EVODAGREC: A DAG-BASED MULTI-AGENT CODE EVOLUTION FRAMEWORK FOR GENERATIVE RECOMMENDATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Large Language Model (LLM) agents have enabled automated code evolution, offering a promising paradigm for algorithm discovery. However, current recursive self-improvement (RSI) research for sequential recommendation faces a critical dilemma: general-purpose code evolution frameworks possess flexible search mechanisms but lack domain-specific feedback, whereas recommender-specific evolution frameworks utilize rich feedback but rely on linear evolution. More critically, naively integrating them fails when applied to Generative Recommendation (GR). Unlike traditional end-to-end models, GR relies on a module-cascaded training pipeline (i.e., tokenization, pre-training, and alignment). In this paradigm, existing naive evolution methods inevitably suffer from *i) cost blindness:* squandering budgets by ignoring stage-asymmetric retraining overheads; and *ii) short-sighted evaluation:* discarding promising upstream innovations that temporarily regress metrics before downstream adaptation. To bridge these gaps, we propose EvoDAGRec, a multi-agent code evolution framework explicitly tailored for GR. EvoDAGRec organizes code evolution as a Directed Acyclic Graph (DAG) that explicitly maintains multiple evolution branches and merges complementary innovations, and harmonizes an advanced search mechanism with rich GR-specific diagnostic feedback. We further introduce *stage-aware cost planning* to optimize real-world GPU-hour budgets, and *active downstream co-adaptation* to navigate temporary regressions and unlock deferred cross-stage gains. On public Amazon datasets, EvoDAGRec marks the first autonomous evolution of a complete GR pipeline, improving offline metrics by up to 37.4% and boosting Agent4Rec simulated user satisfaction by 19.7% over the strongest GR baseline. Furthermore, in industrial settings, the evolved GR achieves a 12.16% relative improvement in offline Hit@1 and, when deployed as an additional retrieval channel on a large-scale commercial media platform, a statistically significant 0.40% increase in online click-through rate (CTR).

1 INTRODUCTION

With the emergence of Large Language Model (LLM) agents, LLM-driven code evolution has emerged as a promising paradigm, paving the way for Recursive Self-Improvement (Romera-Paredes et al., 2024; Jiang et al., 2025; Chen et al., 2026). This approach gives agents research-like capabilities: they propose hypotheses, modify code, run experiments, and collect feedback, enabling algorithms to self-evolve continuously (Lu et al., 2024; Chan et al., 2025; Huang et al., 2024; Liu et al., 2026; Chen et al., 2021). To accelerate experimental iterations and reduce labor costs for recommendation engineers, an increasing number of studies have introduced this paradigm into the field of sequential recommendation (Wang et al., 2026; Kim et al., 2026; Mu et al., 2026).

While general-purpose code evolution has achieved sophisticated search mechanisms, such as multi-branch exploration and cross-branch merging, through population-based approaches (Romera-Paredes et al., 2024; Lange et al., 2026; Jiang et al., 2025) and tree/graph-based architectures (Jin et al., 2026), they generally lack the domain-specific feedback signals essential for recommender systems. Conversely, existing recommender evolution frameworks invest heavily in feedback quality by utilizing simulated users and diagnostic probes, yet they predominantly rely on primitive search

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

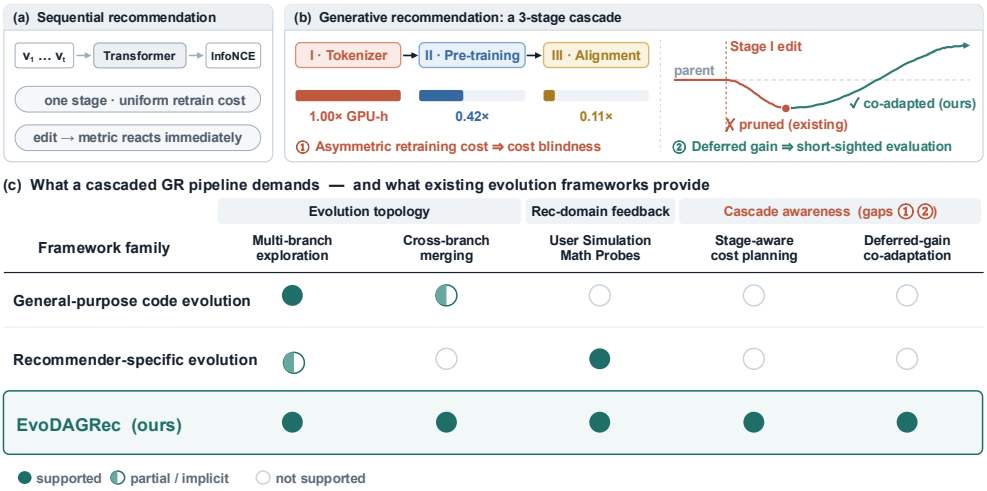

Figure 1: Comparison of recommendation paradigms and evolution framework capabilities. (a) Traditional sequential recommendation relies on an end-to-end architecture characterized by uniform retraining costs and immediate metric feedback. (b) GR utilizes a cascaded pipeline, which introduces stage-asymmetric retraining costs and deferred cross-stage gains, causing existing frameworks to suffer from *cost blindness* and *short-sighted evaluation*. (c) Capability comparison demonstrating that EvoDAGRec uniquely fulfills the specific demands required to evolve cascaded GR pipelines.

mechanisms. For instance, SERS (Wang et al., 2026) uses a simple fan-out/fan-in strategy without branch maintenance; EvoRec performs greedy, round-by-round linear advancement (Mu et al., 2026); and Self-EvolveRec samples a parent node from multiple elites (Kim et al., 2026).

These simple search mechanisms restrict existing recommender evolution frameworks to traditional end-to-end models, where modifications incur a uniform full-training cost and produce immediate metric changes (Kang & McAuley, 2018; Hidasi et al., 2016; Sun et al., 2019). They **fail on Generative Recommendation (GR)**, an increasingly popular paradigm built on a cascaded pipeline (tokenizer, pre-training, and alignment)(Geng et al., 2022; Rajput et al., 2023; Wang et al., 2024; Deng et al., 2025; Pang et al., 2025; Team et al., 2026), because they cannot effectively maintain or merge cross-stage innovations. While explicitly combining advanced general-purpose search mechanisms with domain-specific feedback seems like a natural solution, this naive integration still falls short. General-purpose mechanisms fundamentally rely on two implicit assumptions that a cascaded GR pipeline structurally violates: **1) Cost Blindness vs. Stage-Dependent Costs.** General-purpose search policies evaluate all candidate nodes uniformly, ignoring *how much retraining* an edit requires. In GR, however, retraining costs are highly asymmetric: modifying the upstream Tokenizer forces a full-pipeline retraining, whereas editing the downstream Alignment stage only re-runs the final segment. Being "cost-blind," existing frameworks cannot triage edits based on actual overhead, systematically squandering budgets on expensive upstream explorations. **2) Short-Sighted Evaluation vs. Deferred Gains.** General-purpose frameworks judge modifications by their immediate scores, pruning candidates that initially regress performance. However, because GR stages are tightly coupled, modifying an upstream stage almost always breaks compatibility with the unadapted downstream stage, causing a temporary performance drop. Lacking structural awareness, existing frameworks abandon these upstream innovations, missing the opportunity to actively schedule the necessary downstream adaptations to unlock "deferred" gains(Liu et al., 2025a).

To this end, we present EvoDAGRec, a multi-agent code evolution framework explicitly built for cascaded GR pipelines. EvoDAGRec seamlessly marries advanced graph search mechanisms with rich domain-specific feedback. It represents the search space as a Code Directed Acyclic Graph (Code DAG), driven by a five-stage collaborative Agent workflow (comprising an Operator Selector, Architect-Critic, Coder-Reviewer, Distiller, and Pruner) that digests fine-grained recommendation signals, such as mathematical probes and simulated user (Zhang et al., 2024). Building upon this foundation, EvoDAGRec tightly integrates the multi-agent workflow with two novel mechanisms to overcome the inherent blind spots of cascaded pipelines. First, to address cost blindness, we introduce

Stage-Aware Cost Planning via the Operator Selector. Each DAG node records the exact pipeline stages its edit touches and the corresponding retraining overhead. Empowered by this stage-aware schema, our Operator Selector dynamically balances the expected performance gain against the actual GPU-hour cost, preventing the framework from blindly exhausting budgets on expensive upstream explorations. Second, to overcome short-sighted evaluation, we design **Active Downstream Co-Adaptation.** Instead of rigidly discarding candidates that cause immediate performance drops, EvoDAGRec explicitly retains these seemingly "regressed" upstream branches. When the framework identifies an upstream edit (e.g., Tokenizer) lacking a corresponding downstream adaptation (e.g., Pre-training), the agents deliberately schedule a child node to adapt the downstream stage. This recovers gains from upstream edits that greedy frameworks would discard.

To evaluate our framework fairly, we construct a standardized GR evolution platform equipped with stage-aware feedback and precise GPU-hour accounting, ensuring all compared frameworks evolve from an identical seed codebase. In summary, our core contributions are:

- We identify and formulate the structural mismatch between current code evolution frameworks and cascaded GR pipelines. Specifically, we expose how cost blindness and short-sighted evaluation cause existing methods to systematically misallocate computational budgets and discard promising cross-stage innovations.
- We propose EvoDAGRec, which harmonizes Code DAG search mechanisms with domain-specific feedback via a comprehensive five-stage multi-agent workflow. By further introducing stage-aware cost planning and active co-adaptation, EvoDAGRec enables efficient GR evolution.
- We develop a standardized GR evolution platform. Extensive experiments demonstrate that EvoDAGRec achieves the fully autonomous evolution of a complete GR training pipeline, surpassing both general and recommender-specific evolutionary baselines. It improves offline metrics by up to 37.4% and boosts user satisfaction in the Agent4Rec simulated environment by 19.7% over the strongest GR baseline. Moreover, applying the evolved architecture to **real-world industrial scenarios** yields a 12.16% relative improvement in offline Hit@1 and a statistically significant 0.40% increase in online Click-Through Rate in a 14-day A/B test, demonstrating its practical value.

2 Related Work

LLM-driven Code Evolution. General-purpose code evolution relies on three search paradigms. **Linear inheritance** advances a single thread (e.g., AIDE, MLE-STAR (Nam et al., 2025; Yang et al., 2025; Guo et al., 2024)), often stalling at local optima. **Multi-elite sampling** draws parents from elite archives to preserve diversity (Romera-Paredes et al., 2024; Novikov et al., 2025; Lange et al., 2026; Liu et al., 2025b; 2024; Ye et al., 2024). **Explicit tree search** maintains multiple branches for persistent exploration (e.g., SELA, Arbor (Chi et al., 2024; Jin et al., 2026)); notably, CORAL (Qu et al., 2026) even removes fixed strategies and delegates all search routing to the agent itself. Despite their sophisticated search, these frameworks fail on Generative Recommendation (GR). First, they lack the domain-specific feedback (e.g., mathematical probes, simulated users) essential for GR. Second, facing GR's cascaded pipeline, they lack stage-aware cost planning to handle asymmetric retraining overheads, and cross-stage mechanisms to merge and co-adapt innovations from upstream tokenization down to alignment.

