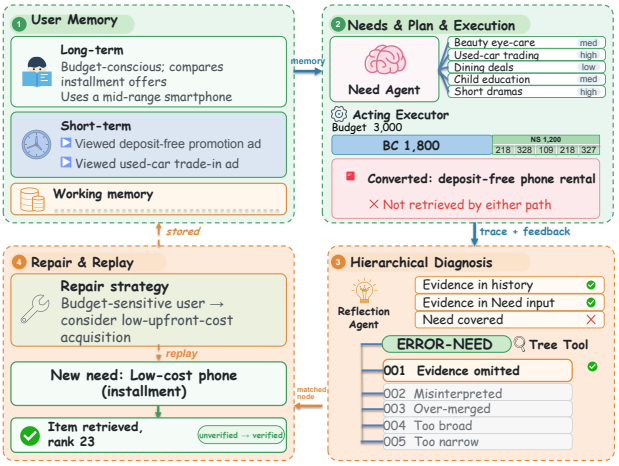


Figure 6: One reflection round (clockwise from top left). BC/NS: Behavior Continuation/Need Satisfaction. User memory is desensitized and abridged.

Case Study. Figure 6 shows one reflection round. Although the user's memory indicates budget sensitivity and a recent view of a deposit-free promotion ad, no inferred need reflects this evidence, and neither path retrieves the subsequently converted deposit-free phone rental. Since the evidence appears in the Need Agent's input, the Reflection Agent localizes the error to ERROR-NEED and matches it to *Evidence omitted*. The repair strategy links budget sensitivity to low-upfront-cost acquisition without naming any item; in replay, the Need Agent combines it with the installment and deposit-free signals in memory to infer a low-cost phone need, the item enters the top-K, and the node is marked as verified.

5 Conclusion

We presented EvoRec, a unified agentic recommendation framework with hierarchical reflection. EvoRec lets agents make recommendation decisions end to end: the Need Agent infers structured user needs from user memory, the Planning Agent allocates the retrieval budget between Behavior Continuation and Need Satisfaction and selects retrieval tools and fusion weights, and the Acting Executor deterministically carries out the plan. The Dynamic Reflection Tree localizes errors to need inference, planning, or previously applied repair strategies, supplies personalized repair strategies through working memory, and abstracts verified experience into generalized error patterns shared across users, enabling self-evolution without updating agent parameters. Experiments on Kuaishou show that adaptive decisions outperform static configurations of the same tools, that reflection brings further gains to users with accumulated experience, and that EvoRec improves advertising revenue and conversion in online deployment. Since reflection currently benefits only users with prior feedback, future work will apply generalized repair strategies to users without working memory and extend EvoRec to other recommendation domains with different tool sets.

References

[1] Keqin Bao, Jizhi Zhang, Yang Zhang, Wenjie Wang, Fuli Feng, and Xiangnan He. 2023. Tallrec: An effective and efficient tuning framework to align large language model with recommendation. In *Proceedings of the 17th ACM conference on recommender systems*. 1007–1014.

[2] Shiteng Cao, Xiaochong Lan, Yuwei Du, Jie Feng, Yinxing Liu, Xinlei Shi, and Yong Li. 2026. Enhancing Local Life Service Recommendation with Agentic Reasoning in Large Language Model. *arXiv preprint arXiv:2604.14051* (2026).

[3] Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*. 758–759.

[4] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. 191–198.

[5] Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. 2025. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965* (2025).

[6] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are we really making much progress? A worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM conference on recommender systems*. 101–109.

[7] Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. 2022. Recommendation as language processing (rlp): A unified pretrain, personalized prompt & predict paradigm (p5). In *Proceedings of the 16th ACM conference on recommender systems*. 299–315.

[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 *European conference on information retrieval*. Springer, 364–381.

[9] Xu Huang, Jianxun Lian, Yuxuan Lei, Jing Yao, Defu Lian, and Xing Xie. 2025. Recommender ai agent: Integrating large language models for interactive recommendations. *ACM Transactions on Information Systems* 43, 4 (2025), 1–33.

[10] Hao Jiang, Peiru Du, Pengfei Yao, Mengting Li, Siyuan Lou, Kuo Cai, Sheng Yu, Qiang Luo, Jian Liang, Ruiming Tang, et al. 2026. WhisperRec: Latent Reasoning for Efficient Foundation Recommendation Models. *arXiv preprint arXiv:2607.26621* (2026).