Generative Recommendation (GR). GR replaces classical ID-based models (Hidasi et al., 2016; Kang & McAuley, 2018; Sun et al., 2019) with direct semantic-ID generation (Tay et al., 2022; Rajput et al., 2023). Training GR forms a *cascaded three-stage pipeline*: a Tokenizer quantizes items into hierarchical IDs (Lee et al., 2022); Pre-training fits a sequence-to-sequence generator; and Post-training aligns it with preference signals (Chen et al., 2024; Deng et al., 2025; OneRec Team, 2025; Yu et al., 2026). Current manual research optimizes each stage locally, spanning better tokenizers (Zheng et al., 2024; Wang et al., 2024; Tan et al., 2024; Hou et al., 2025), quantization (Fifty et al., 2025), and architectures (Zhai et al., 2024). Because stage-wise improvements are *complementary*, the global optimum hides in their combinations, a space no single-threaded search can cover. Furthermore, these stages are tightly coupled (Liu et al., 2025a); upstream innovations often trigger temporary metric drops, requiring downstream re-adaptation to unlock their *delayed* value. No prior work autonomously evolves this complete pipeline.

Self-Evolving Recommender Systems. A nascent line applies code evolution to recommenders. SERS (Wang et al., 2026), deployed in YouTube's production system, evolves delta edits against a fixed baseline via a simple fan-out/fan-in strategy. EvoRec (Mu et al., 2026) advances the model architecture greedily, round by round, keeping only the single best variant at each step. Self-EvolveRec (Kim et al., 2026) employs an LLM-based user simulator (Park et al., 2023; Zhang et al., 2024; Wang et al., 2025), and samples a single parent from multiple elites. These frameworks all introduce recommender-specific feedback signals, yet adopt a simple search mechanism that maintains and merges no parallel branches. They fail when evolving a cascaded GR pipeline: its complementary cross-stage gains must be *merged* to reach the global optimum, and its upstream edits pay off only after downstream re-adaptation.

3 PRELIMINARIES

Let $\mathcal{U}$ and $\mathcal{I}$ denote the sets of users and items. In sequential recommendation, given a user's chronological interaction history S_u, the objective is to predict the next item i_{t+1}. Generative Recommendation (GR) reformulates this as a conditional sequence generation problem. It maps items to discrete semantic identifiers (SIDs) and autoregressively generates the SID tokens of i_{t+1} conditioned on S_u. Training a GR model therefore involves three tightly coupled stages: (1) **Stage I: Tokenization.** A tokenizer (e.g., RQ-VAE) discretizes item content into hierarchical semantic IDs. (2) **Stage II: Generative Pre-training.** A sequence-to-sequence Transformer is trained to predict the next item's tokens via hierarchical cross-entropy. (3) **Stage III: Post-training Alignment.** The model is further aligned with desired recommendation objectives (e.g., exact match or prefix similarity) using preference optimization or reinforcement learning. We detail the concrete implementation details and mathematical formulations of these three stages (e.g., RQ-Kmeans quantization, exact baseline objectives, and post-training variants) to Appendix A.3.

4 EVODAGREC

We present **EvoDAGRec**, a multi-agent code evolution framework designed for the cascaded training pipeline of generative recommendation. As shown in Figure 2, EvoDAGRec explicitly represents the evolution history as a Code DAG. This representation enables the framework to preserve multiple promising branches, revisit earlier pipeline stages, and semantically merge complementary improvements discovered along different branches.

At evolution round t, the global search state is encapsulated as $\mathcal{S}^{(t)} = (\mathcal{G}^{(t)}, b^{(t)})$, where $\mathcal{G}^{(t)} = (\mathcal{V}^{(t)}, \mathcal{E}^{(t)})$ is the current Code DAG, and $b^{(t)}$ denotes the remaining evolution budget. Each node $v_i \in \mathcal{V}^{(t)}$ acts as a self-contained experimental record tracking three categories of structured attributes: (1) **Provenance**, including the parent set $\mathcal{P}_i$, applied search operator $o_i \in \{\text{EXPLORE}, \text{MUTATE}, \text{MERGE}\}$, underlying hypothesis $\mathcal{H}_i$, and branch status $\mathcal{Q}_i$; (2) **Artifacts**, covering the incremental code modification Δ_i, executable codebase snapshot $\mathcal{B}_i$, and associated artifacts $\mathcal{A}_i$ (e.g., vocabularies, checkpoints); and (3) **Empirical Execution**, storing the validation feedback $\mathcal{F}_i$, distilled lessons $\mathcal{D}_i$, and consumed experimental cost $\mathcal{C}_i$. The edge set is naturally induced by developmental dependencies, $\mathcal{E}^{(t)} = \{(u, v_i) \mid v_i \in \mathcal{V}^{(t)}, u \in \mathcal{P}_i\}$. Because each new node is derived exclusively from previously evaluated nodes and never modifies its ancestry, $\mathcal{G}^{(t)}$ is strictly guaranteed to be a Directed Acyclic Graph.

4.1 SEARCH OPERATORS

EvoDAGRec transitions the search state by applying one of three search operators to generate a new child node v_c. These operators determine the parent set $\mathcal{P}_c$ and dictate the semantic instructions delivered to the multi-agent system.

Exploration. Exploration branches out from a single parent node ($\mathcal{P}_c = \{v_p\}$) to investigate a fundamentally different direction. Agents formulate a novel hypothesis $\mathcal{H}_c$ distinct from its ancestry, such as introducing a new training objective, changing the user-conditioning architecture, or applying a different post-training algorithm.

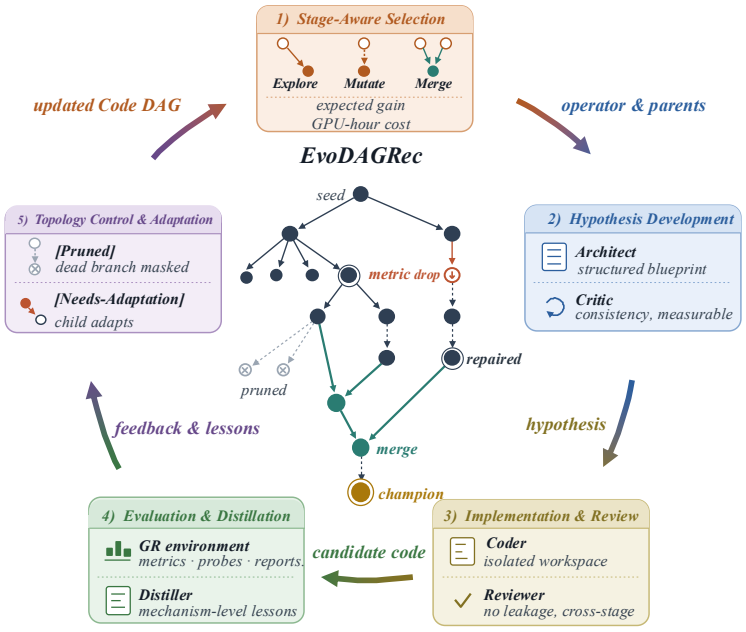

Figure 2: Workflow of EvoDAGRec.

Mutation. Mutation performs exploitation and local refinement on a single parent node ($\mathcal{P}_c = \{v_p\}$). Agents keep the core hypothesis of v_p intact while performing localized optimizations, such as adjusting loss weights, tuning hyperparameters, or modifying sampling strategies.

Merge. The MERGE operator synthesizes complementary improvements discovered across multiple independent branches ($\mathcal{P}_c = \{v_{p_1}, \ldots, v_{p_r}\}$, $r \geq 2$). Unlike single-parent operators, MERGE aggregates a multi-dimensional context. The framework packages the hypotheses, codebases, and validation feedback of all parent nodes and forwards them to the pipeline. Agents are tasked to identify complementary mechanisms, resolve semantic code conflicts or interface dependencies, and integrate them into a unified functional codebase $\mathcal{B}_c$ under a synergized hypothesis $\mathcal{H}_c$. To keep the context tractable, we do not inject full codebase snapshots; instead, each parent is serialized as a unified diff against the lowest common ancestor in the DAG. Each diff is paired with its structured hypothesis $\mathcal{H}$ and distilled lessons $\mathcal{D}$. This representation stays within the context window while preserving the information needed to resolve cross-branch conflicts.

4.2 MULTI-AGENT PIPELINE

Each node expansion is executed by a five-stage pipeline instantiated by seven specialized LLM agents: an Operator Selector, an Architect-Critic pair, a Coder-Reviewer pair, a Distiller, and a Pruner. To ensure rigorous evolution, agents communicate through structured intermediate records (e.g., JSON schemas) rather than unconstrained natural language. The complete workflow is summarized in Algorithm 1 (Appendix A.1).

Stage-Aware Cost Planning via Operator Selector. Instead of simply counting search iterations, EvoDAGRec manages a realistic GPU-hour budget $b^{(t)}$ via empirical cost profiling. Every evaluated node explicitly records the actual time spent on each pipeline stage (e.g., Tokenizer, Pretraining, Alignment). Provided with these historical records, the Operator Selector specifies the `target_stage` and `training_scope` of its intended edit using a structured JSON schema. This allows the system to estimate the expected GPU-hour cost $\mathcal{C}_c$ before execution. By weighing this expected cost against anticipated performance gains, the Operator Selector makes strategic budgeting decisions. For instance, it deliberately postpones expensive upstream edits (which force

full-pipeline retraining) until cheaper downstream architectures fully mature, preventing premature budget exhaustion.

Hypothesis development. Conditioned on the selected operator o_c and parents $\mathcal{P}_c$, the Architect formulates the hypothesis $\mathcal{H}_c$ as a structured blueprint detailing the diagnosed limitation, proposed mechanism, affected stages, expected diagnostic changes, and potential risks. The Critic then evaluates this blueprint against historical lessons and graph ancestry to verify logical consistency and ensure measurable effects. They engage in an iterative refinement loop until the hypothesis passes all structural and semantic checks.

Implementation and review. Guided by $\mathcal{H}_c$, the Coder implements the codebase snapshot $\mathcal{B}_c$ in an isolated workspace initialized from the parents. The Coder and Reviewer execute a multi-round repair loop informed by static analysis and peer review. The Reviewer verifies that the implementation faithfully realizes the blueprint, preserves the evaluation protocol without data leakage, and maintains cross-stage compatibility. For MERGE nodes, it additionally ensures mechanisms inherited from all branches remain functionally active. This loop terminates when the candidate passes all tests or exhausts the repair budget.