[11] Hao Jiang, Long Zhang, Guoquan Wang, Sheng Yu, Yang Zeng, Wencong Zeng, Fei Pan, Peng Jiang, and Guorui Zhou. 2026. ReST: A Plug-and-Play Spatially-Constrained Representation Enhancement Framework for Local-Life Recommendation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 4682–4687.

[12] Wang-Cheng Kang and Julian McAuley. 2018. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*. IEEE, 197–206.

[13] Changxin Lao, Fei Pan, Guozhuang Ma, Han Li, Huihuang Lin, Jijun Shi, Kangzhi Zhao, Kun Gai, Mo Zhou, Qinqin Zhou, et al. 2026. Agentx: Towards agent-driven self-iteration of industrial recommender systems. *arXiv preprint arXiv:2606.26859* (2026).

[14] Fuchun Li, Qian Li, Xingyu Gao, Bocheng Pan, Yang Wu, Jun Zhang, Huan Yu, Jie Jiang, Jinsheng Xiao, and Hailong Shi. 2026. ChainRec: An Agentic Recommender Learning to Route Tool Chains for Diverse and Evolving Interests. *arXiv preprint arXiv:2602.10490* (2026).

[15] Tianyi Li, Zixuan Wang, Guidong Lei, Xiaodong Li, and Hui Li. 2026. AgentiCRec: A Recommendation-Oriented Agentic Framework with Progressive Tool-Integrated Reasoning Optimization. *arXiv preprint arXiv:2603.21613* (2026).

[16] Jiayi Liao, Sihang Li, Zhengyi Yang, Jiancan Wu, Yancheng Yuan, Xiang Wang, and Xiangnan He. 2024. Llara: Large language-recommendation assistant. In *Proceedings of the 47th international ACM SIGIR conference on research and development in information retrieval*. 1785–1795.

[17] Zhanyu Liu, Shiyao Wang, Xingmei Wang, Rongzhou Zhang, Jiaxin Deng, Honghui Bao, Jinghao Zhang, Wuchao Li, Pengfei Zheng, Xiangyu Wu, et al. 2025. Onerec-think: In-text reasoning for generative recommendation. *arXiv preprint arXiv:2510.11639* (2025).

[18] Kelong Mao, Jieming Zhu, Xi Xiao, Biao Lu, Zhaowei Wang, and Xiuqiang He. 2021. UltraGCN: ultra simplification of graph convolutional networks for recommendation. In *Proceedings of the 30th ACM international conference on information & knowledge management*. 1253–1262.

[19] Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. 2026. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, Vol. 2026. 94327–94354.

[20] Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. 2023. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems* 36 (2023), 10299–10315.

[21] Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and trends® in information retrieval* 4, 1-2 (2009), 1–174.

[22] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*. 285–295.

[23] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems* 36 (2023), 8634–8652.

[24] OneRec Team, Biao Yang, Boyang Ding, Chenglong Chu, Dunju Zang, Fei Pan, Han Li, Hao Jiang, Honghui Bao, Huanjie Wang, et al. 2026. OneReason Technical Report. *arXiv preprint arXiv:2606.06260* (2026).

[25] Yun Wang, Long Zhang, Jingren Liu, Jiaqi Yan, Zhanjie Zhang, Jiahao Zheng, Xun Yang, Dapeng Wu, Xiangyu Chen, and Xuelong Li. 2025. Episodic memory representation for long-form video understanding. *arXiv preprint arXiv:2508.09486* (2025).

[26] Zhefan Wang, Yuanqing Yu, Wendi Zheng, Weizhi Ma, and Min Zhang. 2024. Macrec: A multi-agent collaboration framework for recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2760–2764.

[27] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388* (2025).

[28] Mingdai Yang, Nurendra Choudhary, Jiangshu Du, Edward W Huang, Philip Yu, Karthik Subbian, and Danai Koutra. 2026. AgentDR: Dynamic Recommendation with Implicit Item-Item Relations via LLM-based Agents. In *Proceedings of the ACM Web Conference 2026*. 6230–6241.

[29] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* (2022).

[30] Chao Yi, Dian Chen, Gaoyang Guo, Jiakai Tang, Jian Wu, Jing Yu, Mao Zhang, Sunhao Dai, Wen Chen, Wenjun Yang, et al. 2025. Recgpt technical report. *arXiv preprint arXiv:2507.22879* (2025).

[31] Xinyang Yi, Ji Yang, Lichan Hong, Derek Zhiyuan Cheng, Lukasz Heldt, Aditee Kumthekar, Zhe Zhao, Li Wei, and Ed Chi. 2019. Sampling-bias-corrected neural modeling for large corpus item recommendations. In *Proceedings of the 13th ACM conference on recommender systems*. 269–277.

[32] Bin Zhang, Bowen Zheng, Chao Yi, Chengyu Lai, Dian Chen, Dimin Wang, Gaoyang Guo, Jialin Zhu, Jian Wu, Jing Yu, et al. 2026. DREAM Technical Report. *arXiv preprint arXiv:2608.09408* (2026).

[33] Haobo Zhang, Yutao Zhu, Kelong Mao, Tianhao Li, and Zhicheng Dou. 2026. RecThinker: An Agentic Framework for Tool-Augmented Reasoning in Recommendation. *arXiv preprint arXiv:2603.09843* (2026).

[34] Long Zhang, Hao Jiang, Sheng Yu, Fei Pan, Peng Jiang, and Kun Gai. 2026. Interpretable Representation via LLM-Driven Generative Disentanglement for Local-Life Service Recommendation. *arXiv preprint arXiv:2607.27944* (2026).

[35] Long Zhang, Peipei Song, Jianfeng Dong, Kun Li, and Xun Yang. 2025. Enhancing partially relevant video retrieval with robust alignment learning. *arXiv preprint arXiv:2509.01383* (2025).

[36] Long Zhang, Peipei Song, Zhangling Duan, Shuo Wang, Xiaojun Chang, and Xun Yang. 2025. Video corpus moment retrieval with query-specific context learning and progressive localization. *IEEE Transactions on Circuits and Systems for Video Technology* 35, 6 (2025), 5659–5670.

[37] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. 2026. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, Vol. 2026. 86069–86100.

[38] Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, et al. 2025. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *arXiv preprint arXiv:2506.05176* (2025).

[39] Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 38. 19632–19642.

[40] 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. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1796–1806.

[41] Guorui Zhou, Jiaxin Deng, Jinghao Zhang, Kuo Cai, Lejian Ren, Qiang Luo, Qianqian Wang, Qigen Hu, Rui Huang, Shiyao Wang, et al. 2025. Onerec technical report. *arXiv preprint arXiv:2506.13695* (2025).

[42] Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, et al. 2025. Where llm agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370* (2025).

A Implementation Details

A.1 LLM Inference

All agents are prompted without fine-tuning to produce structured JSON outputs, with thinking mode disabled for the Need and Planning Agents and enabled for the Reflection Agent, temperature 0.2, and vLLM serving on 8 NVIDIA GPUs. User memory is truncated to the most recent 200 interactions and a long-term profile of at most 1,000 tokens.

A.2 Retrieval, Ranking, and Reflection

EvoRec retrieves $B = 3{,}000$ candidates per user from five tools: Item-CF, User-CF, embedding-based retrieval, BM25, and OneReason. A shared, frozen Qwen3-Embedding-8B [38] encoder produces 4,096-dimensional item representations for both embedding-based retrieval and the Two-Tower ranker; embedding-based retrieval also fills the remaining budget when the selected tools return fewer candidates than their budgets. SASRec is a Transformer with 2 layers, 2 attention heads, dropout of 0.2, and a maximum sequence length of 100, trained with Adam using a learning rate of 10^{-4}, a batch size of 512, and early stopping. Two-Tower takes gated profile embeddings as user inputs and projects both towers into a 64-dimensional space, trained with temperature-scaled sampled softmax ($\tau = 0.05$) using one positive conversion item and 50 uniformly sampled negatives with popularity correction per example. Both rankers are selected by validation NDCG@20. Ranking fusion follows Eq. (3) with $c = 60$ and a fixed Base weight of 1.0; the fixed-fusion variants use $\alpha = 0.5$ for all users. For reflection, the attribution cutoff is $K = 50$, and at most 5 feedback items are diagnosed per user per day, with conversions prioritized over clicks and recent items prioritized within each type. A diagnosed error is considered mitigated if the feedback item enters the top-K after replay, in which case the user-level node is marked as verified; an unverified node is promoted once a recurrence of the same error is mitigated under its strategy in a later round. Abstraction is triggered once 20 verified pending nodes accumulate under the same top-level node, and Merge, Split, and Reparent are applied only after checking that the nodes involved describe a consistent error mechanism and that the resulting hierarchy contains no redundant sibling nodes.