Evaluation and lesson distillation. The standardized environment executes compiled candidates and returns a comprehensive feedback object $\mathcal{F}_c$, encapsulating validation metrics, mathematical probes, user diagnostic reports, and consumed computational costs. The Distiller then analyzes the hypothesis $\mathcal{H}_c$, the code delta Δ_c, and the empirical feedback $\mathcal{F}_c$ to synthesize mechanism-level lessons $\mathcal{D}_c$. These lessons explicitly record the tested mechanism, supporting evidence, applicable conditions, and failure attributes.

Search control and active co-adaptation. After evaluating v_c, the Pruner acts as a search router rather than a simple filter. Jointly analyzing quantitative metrics and the Distiller's lessons $\mathcal{D}_c$, the Pruner dynamically dictates branch state transitions via structured outputs. First, for paths with irrecoverable collapses, it assigns a `[Pruned]` state, masking them from the Operator Selector to conserve computation. Second, to address cascaded pipelines, it enables active downstream co-adaptation. When an upstream innovation (e.g., Tokenizer edit) causes a metric regression due to cross-stage incompatibility, the Pruner emits a `[Needs-Downstream-Adaptation]` tag. Subsequent Operator Selector prompts are strictly conditioned on these tags, forcing the system to freeze upstream parameters and mutate the downstream architecture. Consequently, tightly coupled cross-stage mechanisms are co-evolved rather than prematurely discarded.

5 EXPERIMENTS

In this section, we conduct numerical experiments to answer five research questions. **(RQ1)**: Does code-level *DAG* evolution with domain-specific feedback beat both general and recommender-specific evolutionary frameworks? **(RQ2)**: Does *stage-aware cost planning* overcome *cost blindness* and budget for expensive upstream (Tokenizer) innovations? **(RQ3)**: Do *active downstream co-adaptation* and *cross-branch merging* overcome *short-sighted evaluation*? **(RQ4)**: Can the evolved architecture transfer to real-world industrial scenarios? **(RQ5)**: Are user diagnosis and mathematical probes *causally* useful in driving mutations?

Setup and Protocol. All frameworks evolve on a unified platform under a fixed budget $B = 24$ GPU-hours, using Claude CLI (Liu et al., 2026) with all inference routed to GLM-5.2 (GLM-5-Team et al., 2026), starting from an identical TIGER seed codebase (plus post-training variants SDPO, SPRec, ReReGRPO, APAO) on the Amazon Office Products dataset. The final champion architecture evolved on this public dataset is subsequently transferred to a large-scale industrial platform for real-world validation (Section 5). We compare against general frameworks (AlphaEvolve (Novikov et al., 2025), DeepEvolve (Liu et al., 2025b), CORAL (Qu et al., 2026), Arbor (Jin et al., 2026)) and recommender-specific ones (EvoRec (Mu et al., 2026), Self-EvolveRec (Kim et al., 2026)); detailed configurations of the datasets, codebases, and baselines are provided in Appendix A.2, A.3, A.4. The platform exposes three feedback channels: **(i)** offline metrics (Hit@K/NDCG@K), **(ii)** deterministic mathematical probes, and **(iii)** LLM-based user diagnosis (Appendix A.5). Standard

Method	Testset Metrics ↑				Agent4Rec ↑		
	Hit@10	Hit@20	NDCG@10	NDCG@20	View	Sat.	Depth
▼ *Existing Non-Evolutionary Baselines*							
TIGER (Pre-train) (Rajput et al., 2023)	0.0626	0.0944	0.0316	0.0396	0.270	5.45	2.45
SDPO (Chen et al., 2024)	0.0614	0.0946	0.0313	0.0396	0.288	5.75	2.55
SPRec (Gao et al., 2025)	0.0626	0.0975	0.0310	0.0398	0.305	5.95	2.70
ReReGRPO (Tan et al., 2025)	0.0663	0.0991	0.0331	0.0414	0.333	6.20	2.80
APAO (Yu et al., 2026)	0.0697	0.1081	0.0348	0.0444	0.355	6.45	2.95
▼ *General Evolutionary Frameworks (Offline Metrics Only)*							
AlphaEvolve (Novikov et al., 2025)	0.0685	0.1062	0.0338	0.0440	0.345	6.30	2.85
DeepEvolve (Liu et al., 2025b)	0.0705	0.1110	0.0345	0.0465	0.360	6.52	2.95
CORAL (Qu et al., 2026)	0.0785	0.1250	0.0395	0.0520	0.405	7.05	3.35
Arbor (Jin et al., 2026)	0.0792	0.1235	0.0388	0.0535	0.412	6.98	3.42
▼ *Naive Integration (General Search + Domain Diagnostics)*							
CORAL + Diag.	0.0804	0.1276	0.0412	0.0542	0.420	7.18	3.48
Arbor + Diag.	0.0814	0.1272	0.0409	0.0550	0.425	7.12	3.51
▼ *Recommender-side Evolutionary Frameworks (with Diagnostics)*							
EvoRec (Mu et al., 2026)	0.0692	0.1075	0.0340	0.0452	0.352	6.38	2.88
Self-EvolveRec (Kim et al., 2026)	0.0718	0.1165	0.0360	0.0482	0.375	6.68	3.08
EvoDAGRec (Ours)	**0.0869**	**0.1389**	**0.0478**	**0.0609**	**0.466**	**7.72**	**3.70**
Δ vs. Arbor + Diag.	+6.8%	+9.2%	+16.9%	+10.7%	+9.6%	+8.4%	+5.4%
Δ vs. Self-EvolveRec	+21.0%	+19.2%	+32.8%	+26.3%	+24.3%	+15.6%	+20.1%
Δ vs. APAO	**+24.7%**	**+28.5%**	**+37.4%**	**+37.2%**	**+31.3%**	**+19.7%**	**+25.4%**

Table 1: Main results. Evolutionary methods evolve the complete GR pipeline, from upstream Tokenizer to downstream post-training. Δ rows show the relative improvement of EvoDAGRec over Arbor + Diag., Self-EvolveRec, and APAO.

general baselines receive only offline metrics channel (i), whereas recommender-specific frameworks (EvoRec, Self-EvolveRec) and EvoDAGRec access all three channels. To strictly decouple the benefit of diagnostic feedback from evolutionary search topology, we further construct two naive integration baselines, namely **CORAL + Diag.** and **Arbor + Diag.**, which augment the strongest general search frameworks with identical multi-channel feedback. Model selection uses offline validation metrics alone. Both user diagnostic reports and Agent4Rec simulations are driven by DeepSeek-v4-flash(DeepSeek-AI et al., 2026) with a temperature of 0 to ensure determinism and reproducibility. Crucially, evolutionary user diagnosis samples 50 simulated users, whereas post hoc Agent4Rec metrics (Zhang et al., 2024) are evaluated on 100 separate simulated users, keeping the two user cohorts strictly disjoint and non-overlapping to avoid evaluation leakage (Appendix A.6).

Main Results. As shown in Table 1, we make the following observations. **(1) Consistent dominance.** EvoDAGRec leads on every metric, improving over the strongest non-evolutionary baseline APAO by $+24.7\%$/$+28.5\%$ on Hit@10/@20 and $+37.4\%$/$+37.2\%$ on NDCG@10/@20. **(2) Disentangling domain feedback from search topology.** A critical question is whether the gains of EvoDAGRec stem simply from richer feedback or from its DAG-based evolutionary mechanisms. Comparing Arbor with Arbor + Diag. confirms that domain diagnostics indeed help (Hit@10 increases from 0.0792 to 0.0814). However, naively integrating domain feedback into general tree-based or population-based frameworks remains suboptimal: EvoDAGRec still significantly outperforms Arbor + Diag. ($+6.8\%$ Hit@10, $+16.9\%$ NDCG@10) and CORAL + Diag. ($+8.1\%$ Hit@10, $+16.0\%$ NDCG@10). This empirical gap demonstrates that feedback richness alone cannot substitute for structural search capabilities: without DAG-based cross-branch merging and stage-aware cost planning, general search frameworks fail to afford upstream tokenizer retraining or overcome temporary regressions during downstream co-adaptation. Furthermore, EvoDAGRec beats the strongest domain-specific baseline Self-EvolveRec by $+21.0\%$ Hit@10 / $+32.8\%$ NDCG@10 due to its multi-branch exploration. **(3) Offline and simulated experience move together.** Beyond offline metrics, the Agent4Rec leaps (View $+31.3\%$, Satisfaction $+19.7\%$, Depth $+25.4\%$ over APAO) confirm that the discovered architectures yield genuinely better recommendations, answering **RQ1**.

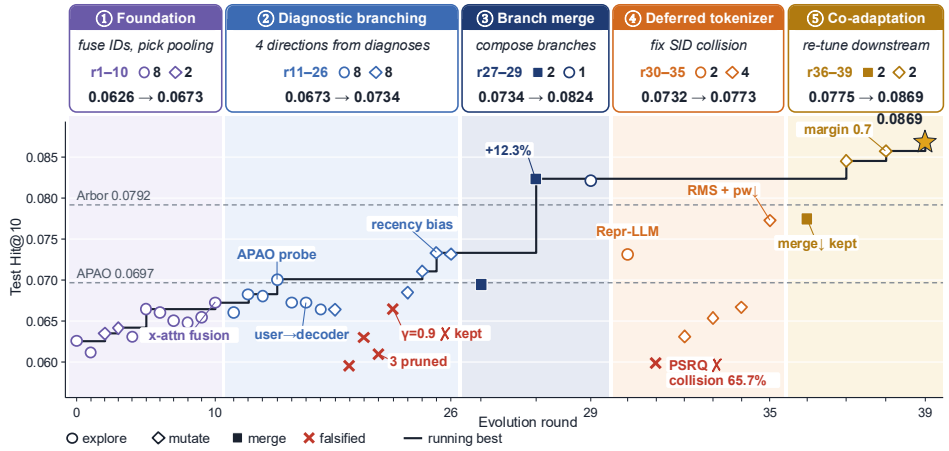

Figure 3: Evolution trajectory of EvoDAGRec (39 rounds, test Hit@10), where candidate models from every round are independently inferred offline on the test set post hoc to trace generalization without leakage. Header cards summarize the five phases (goal, rounds, operator counts, and Hit@10 change); marker shape denotes the mutation operator, red crosses represent falsified nodes, and the line traces the running best score. The x axis is scaled per phase.

Analysis of Evolution Trajectory (RQ1-RQ3). Figure 3 illustrates the 39-round evolution trajectory from the initial TIGER seed (Hit@10 0.0626) to the final architecture (0.0869) across five distinct phases, directly answering RQ1–RQ3. **(Phase 1)** The framework first establishes a robust baseline by injecting both collaborative and semantic IDs into the TIGER encoder, adjusting InfoNCE weights, and grid-searching feature fusion and user pooling methods ($0.0626 \rightarrow 0.0673$). **(Phase 2)** Next, EvoDAGRec opens multiple orthogonal branches based on diagnostic signals. This DAG structure safely explores and deepens parallel mutations (e.g., recency bias, user-conditional decoding) without destabilizing the main trunk, demonstrating a clear advantage over linear frameworks (RQ1). **(Phase 3)** It then pairwise merges these independently matured branches and stacks them with APAO, creating a single-step performance leap to 0.0824 ($+12.3\%$) and validating the effectiveness of cross-branch merging (RQ3). **(Phase 4)** Because modifying the upstream tokenizer requires expensive full-pipeline retraining, EvoDAGRec strategically defers this to avoid budget waste. It tests the new Repr-LLM encoder on an isolated branch, where introducing PSRQ initially causes a collapse to 0.0599. Diagnostics quickly repair it via RMS normalization and parameter sweeps before trunk integration, demonstrating how stage-aware cost planning overcomes cost blindness (RQ2). **(Phase 5)** Finally, when this repaired tokenizer is merged into the main trunk, performance initially regresses to 0.0775. Instead of discarding it via greedy search, EvoDAGRec retains the node and actively co-adapts downstream components by re-tuning the fine-margin loss. This successfully recovers performance, pushing the model to its final peak of 0.0869 after a final merge with the APAO branch, thereby overcoming short-sighted evaluation (RQ3).

Ablation of Search Mechanisms (RQ2 & RQ3). We re-evolve under restricted variants: **A** (greedy: no multi-branch, prunes any regression), **B** (multi-elite but no cross-branch merge), and **C** (full search but without cost planning, uniform budget across stages). As shown in Figure 4(a,b), performance rises monotonically from A to Full EvoDAGRec on both testset and Agent4Rec metrics. **A** stalls at a local optimum, as its short-sighted greedy pruning cannot survive the temporary regressions required for downstream co-adaptation. **B** tolerates these delayed payoffs but remains blind to the compounding gains of merging independent branches, answering RQ3. **C** adds merging but suffers from cost blindness; lacking stage-aware planning, it exhausts the budget uniformly and cannot afford the expensive upstream tokenizer retraining. Full EvoDAGRec finally triages its budget properly to conquer the upstream stage, unlocking the terminal peak and answering RQ2.

Real-World Industrial Deployment (RQ4). We deploy the champion GR architecture to commercial media platform with over 10 million daily active users. The production system follows a standard retrieval-pre-ranking-ranking cascade. **Offline Transferability.** We train the champion architecture and the strongest baseline APAO from scratch using 10 days of industrial logs (>10

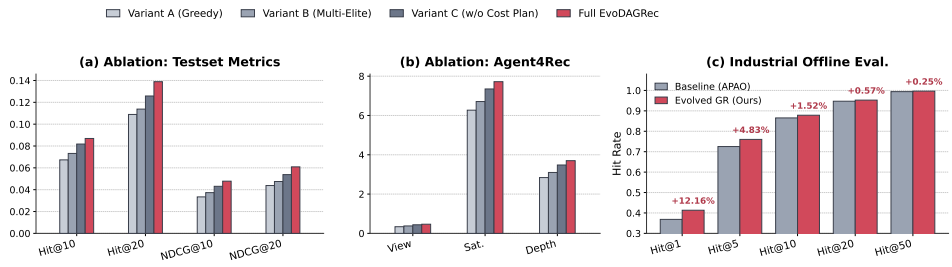

Figure 4: Ablation of search mechanisms and industrial evaluation. (a) Testset metrics and (b) Agent4Rec metrics under restricted search variants (A: greedy; B: multi-elite; C: w/o cost planning) versus Full EvoDAGRec, where performance rises monotonically from A to Full. (c) Real-world industrial offline evaluation of the evolved GR pipeline over the APAO baseline across Hit@K.

million interactions). As shown in Figure 4(c), our evolved model strictly improves over APAO across all metrics($+12.16\%$ Hit@1, $+4.83\%$ Hit@5, $+0.25\%$ Hit@50). **Online A/B Testing.** We subsequently deployed the evolved model as a novel retrieval channel for a 14-day A/B test on 5% of live traffic (serving >1M users). To ensure strict statistical rigor, we applied CUPED variance reduction and controlled the false discovery rate via tiered Benjamini-Hochberg correction ($q \leq 0.05$). The online results reveal a significant overall system CTR increase of $+0.40\%$ ($p = 0.019$). Furthermore, the new channel contributes over 10% of unique exposures (relevant items completely missed by existing channels) and achieves a channel-specific CTR $\approx$13% higher than the system average.

Validation of Diagnostic Signals (RQ5). To answer whether diagnostic signals are causally useful in driving mutations, we highlight three critical turning points where offline metrics alone offered no actionable direction. **(i) User Diagnosis:** When performance plateaued, simulated user feedback explicitly reported that recommendations were "outdated," motivating the agent to inject an exponential recency bias. When an initial weak prior (γ=0.9) showed no offline improvement, diagnostics confirmed it was absorbed by learnable weights, prompting the agent to apply a stronger signal (γ=0.5) that successfully broke the plateau. **(ii) Mathematical Probes:** Standard metrics only indicate overall correctness, but mathematical probes isolated a specific "same coarse, wrong fine" error mode. This directly drove the agent to design a targeted within-coarse hard-negative contrastive loss that random mutation could not easily discover. **(iii) Distribution Probes:** When the new PSRQ tokenizer caused a catastrophic performance drop, an offline-only search would have discarded it. However, distribution probes pinpointed a semantic ID collision spike (to 65.7%) due to a k-means scale imbalance. This causal insight guided the agent to apply an RMS normalization repair, dropping collisions back to 1.0% and unlocking the tokenizer's potential. Ultimately, while offline metrics only flag failures, these diagnostics provide the exact causal mechanisms required for the agent to execute targeted repairs.

6 Conclusion

In this paper, we introduce EvoDAGRec, a multi-agent code evolution framework tailored for Generative Recommendation (GR). We observe that naively integrating existing evolution frameworks fails on GR's cascaded pipelines due to two fundamental bottlenecks: cost blindness and short-sighted evaluation. To bridge this gap, EvoDAGRec harmonizes an advanced Code Directed Acyclic Graph (Code DAG) search mechanism with rich diagnostic feedback. Building upon this foundation, it introduces *stage-aware cost planning* to explicitly triage edits based on actual GPU overhead, and *active downstream co-adaptation* to safely navigate temporary regressions and unlock deferred gains. Extensive evaluations on a standardized GR platform demonstrate that EvoDAGRec achieves state-of-the-art performance, outperforming the leading non-evolutionary baseline (APAO) by up to 37.4% in offline metrics and 19.7% in simulated user satisfaction. In addition, successful migration to real-world industrial scenarios yields a 12.16% relative improvement in offline Hit@1 and a statistically significant 0.40% gain in online Click-Through Rate. Moving forward, we plan to directly evolve GR pipelines within complex industrial scenarios. Moreover, since Large Language Model (LLM) training inherently involves cascaded stages, we envision our framework can be naturally extended to optimize multi-stage LLM pipelines.

AI USE STATEMENT

In this work, we used generative AI tools only for tasks with recommended disclosure. Specifically, (1) we used a large language model to assist in reading and summarizing prior baseline works to aid our own understanding of the related literature, and (2) we used a large language model to polish the writing (grammar, phrasing, and clarity) of the manuscript. We did not use generative AI tools for any tasks with required disclosure: the research ideas, the proposed EvoDAGRec framework and its algorithms, the experimental design, and all code and experiments were conceived and implemented by the authors. Data generation and other required-disclosure tasks are not applicable to this work.

We have reviewed all AI-assisted work. All AI-generated summaries of baseline methods were cross-checked by the authors against the original papers to ensure factual accuracy, and all AI-polished text was manually verified by the authors to preserve the intended technical meaning. We take full responsibility for the final content of this work, including all text, claims, and artifacts.

REFERENCES

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Mingguang Chen, Licheng Wang, and Bo Qu. Recursive self-improvement in ai: From bounded self-refinement to autonomous research loops, 2026. URL https://arxiv.org/abs/2607.07663.

Yuxin Chen, Junfei Tan, An Zhang, Zhengyi Yang, Leheng Sheng, Enzhi Zhang, Xiang Wang, and Tat-Seng Chua. On softmax direct preference optimization for recommendation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. SELA: Tree-search enhanced LLM agents for automated machine learning. *arXiv preprint arXiv:2410.17238*, 2024.

DeepSeek-AI, Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chengyu Hou, Chenhao Xu, Chenze Shao, Chong Ruan, Conner Sun, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Donghao Li, Dongjie Ji, Erhang Li, Fang Wei, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoling Zhang, Haoming Luo, Haoran Wei, Haotian Yuan, Haowei Zhang, Haowen Luo, Haoyu Chen, Haozhe Ji, Hengqing Zhang, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J Yang, JQ Zhu, Jia Luo, Jia Song, Jia Yu, Jialiang Huang, Jialu Cai, Jian Liang, Jiangting Zhou, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jieyu Yang, Jin Chen, Jin Yan, Jingchang Chen, Jingli Zhou, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jingzi Zhou, Jinhua Zhu, Jiping Yu, Joseph Sun, Jun Ran, Junguang Jiang, Junjie Qiu, Junlong Li, Junmin Zheng, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Leyi Xia, Li Zhang, Liang Zhao, Lihua Guo, Lingxiao Luo, Linwang Ma, Linyan Zhu, Litong Wang,

Liyu Cai, Liyue Zhang, Longhao Chen, MS Di, MY Xu, Max Mei, Miaojun Wang, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingming Li, Mingxu Zhou, Minmin Han, Ning Wang, Panpan Huang, Panpan Wang, Peixin Cong, Peiyi Wang, Peng Zhang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qiwei Jiang, Rui Tian, Ruifan Xu, Ruijie Lu, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqian Chen, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, Ruyi Chen, SH Liu, Shanghao Lu, Shangmian Sun, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoqing Wu, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Shuying Yu, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tun Wang, W Zhang, WL Xiao, Wangding Zeng, Wei An, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao, Wenjun Gao, Wenkai Yang, Wenlve Huang, Wenqing Hou, Wentao Zhang, Wenting Ma, Xi Gao, Xiang He, Xiangwen Wang, Xianzu Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaowen Sun, Xiaoxiang Wang, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyu Zhang, Xu Chen, Xuanyu Wang, Xuecheng Su, Xueyin Chen, Xuheng Lin, Xuwei Fu, YC Yan, YQ Wang, YW Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Xu, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Shao, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Yiling Ma, Ying He, Ying Tang, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, YuKun Li, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuehan Yang, Yuer Xu, Yuhan Wu, Yuhao Meng, Yuheng Zou, Yukun Zha, Yunfan Xiong, Yupeng Chen, Yuping Lin, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yuting Yan, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhen Huang, ZF Wu, Zehao Wang, Zehua Zhao, Zehui Ren, Zekai Zhang, Zhangli Sha, Zhe Fu, Zhe Ju, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zheren Gao, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhixuan Chen, Zhiyu Wu, Zhizhou Ren, Zhongyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihua Qu, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Ziyi Wan, Zizheng Pan, and Zongqing Yao. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026. URL https://arxiv.org/abs/2606.19348.