A.3 Baselines

All baselines are trained on data up to day D and evaluated on the same users and candidate pool as EvoRec. SASRec and UltraGCN use their official implementations, with hyperparameters tuned on day D. TIGER and OneRec use beam search with a beam size of 200.

A.4 Evaluation Metrics

We evaluate recommendation accuracy using Hit@K and NDCG@K, and semantic diversity using ILD@K. All metrics are computed over the top-K recommendations, with cutoffs specified in the tables. Accuracy metrics are averaged across evaluation users. ILD@K measures the mean pairwise cosine distance between item embeddings and is averaged over users with at least two valid embeddings. Hit is reported as a percentage. Let $\mathcal{U}$ denote the evaluation users, $\mathcal{G}_u$ the ground-truth items of user u, $\mathcal{L}_u^K$ the top-K list, and $\mathcal{L}_{u,r}$ its item at rank r. All metrics are computed per user and macro-averaged over $\mathcal{U}$:

$$\text{Hit@}K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \mathbb{I}\left[\mathcal{L}_u^K \cap \mathcal{G}_u \neq \varnothing\right], \tag{4}$$

$$\text{NDCG@}K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\sum_{r=1}^{K} \mathbb{I}[\mathcal{L}_{u,r} \in \mathcal{G}_u] / \log_2(r+1)}{\sum_{r=1}^{\min(K, |\mathcal{G}_u|)} 1/\log_2(r+1)}, \tag{5}$$

$$\text{MRR@}K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\mathbb{I}\left[\mathcal{L}_u^K \cap \mathcal{G}_u \neq \varnothing\right]}{r_u^\star}, \tag{6}$$

where $r_u^\star$ is the rank of the first ground-truth item in $\mathcal{L}_u^K$. ILD@K is one minus the mean pairwise cosine similarity between item embeddings in $\mathcal{L}_u^K$, averaged over users with at least two valid embeddings; Avg. Sim.@K equals 1 − ILD@K.

B Additional Experimental Results

B.1 Retrieval Tool Collaboration

Table 7 extends the main-text analysis across retrieval cutoffs and diversity metrics. The full configuration achieves the best results on all reported accuracy metrics. Removing User-CF produces the largest MRR drop, from 0.0419 to 0.0296, whereas removing Item-CF causes the largest Hit@100 drop, from 25.72% to 22.05%, suggesting distinct contributions to early and deeper retrieval. Removing Embedding reduces Hit@50 by 4.2% but Hit@3000 by 8.2%, from 52.55% to 48.26%, highlighting its contribution at larger retrieval depths. The tools also differ in their effects on diversity: removing Item-CF or User-CF lowers ILD@1000 from 0.5199 to 0.5002 and 0.4970, respectively, whereas removing Embedding raises it to 0.5260. Thus, the accuracy gains from tool collaboration do not uniformly translate into greater candidate diversity. These findings support coordinating complementary retrieval tools while accounting for their different effects on retrieval depth and diversity.

B.2 Ranking Signal Fusion

Table 8 extends the main-text analysis by comparing accuracy and diversity across ranking strategies. With the same three ranking signals, agentic fusion selects fusion weights per user rather than applying a shared fixed configuration. It improves all reported accuracy metrics over fixed fusion, raising Hit@50 from 23.28% to 23.90% and NDCG@10 from 0.0582 to 0.0600. These gains support adapting signal fusion to individual users. Meanwhile, ILD@50 and ILD@100 decrease slightly, from 0.5355 to 0.5339 and from 0.5471 to 0.5438, respectively, while SASRec alone achieves the highest diversity. Because the candidate sets remain unchanged, these differences arise from candidate ordering rather than expanded coverage.

C Prompts

Table 9 summarizes the prompt of the Need Agent.

Table 7: Effects of retrieval-tool ablation on accuracy and candidate diversity. Hit@K is reported in percent; other metrics use their original scales. Bold and underlined values indicate the best and second-best results, respectively.