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

Christopher Fifty, Ronald G. Junkins, Dennis Duan, Aniketh Iyengar, Jerry W. Liu, Ehsan Amid, Sebastian Thrun, and Christopher Ré. Restructuring vector quantization with the rotation trick. In *Proceedings of the 13th International Conference on Learning Representations (ICLR)*, 2025.

Chongming Gao, Ruijun Chen, Shuai Yuan, Kexin Huang, Yuanqing Yu, and Xiangnan He. Sprec: Self-play to debias llm-based recommendation. In *Proceedings of the ACM on Web Conference 2025*, pp. 5075–5084, 2025.

Shijie Geng, Shuchang Liu, Zuohui Fu, Yingqiang Ge, and Yongfeng Zhang. Recommendation as language processing (RLP): A unified pretrain, personalized prompt & predict paradigm (P5). In *Proceedings of the 16th ACM Conference on Recommender Systems (RecSys)*, 2022.

GLM-5-Team, :, Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, Chenzheng Zhu, Congfeng Yin, Cunxiang Wang, Gengzheng Pan, Hao Zeng, Haoke Zhang, Haoran Wang, Huilong Chen, Jiajie Zhang, Jian Jiao, Jiaqi Guo, Jingsen Wang, Jingzhao Du, Jinzhu Wu, Kedong Wang, Lei Li, Lin Fan, Lucen Zhong, Mingdao Liu, Mingming Zhao, Pengfan Du, Qian Dong, Rui Lu, Shuang-Li, Shulin Cao, Song Liu, Ting Jiang, Xiaodong Chen, Xiaohan Zhang, Xuancheng Huang, Xuezhen Dong, Yabo Xu, Yao Wei, Yifan An, Yilin Niu, Yitong Zhu, Yuanhao Wen, Yukuo Cen, Yushi Bai, Zhongpei Qiao, Zihan Wang, Zikang Wang, Zilin Zhu, Ziqiang Liu, Zixuan Li, Bojie Wang, Bosi Wen, Can Huang, Changpeng Cai, Chao Yu, Chen Li, Chengwei Hu, Chenhui Zhang, Dan Zhang, Daoyan Lin, Dayong Yang, Di Wang, Ding Ai, Erle Zhu, Fangzhou Yi, Feiyu Chen, Guohong Wen, Hailong Sun, Haisha Zhao, Haiyi Hu, Hanchen Zhang, Hanrui Liu, Hanyu Zhang, Hao Peng, Hao Tai, Haobo Zhang, He Liu, Hongwei Wang, Hongxi Yan, Hongyu Ge, Huan Liu, Huanpeng Chu,

Jia'ni Zhao, Jiachen Wang, Jiajing Zhao, Jiamin Ren, Jiapeng Wang, Jiaxin Zhang, Jiayi Gui, Jiayue Zhao, Jijie Li, Jing An, Jing Li, Jingwei Yuan, Jinhua Du, Jinxin Liu, Junkai Zhi, Junwen Duan, Kaiyue Zhou, Kangjian Wei, Ke Wang, Keyun Luo, Laiqiang Zhang, Leigang Sha, Liang Xu, Lindong Wu, Lintao Ding, Lu Chen, Minghao Li, Nianyi Lin, Pan Ta, Qiang Zou, Rongjun Song, Ruiqi Yang, Shangqing Tu, Shangtong Yang, Shaoxiang Wu, Shengyan Zhang, Shijie Li, Shuang Li, Shuyi Fan, Wei Qin, Wei Tian, Weining Zhang, Wenbo Yu, Wenjie Liang, Xiang Kuang, Xiangmeng Cheng, Xiangyang Li, Xiaoquan Yan, Xiaowei Hu, Xiaoying Ling, Xing Fan, Xingye Xia, Xinyuan Zhang, Xinze Zhang, Xirui Pan, Xu Zou, Xunkai Zhang, Yadi Liu, Yandong Wu, Yanfu Li, Yidong Wang, Yifan Zhu, Yijun Tan, Yilin Zhou, Yiming Pan, Ying Zhang, Yinpei Su, Yipeng Geng, Yong Yan, Yonglin Tan, Yuean Bi, Yuhan Shen, Yuhao Yang, Yujiang Li, Yunan Liu, Yunqing Wang, Yuntao Li, Yurong Wu, Yutao Zhang, Yuxi Duan, Yuxuan Zhang, Zezhen Liu, Zhengtao Jiang, Zhenhe Yan, Zheyu Zhang, Zhixiang Wei, Zhuo Chen, Zhuoer Feng, Zijun Yao, Ziwei Chai, Ziyuan Wang, Zuzhou Zhang, Bin Xu, Minlie Huang, Hongning Wang, Juanzi Li, Yuxiao Dong, and Jie Tang. Glm-5: from vibe coding to agentic engineering, 2026. URL https://arxiv.org/abs/2602.15763.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-Agent: Automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *Proceedings of the 4th International Conference on Learning Representations (ICLR)*, 2016.

Yupeng Hou, Jianmo Ni, Zhankui He, Noveen Sachdeva, Wang-Cheng Kang, Ed H. Chi, Julian McAuley, and Derek Zhiyuan Cheng. ActionPiece: Contextually tokenizing action sequences for generative recommendation. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

Jiajie Jin, Yuyang Hu, Kai Qiu, Qi Dai, Chong Luo, Guanting Dong, Xiaoxi Li, Tong Zhao, Xiaolong Ma, Gongrui Zhang, et al. Toward generalist autonomous research via hypothesis-tree refinement. *arXiv preprint arXiv:2606.11926*, 2026.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *Proceedings of the IEEE International Conference on Data Mining (ICDM)*, 2018.

Sein Kim, Sangwu Park, Hongseok Kang, Wonjoong Kim, Jimin Seo, Yeonjun In, Kanghoon Yoon, Hyunsik Jeon, and Chanyoung Park. Self-EvolveRec: Self-evolving recommender systems with LLM-based directional feedback. *arXiv preprint arXiv:2602.12612*, 2026.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *Proceedings of the 14th International Conference on Learning Representations (ICLR)*, 2026.

Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

Enze Liu, Bowen Zheng, Cheng Ling, Lantao Hu, Han Li, and Wayne Xin Zhao. Generative recommender with end-to-end learnable item tokenization. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025a.

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

Gang Liu, Yihan Zhu, Jie Chen, and Meng Jiang. Scientific algorithm discovery by augmenting AlphaEvolve with deep research. *arXiv preprint arXiv:2510.06056*, 2025b.

Jiacheng Liu, Xiaohan Zhao, Xinyi Shang, and Zhiqiang Shen. Dive into claude code: The design space of today's and future ai agent systems, 2026. URL https://arxiv.org/abs/2604.14228.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI Scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Lingyu Mu, Hao Deng, Haibo Xing, Jinxin Hu, Yu Zhang, and Xiaoyi Zeng. EvoRec: Self-evolving agentic recommender systems. *arXiv preprint arXiv:2606.28368*, 2026.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Ö. Arık, and Tomas Pfister. MLE-STAR: Machine learning engineering agent via search and targeted refinement. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OneRec Team. OneRec-V2 technical report. *arXiv preprint arXiv:2508.20900*, 2025.

Yunsheng Pang, Zijian Liu, Yudong Li, Shaojie Zhu, Zijian Luo, Chenyun Yu, Sikai Wu, Shichen Shen, Cong Xu, Bin Wang, et al. Higr: Industrial-scale hierarchical generative slate recommendation framework in tencent. *arXiv preprint arXiv:2512.24787*, 2025.

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*, 2023.

Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Q. Tran, Jonah Samost, Maciej Kula, Ed H. Chi, and Maheswaran Sathiamoorthy. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, 2019.

Junfei Tan, Yuxin Chen, An Zhang, Junguang Jiang, Bin Liu, Ziru Xu, Han Zhu, Jian Xu, Bo Zheng, and Xiang Wang. Reinforced preference optimization for recommendation. *arXiv preprint arXiv:2510.12211*, 2025.

Juntao Tan, Shuyuan Xu, Wenyue Hua, Yingqiang Ge, Zelong Li, and Yongfeng Zhang. Towards LLM-RecSys alignment with textual ID learning. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024.

Yi Tay, Vinh Q. Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, et al. Transformer memory as a differentiable search index. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

TGR Team, Lei Cheng, Haonan Hu, Beibei Kong, Yudong Li, Zang Li, Yunsheng Pang, Hongyang Su, Jianchao Tu, Yunlong Wang, Bing Wen, Junzhang Zhu, Shaojie Zhu, and Chengxiang Zhuo. Tgr: Advancing industrial recommendation from generative-paradigm ranking toward unified generation and reasoning, 2026. URL https://arxiv.org/abs/2609.00986.

Haochen Wang, Yi Wu, Daryl Chang, Li Wei, and Lukasz Heldt. Self-evolving recommendation system: End-to-end autonomous model optimization with LLM agents. In *Proceedings of the 20th ACM Conference on Recommender Systems (RecSys)*, 2026. arXiv:2602.10226.

Lei Wang, Jingsen Zhang, Hao Yang, Zhi-Yuan Chen, Jiakai Tang, Zeyu Zhang, Xu Chen, Yankai Lin, Ruihua Song, Wayne Xin Zhao, et al. User behavior simulation with large language model-based agents. *ACM Transactions on Information Systems*, 43(2):1–37, 2025.

Wenjie Wang, Honghui Bao, Xinyu Lin, Jizhi Zhang, Yongqi Li, Fuli Feng, See-Kiong Ng, and Tat-Seng Chua. Learnable item tokenization for generative recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management (CIKM)*, 2024.

Xu Yang, Xiao Yang, Shikai Fang, Yifei Zhang, Jian Wang, Bowen Xian, Yuge Zhang, Weiqing Liu, Jiang Bian, et al. R&D-Agent: An LLM-agent framework towards autonomous data science. *arXiv preprint arXiv:2505.14738*, 2025.

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

Yuanqing Yu, Yifan Wang, Weizhi Ma, Zhiqiang Guo, and Min Zhang. Apao: Bridging the training-inference gap in generative recommendation via adaptive prefix-aware optimization. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 6338–6349. ACM, August 2026. doi: 10.1145/3770855.3818052. URL http://dx.doi.org/10.1145/3770855.3818052.

Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, et al. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, 2024.

An Zhang, Yuxin Chen, Leheng Sheng, Xiang Wang, and Tat-Seng Chua. On generative agents in recommendation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024.

Bowen Zheng, Yupeng Hou, Hongyu Lu, Yu Chen, Wayne Xin Zhao, Ming Chen, and Ji-Rong Wen. Adapting large language models by integrating collaborative semantics for recommendation. In *Proceedings of the 40th IEEE International Conference on Data Engineering (ICDE)*, 2024.

A Appendix

A.1 Workflow

Algorithm 1 expands a single evolution round of EvoDAGRec. We present the round in the same five stages as Section 4.2, and use this section to make precise what the main text states only qualitatively: the records exchanged between the agents, the accounting of the budget $b^{(t)}$, and the transition of the branch status $\mathcal{Q}$.

Agent records. Each agent is invoked as a single call constrained to one structured record, which the harness parses directly; a response that cannot be parsed is discarded and the call is repeated rather than repaired, so that no free-form residue enters the node attributes and every intermediate decision remains machine-checkable. The Operator Selector consumes the serialised graph $\mathcal{G}^{(t)}$ together with the remaining budget $b^{(t)}$ and returns the operator o_c and parent set $\mathcal{P}_c$; the Architect consumes these and the ancestral lessons and returns the hypothesis $\mathcal{H}_c$; the Coder and Reviewer consume $\mathcal{H}_c$ and return the codebase snapshot $\mathcal{B}_c$ with the delta Δ_c; the Distiller consumes $\mathcal{H}_c$, Δ_c, and the feedback $\mathcal{F}_c$ and returns the lesson $\mathcal{D}_c$; and the Pruner consumes $\mathcal{F}_c$, $\mathcal{D}_c$, o_c, and the touched stage and returns the branch status $\mathcal{Q}_c$. Two of these choices are deliberate. The Critic is instructed to reject a blueprint only when it violates a hard invariant, cannot yield an executable forward pass, is a no-op, or re-proposes a direction already falsified without addressing the recorded lesson $\mathcal{D}_i$; a rejected blueprint costs a full round of agent calls, whereas an accepted but unpromising one costs only a single evaluation, so the asymmetry favours acceptance. The Distiller is given the Coder's workspace together with $(\mathcal{H}_c, \Delta_c, \mathcal{F}_c)$ and inspects Δ_c before writing $\mathcal{D}_c$, so that a lesson attributes the outcome to a mechanism (a specific layer, objective, or initialization) instead of restating $\mathcal{H}_c$.

Budget accounting. A round is charged the GPU-hours measured by the environment, $\mathcal{C}_c$, and not the expected cost estimated from the `target_stage` and `training_scope` attributes. That estimate only informs the Operator Selector's strategic decision; the decrement $b^{(t+1)} \leftarrow b^{(t)} - \mathcal{C}_c$ always uses the measured value, which is recovered stage by stage and stored on the node, so that the cost profile available to later rounds reflects the true stage-asymmetric overheads of the pipeline. A round that fails review or is interrupted before yielding a usable candidate is charged τ_{fail}, the GPU-hours spent before the failure, so that repeatedly proposing unexecutable candidates cannot be used to explore at no cost. Token consumption is logged separately and does not draw on $b^{(t)}$.

Branch-status transition. A node that passes review and yields a valid checkpoint, whether or not it improves on its parent, keeps its default status and remains selectable, since an immediate regression on an upstream edit is precisely the signal that downstream co-adaptation is required. Only the two tagged statuses alter the search: `[Pruned]` is assigned on an irrecoverable collapse (unexecutable or protocol-violating code, a broken item↔code mapping, or a metric collapse with no diagnostic route) and removes the subtree permanently, masking it from the Operator Selector; `[Needs-Downstream-Adaptation]` is assigned when a metric regression is caused by an upstream (tokenization or pre-training) edit and conditions the subsequent Operator Selector prompts so that upstream parameters are frozen and the child node must mutate the downstream stage to restore cross-stage compatibility.

Algorithm 1 EvoDAGRec

Require: Seed codebase $\mathcal{B}_0$, maximum GPU-hour budget B, standardized environment $\mathcal{E}$, failure penalty τ_{fail}
Ensure: The best candidate node selected by validation performance

Initialize root node v_0 containing $\mathcal{B}_0$
Initialize global graph $\mathcal{G}^{(0)} = (\{v_0\}, \emptyset)$
$t \leftarrow 0, b^{(0)} \leftarrow B$
while $b^{(t)} > 0$ **do**
  # 1. Stage-Aware Operator and Parent Selection
  $(o_c, \mathcal{P}_c) \leftarrow \textsc{Selector}(\text{Serialize}(\mathcal{G}^{(t)}), b^{(t)})$
  # 2. Hypothesis Development
  $\mathcal{H}_c \leftarrow \textsc{ArchitectCritic}(\mathcal{P}_c, o_c, \mathcal{G}^{(t)})$
  # 3. Implementation and Review
  $\mathcal{B}_c, \Delta_c \leftarrow \textsc{CoderReviewer}(\mathcal{H}_c, \mathcal{P}_c, o_c)$
  # 4. Evaluation and Lesson Distillation
  if $\mathcal{B}_c$ passes review and is executable **then**
    $\mathcal{F}_c, \text{tokens}_t, \mathcal{C}_c \leftarrow \textsc{Evaluate}(\mathcal{B}_c, \mathcal{E})$ # $\mathcal{C}_c$ is the actual GPU-hour cost
  else
    $\mathcal{F}_c \leftarrow$ Failure feedback, $\text{tokens}_t \leftarrow$ Debugging cost, $\mathcal{C}_c \leftarrow \tau_{\text{fail}}$ # measured GPU-hours spent before failure
  end if
  $\textsc{Log}(\text{tokens}_t)$ # Track LLM token consumption
  $\mathcal{D}_c \leftarrow \textsc{Distill}(\mathcal{H}_c, \Delta_c, \mathcal{F}_c)$
  # 5. Search Control and Active Co-Adaptation
  Construct child node v_c with attributes $(\mathcal{P}_c, o_c, \mathcal{H}_c, \Delta_c, \mathcal{B}_c, \mathcal{F}_c, \mathcal{D}_c, \mathcal{C}_c, \dots)$
  Add v_c and derivation edges to form intermediate graph $\mathcal{G}'$
  $state, stage \leftarrow \textsc{Pruner}(\mathcal{F}_c, \mathcal{D}_c, o_c)$ # $stage$: touched pipeline stage read from $\mathcal{H}_c$
  if $state ==$ DEAD_END **then**
    Mark v_c as `[Pruned]`
  else if $state ==$ METRIC_DROP AND stage $==$ UPSTREAM **then**
    Mark v_c as `[Needs-Downstream-Adaptation]`
  end if
  $\mathcal{G}^{(t+1)} \leftarrow \text{UpdateGraph}(\mathcal{G}', v_c)$
  Update remaining budget: $b^{(t+1)} \leftarrow b^{(t)} - \mathcal{C}_c$
  $t \leftarrow t + 1$
end while
return $\arg\max_{v_i \in \mathcal{V}} \text{ValidationMetric}(\mathcal{F}_i)$

A.2 DATASETS

In this section, we introduce the open-source Amazon Office dataset used for evolution, as well as the industrial dataset used to verify whether the evolved GR architecture is effective in real-world industrial scenarios.

Amazon Office Dataset. Following the evaluation protocol of the initial codebase (Yu et al., 2026) and prior generative recommendation studies (Rajput et al., 2023), we conduct all evolution experiments on the *Office Products* category of the public Amazon Product Review corpus[1]. We **directly adopt the preprocessed data released by** Yu et al. (2026). For completeness, we restate the protocol under which the released data were produced: each review is treated as one implicit positive interaction, and every user's interactions are sorted chronologically into a single behaviour sequence. Five-core filtering then iteratively discards users and items with fewer than five interactions. Each history is truncated to the most recent 20 items. Finally, the data is split with the standard leave-one-out strategy. For every user, the chronologically last interaction forms the test instance, the second-to-last forms the validation instance, and the rest are used for training. All evolutionary frameworks therefore optimise and are selected on exactly the same validation split, and the test split is touched only once, for the final checkpoint selected within the budget B. Item side information (title, brand, category, price and description) is concatenated into a textual description and used exclusively by the item tokenizer. The statistics of the released data are summarised in Table 2. Let $\mathcal{U}$, $\mathcal{I}$ and $\mathcal{D}$ denote the user set, the item set and the interaction set, respectively. We report the average sequence length $\overline{L} = |\mathcal{D}|/|\mathcal{U}|$ and the interaction density $\rho = |\mathcal{D}| \big/ (|\mathcal{U}| \times |\mathcal{I}|)$.

Table 2: Statistics of the Amazon Office dataset. We directly use the preprocessed data released by Yu et al. (2026), obtained under five-core filtering and leave-one-out splitting.

Dataset	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	$\overline{L}$	ρ
Amazon Office	4,905	2,420	53,258	10.86	0.45%

We deliberately select Office as the *evolution* dataset rather than a larger benchmark. This is because a compact corpus keeps a single train/validate cycle below one GPU-hour, so that the unified budget of $B = 24$ GPU-hours admits a meaningful number of evolution rounds for *every* framework.

Real-world Industrial Dataset. The industrial data are collected from a large-scale commercial media platform with over 10 million daily active users. The production model is trained continuously on streaming logs. For a controlled offline comparison, all models are trained on the same 10 consecutive days of logs (over 10 million interactions on more than 300 thousand items) and evaluated on the following day. A positive interaction is a click. The item text fed to the tokenizer concatenates textual metadata (e.g., title, category, and tags) with a caption of the item image generated by a multimodal LLM. User histories are truncated to the 500 most recent interactions. The platform serves recommendations through a retrieval–pre-ranking–ranking cascade, in which the evolved GR model is added as one retrieval channel for the online A/B test (Section 5). Exact user and item counts and absolute CTR values are withheld for confidentiality.

A.3 GR CODEBASE

In this section, we introduce the codebase provided for the evolutionary framework.

All frameworks evolve upon an identical seed codebase, which is built directly on the officially released APAO implementation (Yu et al., 2026) together with its preprocessed data and its original tokenizer configuration. The codebase is organised into four decoupled modules: (i) item tokenization, (ii) backbone definition, (iii) the pre-training objective, and (iv) a pluggable post-training objective. All four share one data loader and one beam-search evaluator. What is held invariant across all frameworks is the *environment*: the preprocessed interaction data, the train/validation/test split, the evaluation protocol and the budget B. Everything above the data layer, including the tokenization module, is exposed as mutable code and may be modified by a candidate during evolution.