Configuration	Hit@K ↑						MRR ↑	NDCG@K ↑			Diversity@1000	
	@10	@50	@100	@500	@1000	@3000		@5	@10	@20	Avg. sim. ↓	ILD ↑
All tools	**8.97**	**20.02**	**25.72**	**37.68**	**41.30**	**52.55**	**0.0419**	**0.0365**	**0.0466**	**0.0570**	0.4801	0.5199
w/o Item-CF	8.32	17.59	22.05	32.15	36.29	46.94	0.0409	0.0362	0.0450	0.0539	0.4998	0.5002
w/o User-CF	6.82	18.01	23.85	35.07	38.81	51.85	0.0296	0.0234	0.0322	0.0421	0.5030	0.4970
w/o Embedding	8.70	19.18	24.47	35.49	38.62	48.26	0.0410	0.0354	0.0452	0.0550	**0.4740**	**0.5260**
w/o BM25	8.84	19.63	25.18	36.77	40.23	51.09	0.0414	0.0360	0.0459	0.0560	0.4790	0.5210
w/o OneReason	8.66	19.41	24.79	36.58	40.18	51.31	0.0405	0.0352	0.0450	0.0551	0.4835	0.5165

Table 8: Comparison of ranking strategies. Base: snake-order merge; +: RRF fusion with Base; TT: Two-Tower. Fixed uses a balanced preset for all users, whereas agentic selects presets per user. Hit@K is reported in percent; other metrics use their original scales. Bold and underlined values indicate the best and second-best results in each column, respectively.

Strategy	Hit@K ↑						MRR ↑	NDCG@K ↑			ILD@K ↑	
	@1	@5	@10	@20	@50	@100		@5	@10	@20	@50	@100
Base	1.76	5.77	8.97	13.24	20.02	25.72	0.0419	0.0365	0.0465	0.0570	0.5373	0.5412
SASRec	1.40	4.79	7.38	10.86	17.13	23.29	0.0350	0.0302	0.0383	0.0468	**0.5564**	**0.5649**
Two-Tower	1.25	4.16	6.18	9.02	13.82	18.65	0.0299	0.0262	0.0325	0.0394	0.4836	0.4852
+ SASRec	1.81	6.22	9.85	14.41	21.51	28.19	0.0450	0.0389	0.0503	0.0616	0.5392	0.5528
+ SASRec + TT (fixed)	2.02	7.26	11.10	15.85	23.28	29.17	0.0518	0.0461	0.0582	0.0699	0.5355	0.5471
+ SASRec + TT (agentic)	**2.33**	**7.52**	**11.37**	**16.23**	**23.90**	**30.38**	**0.0525**	**0.0479**	**0.0600**	**0.0720**	0.5339	0.5438

Table 9: Prompt of the Need Agent, translated from Chinese and abridged. Braces denote placeholders filled at inference time. The reflection rules are appended only when the user has working memory.

Component	Content
Role	You are the Need Agent of a local-services advertising recommender. Given the long-term profile, recent active behaviors, and current context, output unified needs. Every need must be supported by real input signals; output an empty list when evidence is insufficient.
Source rules	
Recent behavior	Derive a specific service direction from a single ad in the active behavior sequence, without borrowing unrelated ads. Clicks and conversions indicate signal strength only, not completed transactions.
Explicit intent	Derive only from search queries, query sub-intents, and inquiry messages, preserving their semantic anchors; profile conditions may serve only as constraints.
Local-life demand	Derive from explicit local-life demands in messages or the profile, without requiring a matching ad behavior.
Interest preference	Derive from explicit interests that map directly to local services; profile and consumption signals alone cannot form a need.
Cross-domain	Derive only from non-advertising signals (video, live-streaming, and e-commerce interests or consumption), bridging in one hop to a new deliverable service, e.g., gaming → gaming clubs, cooking → local dining, automobiles → test drives or insurance.
Generation and merging	Each need describes one specific, retrievable service direction rather than a broad category. Signals with the same direction are merged with all sources retained; nested or near-duplicate needs are removed, while distinct directions are kept separate. City, price, time, and service form are written as constraints. Output at most 5 needs, preferring clear sources, strong and recent behaviors, and specific directions.
Output	Strict JSON with a list of needs, each containing `need_description`, `evidence_summary`, `constraints`, `priority`, and `source_types`. Priority is high, medium, or low: explicit actions and strong or repeated recent behaviors rank higher, while cross-domain inferences are usually low. Non-compliant items are deleted before output.
Input	Long-term profile: {long_term_memory}; context: {location, season, daypart, holiday}; explicit expressions: {queries, sub-intents, messages}; recent behaviors: {ad ID, timestamp, action, ad content}.
Reflection revision	Additional input: previous needs and reflection memory. Retrieval and conversion facts must not be altered, and diagnostic hypotheses must be re-checked rather than copied. Unaffected needs are kept unchanged; `refine` and `merge` modify only the targeted needs, and `add` creates one new need only when the converted items clearly indicate a specific, independent, retrievable direction, tagged with the source `reflection_feedback`. Generic descriptions such as "shopping experience" cannot form needs. The complete need list is output.