[1] https://jmcauley.ucsd.edu/data/amazon/links.html

Item Tokenization. Every item is represented by a sequence of discrete semantic tokens rather than a single atomic ID. The seed codebase ships with the tokenizer configuration of Yu et al. (2026): a semantic embedding is first extracted for each item by feeding its concatenated textual metadata into LLAMA-3.1-8B-INSTRUCT and taking the final-layer representation; a **4-level RQ-K-means** quantizer then residually quantizes each embedding into $T = 4$ codes (y^1, y^2, y^3, y^4), one per level. Each level owns an independent codebook, and a de-duplication suffix is appended whenever two items collide on all four levels, guaranteeing an injective item↔code mapping. A user history $x = \{i_1, \ldots, i_{n-1}\}$ is thus flattened into a token sequence of length $4(n-1)$, and predicting the next item amounts to autoregressively generating its 4 semantic tokens. Note that the tokenization module is *not* frozen: a candidate is free to re-fit or redesign the quantizer (e.g. changing the number of levels, the codebook size or the quantization algorithm) as part of its proposed modification, in which case the downstream backbone is retrained accordingly. Only the upstream interaction data and the splits are fixed. The seed baselines reported in the main text all use the original configuration above.

Backbone. The seed backbone is TIGER (Rajput et al., 2023), a T5-style Transformer *encoder-decoder* operating entirely in the semantic-token space. The encoder consumes the flattened history tokens and the decoder autoregressively emits the 4 codes of the target item; at inference time decoding is performed with constrained beam search over the valid code trie. Both the encoder and the decoder consist of 4 layers with 6 self-attention heads of head dimension 64, ReLU activations in all feed-forward blocks, and a learnable token embedding dimension of 128, yielding roughly 0.01B parameters in total. The full architectural and optimisation configuration is listed in Table 3; all values are inherited from Yu et al. (2026) without modification.

Table 3: Architecture and optimisation configuration of the seed GR codebase. These values are shared by the pre-training baseline and all post-training variants, and constitute the identical starting point from which every evolutionary framework departs.

Group	Configuration	Value
Tokenizer	Semantic encoder	LLAMA-3.1-8B-INSTRUCT
	Quantizer	RQ-Kmeans
	# Levels T	4
	Collision handling	de-duplication suffix
Backbone (TIGER)	Architecture	Transformer encoder-decoder
	# Encoder / decoder layers	4 / 4
	# Attention heads	6
	Head dimension	64
	Token embedding dimension	128
	Activation	ReLU
Optimisation	Optimizer	AdamW
	Peak learning rate	5e-4
	LR schedule	cosine decay, 1% warmup
	Batch size	1024 (128×8 grad. accum.)
	Max epochs	200
	Early-stopping patience	20
	Negative sampling	uniform random
	# Negatives per positive	100
Inference	Decoding	constrained beam search
	Beam size K	20
	Max history length	20 items

Pre-training Baseline. **TIGER (Pre-train)** (Rajput et al., 2023) optimises the standard token-level cross-entropy (maximum-likelihood) objective under teacher forcing,

$$\mathcal{L}_{\mathrm{CE}} = -\sum_{t=1}^{T} \log \pi_\theta\big(y_+^t \mid x,\, y_+^{<t}\big)\,, \tag{1}$$

where y_+ denotes the ground-truth code sequence. Every post-training variant below is initialised from this converged checkpoint, so that the observed differences are attributable purely to the post-training objective.

Post-training Variants. The seed codebase ships with four post-training objectives, which together span the mainstream design space of preference- and reinforcement-based alignment for generative recommendation. They constitute the algorithmic prior that the evolutionary frameworks may recombine, ablate or extend.

- **SDPO** (Chen et al., 2024) generalises Direct Preference Optimization to a multi-negative, softmax-weighted ranking loss. Given a positive item $e_{i,+}$ and a sampled negative set $\mathcal{N}$, it optimises

$$\mathcal{L}_{\mathrm{SDPO}} = -\log\sigma\left(-\log\sum_{j\in\mathcal{N}}\exp\Big(\beta\log\tfrac{\pi_\theta(e_{j,-}|x)}{\pi_{\mathrm{ref}}(e_{j,-}|x)} - \beta\log\tfrac{\pi_\theta(e_{i,+}|x)}{\pi_{\mathrm{ref}}(e_{i,+}|x)}\Big)\right), \tag{2}$$

  where π_{ref} is the frozen pre-trained reference policy. Crucially, the preference signal is defined at the *complete sequence* level, i.e. on the cumulative likelihood of the whole item, with a single summation over $\mathcal{N}$.
- **SPRec** (Gao et al., 2025) is a self-play framework designed to suppress the over-recommendation / filter-bubble bias that plain DPO induces. Each self-play iteration performs one SFT step on the offline interactions followed by one DPO step in which the *model's own predictions from the previous iteration* serve as negatives. Because the negative distribution is the current policy's own output, the DPO loss is implicitly re-weighted by the model logits, adaptively penalising items the model already over-exposes. No extra data or manual intervention is needed.
- **ReReGRPO** (Tan et al., 2025) casts post-training as reinforcement learning with verifiable rewards on top of GRPO. Two domain-specific adaptations make GRPO viable in the constrained item space: (i) candidates are sampled by *constrained beam search* rather than i.i.d. sampling, which yields a group of valid, distinct and on-policy hard negatives in a single pass; and (ii) the sparse binary rule reward (exact match with the target) is augmented with a *ranking reward* that penalises a generated negative proportionally to its probability rank, $r(e_k) = R_{\mathrm{rule}}(e_k) + \lambda R_{\mathrm{rank}}(e_k)$. Advantages are normalised within the group, $\hat{A}_k = (r_k - \mathrm{mean}(\mathbf{r}))/\mathrm{std}(\mathbf{r})$, and a KL penalty towards the reference policy stabilises the update.
- **APAO** (Yu et al., 2026) attacks the training-inference inconsistency directly by moving supervision from the sequence level to every intermediate *prefix*. Defining the cumulative score of a length-m prefix as $s^m = \sum_{t\le m}\log\pi_\theta(y^t \mid x, y^{<t})$, the pairwise variant optimises

$$\mathcal{L}_{\mathrm{pair}} = \mathcal{L}_{\mathrm{CE}} + \beta\sum_{m=1}^{T} w_m\Big[-\log\sigma\big(-\log\sum_{j\in\mathcal{N}_m}\exp(s^m_{j,-} - s^m_{i,+})\big)\Big], \tag{3}$$

  while the pointwise variant replaces the bracketed term with the negative length-normalised prefix log-likelihood. The prefix weights $\mathbf{w} \in \Delta^{T-1}$ are *not* fixed but follow an exponentiated-gradient update, $w_m \leftarrow w_m \exp(\eta\mathcal{L}_m)$ followed by re-normalisation, which realises *adaptive worst-prefix optimisation*: the step whose prefix is currently most likely to fall out of the beam receives the largest weight.

A.4 Evolution Baselines

We compare EvoDAGRec against six evolutionary frameworks, grouped into *general evolutionary frameworks* (domain-agnostic) and *recommender-specific frameworks* (built for recommendation). Every baseline is instantiated on the same seed codebase, executed by the same coding agent (Claude CLI), and constrained by the same budget $B = 24$ GPU-hours described in the main text, so that the differences reported stem from the search strategy itself rather than from the environment.

General evolutionary frameworks.

- **AlphaEvolve** (Novikov et al., 2025) orchestrates an autonomous LLM-driven evolutionary pipeline that iteratively evolves and optimizes entire algorithmic codebases by directly editing and testing code against evaluator feedback. It generalizes the FunSearch/Eureka line of LLM-driven code evolution from single functions to whole codebases, but is guided primarily by scalar metrics and carries no recommendation-specific inductive bias.
- **DeepEvolve** (Liu et al., 2025b) extends the evolutionary loop with retrieval-augmented generation (RAG): it retrieves external scientific knowledge (e.g., academic papers from arXiv) to systematically inform hypothesis generation and codebase refinement, grounding the evolution in prior work rather than relying on the model's internal knowledge alone.
- **CORAL** (Qu et al., 2026) is a multi-agent framework for open-ended discovery that replaces rigid control flow with long-running agents that explore, reflect, and collaborate through a shared persistent memory and asynchronous execution.
- **Arbor** (Jin et al., 2026) is a general autonomous-research framework built on Hypothesis Tree Refinement : a persistent tree links hypotheses, artifacts, evidence, and distilled insights across time. A long-lived coordinator manages the global strategy over the tree, while short-lived executors implement and test individual hypotheses in isolated worktrees; returned results are propagated back into the tree as reusable lessons that refine the search frontier.

Recommender-specific frameworks.

- **EvoRec** (Mu et al., 2026) is a multi-agent framework that co-evolves both the recommendation model and the optimization methodology driving it. Four agents (an Orchestrator, a Research Agent, a Code Agent, and a Skill Evolver) run a dual-track loop: the Research and Code agents iterate the model each round, while the Skill Evolver periodically distills reusable methodology from a persistent memory of past experiments into a self-evolving skill library. It is domain-aware but is guided primarily by offline metrics, without the directional diagnosis.
- **Self-EvolveRec** (Kim et al., 2026) introduces *directional feedback* for recommender evolution, pairing a *User Simulator* that emits natural-language critiques with a *Model Diagnosis Tool* that probes internal structure (e.g., embedding collapse, ranking margin) and interprets the numbers into a structured diagnosis. A *Diagnosis Tool Co-Evolution* mechanism keeps the diagnosis tool compatible with the evolving model, and an evolutionary archive retains the codebase history.

A.5 Multi-dimension Diagnostics

During evolution the environment produces, for every executed candidate, a structured feedback bundle assembled from three complementary diagnostic channels, following the directional-feedback design of Self-EvolveRec (Kim et al., 2026) (which pairs a qualitative *User Simulator* with a quantitative *Model Diagnosis Tool*) and specialising it to the 4-layer semantic-ID (SID) generative recommender. The three channels are: *(i)* offline validation metrics, giving a macro-level view of ranking quality; *(ii)* deterministic mathematical probes, exposing internal failure modes that aggregate metrics cannot express; and *(iii)* qualitative user diagnosis, surfacing behaviour-level deficiencies in natural language. While offline metrics in Channel *(i)* are available to every framework, diagnostic feedback in Channels *(ii)* and *(iii)* is provided exclusively to domain-specific methods, naive integration variants (Arbor + Diag. and CORAL + Diag.), and EvoDAGRec. For these methods, the diagnostic content is emitted *stage-aware*: the bundle carries separate fields for the supervised-fine-tuning stage (`sft_*`) and the post-training stage (`post_*`), so that multi-stage diagnostics are produced only if a candidate actually modifies post-training; a single-stage candidate (e.g., pure TIGER) receives only the `sft_*` fields.

Channel (i): Offline validation metrics. The environment computes standard ranking metrics (Hit@K and NDCG@K, $K \in \{10, 20\}$) on the validation set via constrained beam search (beam size 20), as formalised in Appendix A.6. These scalars quantify *how well* the candidate ranks, but not *why* it fails, and are therefore complemented by the two channels below. In the feedback bundle the metrics appear both as the final-model value (`offline_val`) and as the per-stage values (`sft_stage_val`, `post_stage_val`) to enable stage-level credit assignment.

Channel (ii): Deterministic mathematical probes. The probe suite deterministically measures structural and decoding properties of the trained model, and is grouped into four families summarised in Table 4. All values are computed by the environment on the validation split; the baseline probe set is fixed and cannot be altered by any framework.

Probe extensibility. Although the baseline probe set is fixed, a candidate is permitted to implement and upload *additional* probe scripts to broaden its own diagnostic coverage. The environment merges candidate-contributed metrics into the baseline under an `agent.*` namespace and enforces an additive-only policy (candidate probes may not overwrite baseline probes), so the official diagnostic surface remains identical across frameworks while a framework may still monitor candidate-specific signals. The execution status of any submitted probe is reported back in the `sft_probe_fn_status` / `post_probe_fn_status` fields.

Table 4: The four families of deterministic mathematical probes. Each probe yields one or more numeric values (indexed by SID layer or popularity tier) and targets a specific internal failure mode of generative recommendation.

Representative probes	Diagnosed failure mode
Family: `sid_generation`	
`layer_entropy`, `layer_confidence`	per-layer decoding confidence; last-step or coarse-level collapse
`layer_top5_mass`	
`prefix_diversity_topk`	whether the top-K list concentrates on few shared prefixes (i.e., SID utilisation)
`branching_factor_per_layer`	fan-out per SID layer; whether the top-K list is a rich subtree or K parallel trajectories
`avg_lcp_depth_topk`	beam collapse to a single coarse prefix
`beam_prefix_collapse_rate`	
Family: `sid_generation_raw`	
`raw_layer_entropy`, `raw_layer_confidence`	intrinsic (unmasked) decoding behaviour, decoupled from trie constraints
`raw_layer_top5_mass`	
`raw_gt_layer_rank`, `raw_gt_layer_prob`	whether the ground-truth SID token is buried below distractors
Family: `output_behavior`	
`repetition_rate_topk`	failure to suppress already-owned items
`off_genre_rate`	over-broad generalisation / monotone or scattered lists
`subcategory_diversity_topk`	
`pop_bias_corr_pool`, `topk_pop_split`	popularity bias
`avg_topk_pop_rank_pct`	
`brand_concentration_topk`	brand over-fixation
Family: `representation_health`	
`embedding_collapse_score`	representation degeneration of item/token embeddings
`token_embedding_collapse.{a,b,c,d}`	per-position representation degeneracy

The raw numeric values are not fed to the coding agent directly. Instead, an LLM-based *Diagnosis Analyst* interprets them into a structured diagnosis following the schema of Table 5; the prompt instructs the analyst to ignore any conclusion-inducing wording, to ground every statement in a named metric and its value, and to populate the `headline_metrics` field with the 3-6 numeric values that most directly support the verdict (making the diagnosis auditable).

Channel (iii): Qualitative user diagnosis. Following the user-simulator paradigm in Self-EvolveRec (Kim et al., 2026), we instantiate an LLM-based simulator to produce behaviour-level

critiques. We construct a population of simulated users, each governed by a fixed persona spanning three traits (*activity*, *conformity*, and *diversity*, each ranging from LOW to HIGH). Conditioned on their individual interaction histories, these simulated users evaluate the candidate's top-K recommendations via a two-step protocol:

- **Item-level Assessment:** The simulator inspects each recommended item to predict the user's reaction. It outputs a binary verdict (align/misalign) along with a short causal reasoning for the decision.
- **List-level Feedback:** Building on the item verdicts, the simulator abstracts the overall quality of the top-K list. It identifies which decision signals the model correctly captured, which patterns were overgeneralized, and which user intentions were missed.

Finally, to mitigate individual-user bias and surface systematic flaws, a global aggregator (prompted as a *Lead Analyst*) synthesizes all per-user reports. This yields a single, macro-level system critique that strictly adheres to the unified `StandardDiagnosis` schema, serving as the qualitative counterpart to the mathematical probes.

Unified diagnosis schema. Both the mathematical and the qualitative channels converge on a single schema (Table 5), so that the coding agent consumes one uniform diagnostic interface regardless of which channel produced the content.

Table 5: The unified `StandardDiagnosis` schema shared by the mathematical and qualitative diagnostic channels.

Field	Type	Semantics
`status`	string	verdict in {`CRITICAL`, `NEEDS_IMPROVEMENT`, `STABLE`}
`alignment`	list[str]	behaviours where the model aligns with user choice logic
`misalignment`	list[str]	behaviours where the model misreads user intent
`overused_signals`	list[str]	decision patterns the model relies on excessively
`missing_signals`	list[str]	signals the model under-weights or misses
`primary_failure_mode`	string	the dominant failure mode at the top of the list
`system_interpretation`	string	one sentence on how the model decodes user intent
`diagnostic_signals_to_track`	list[str]	measurable signals to confirm or refute the diagnosis
`headline_metrics`	dict[str,float]	the numeric values backing the diagnosis (math channel)

As an illustration, the following qualitative diagnosis was produced for a pure cross-entropy TIGER candidate, demonstrating the concrete, behavioural language the agent receives:

```
{
  "status": "NEEDS_IMPROVEMENT",
  "system_interpretation": "The system interprets user intent as broad
    category affinity and brand popularity, often ignoring ownership,
    specific needs, and behavioural nuance such as activity level.",
  "misalignment": [
    "It recommends items the user already owns or recently purchased.",
    "It overgeneralizes from a single broad category, recommending many
    items from the same subcategory."
  ],
```

```
  "overused_signals": [
    "Heavy reliance on broad category similarity (e.g., 'Office & School
    Supplies').",
    "Overemphasis on brand popularity and brand-level co-occurrence."
  ],
  "primary_failure_mode": "The top items often include redundant or
    non-essential items because the system overgeneralizes category and
    brand signals."
}
```

A.6 METRICS

In this section, we will introduce offline recommendation metrics (Hit and NDCG) as well as metrics defined in Agent4rec that simulate user experience.

Offline Recommendation Metrics. Under the leave-one-out protocol each user u has exactly one ground-truth next item i_u^*. Given the ranked list R_u of length K produced by constrained beam search, let $\mathrm{rank}_u \in \{1, \dots, K\} \cup \{\infty\}$ denote the position of i_u^* in R_u ($\mathrm{rank}_u = \infty$ if it is not retrieved).

Hit@K measures whether the ground-truth item appears in the top-K list:

$$\mathrm{Hit@}K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \mathbb{I}(\mathrm{rank}_u \le K) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\left| R_u^K \cap \{i_u^*\} \right|}{|\{i_u^*\}|}. \tag{4}$$

NDCG@K additionally rewards placing the ground-truth item near the top of the list. With binary relevance $\mathrm{rel}_j = \mathbb{I}(R_u[j] = i_u^*)$,

$$\mathrm{DCG@}K(u) = \sum_{j=1}^{K} \frac{2^{\mathrm{rel}_j} - 1}{\log_2(j+1)} = \frac{\mathbb{I}(\mathrm{rank}_u \le K)}{\log_2(\mathrm{rank}_u + 1)}, \qquad \mathrm{IDCG@}K(u) = \frac{1}{\log_2 2} = 1, \tag{5}$$

so that

$$\mathrm{NDCG@}K = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\mathrm{DCG@}K(u)}{\mathrm{IDCG@}K(u)} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \frac{\mathbb{I}(\mathrm{rank}_u \le K)}{\log_2(\mathrm{rank}_u + 1)}. \tag{6}$$

We report $K \in \{10, 20\}$. Since the beam size equals 20, the top-20 list is exactly the beam-search output, and the top-10 list is its truncation.

Agent4Rec Metrics. Offline recommendation metrics summarise a single next-item prediction and are therefore blind to list-level behavioural effects such as over-concentration. To complement them, we additionally report three behaviour-level metrics obtained from the LLM-based user simulator Agent4Rec (Zhang et al., 2024). In this simulator each generative agent is initialised from a real user profile (activity, conformity and diversity traits) and interacts with the evaluated recommender *page by page*: four items are displayed per page, the agent chooses which items to view and rates them on a 1-5 scale, and then decides whether to request the next page or to exit, being forced to exit after at most five pages. A rating higher than 3 is treated as a *like*. Let $\mathcal{A}$ denote the evaluation agent population, and for agent a let E_a be the number of exposed items, L_a the number of liked items, and $p_a \in [1, 5]$ the page index at which a exits. We report

$$\bar{P}_{\mathrm{like}} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} \frac{L_a}{E_a}, \qquad \bar{N}_{\mathrm{like}} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} L_a, \qquad \bar{N}_{\mathrm{exit}} = \frac{1}{|\mathcal{A}|} \sum_{a \in \mathcal{A}} p_a. \tag{7}$$

Higher values indicate a better simulated user experience for all three metrics. $\bar{P}_{\mathrm{like}}$ measures the precision of the exposed list at the behaviour level, $\bar{N}_{\mathrm{like}}$ measures the absolute volume of satisfied interests a session delivers, and $\bar{N}_{\mathrm{exit}}$ measures engagement depth: a larger exit page indicates the agent kept browsing rather than abandoning the session early.

Agent4Rec metrics are used strictly for evaluation. They are computed **only once, on the single checkpoint finally retained under the budget** B, and serve exclusively to compare the frameworks in the main results table. They are never computed for intermediate candidates, never exposed to any evolving framework, and never used for model selection. Model selection relies solely on the offline

validation metrics defined above. Consequently, no framework can optimise these numbers directly, and they function as a genuinely held-out behavioural measurement.

We stress that this evaluation simulator is **fully disjoint** from the user simulator that supplies the third feedback channel during evolution. The latter follows the qualitative-diagnosis design of Self-EvolveRec (Kim et al., 2026) and emits natural-language diagnostic reports (Appendix A.5) rather than scalar scores. The two components are isolated along two axes: (i) the user populations are drawn from disjoint user sets, so no agent that participates in evolution-time diagnosis is ever reused for final measurement; and (ii) their outputs are of different kinds and consumed by different parties: diagnostic text is consumed by the coding agent during the search, whereas the three metrics above are consumed only by us, after the search terminates. This separation rules out any feedback path from the evaluation protocol back into the evolutionary process. Within the evaluation simulator itself, the agent population, the random seeds and the interaction templates are fixed once and shared by all frameworks, so the reported comparison is deterministic and paired.