Self-EvolveRec: Self-Evolving Recommender Systems with LLM-based Directional Feedback

Sein Kim[1], Sangwu Park[1], Hongseok Kang[1], Wonjoong Kim[1], Jimin Seo[1]
Yeonjun In[1], Kanghoon Yoon[1], Hyunsik Jeon[2], Chanyoung Park[1,*]

[1]KAIST [2]Roblox

{rlatpdlsgns, sangwu.park, ghdtjr0311, wjkim,
jimin.seo, yeonjun.in, ykhoon08, cy.park}@kaist.ac.kr
hjeon@roblox.com [*]Corresponding author

Abstract

Traditional methods for automating recommender system design, such as Neural Architecture Search (NAS), are often constrained by a fixed search space defined by human priors, limiting innovation to pre-defined operators. While recent LLM-driven code evolution frameworks shift fixed search space target to open-ended program spaces, they primarily rely on scalar metrics (e.g., NDCG, Hit Ratio) that fail to provide qualitative insights into model failures or directional guidance for improvement. To address this, we propose Self-EvolveRec, a novel framework that establishes a directional feedback loop by integrating a User Simulator for qualitative critiques and a Model Diagnosis Tool for quantitative internal verification. Furthermore, we introduce a Diagnosis Tool - Model Co-Evolution strategy to ensure that evaluation criteria dynamically adapt as the recommendation architecture evolves. Extensive experiments demonstrate that Self-EvolveRec significantly outperforms state-of-the-art NAS and LLM-driven code evolution baselines in both recommendation performance and user satisfaction. Our code is available at `https://github.com/Sein-Kim/self_evolverec`.

1 Introduction

Driven by the growth of online data [6, 13], recommender systems have evolved from Matrix Factorization [16, 44, 20, 21] to Graph Neural Networks [51, 17] and Transformers [19, 48]. However, optimal performance relies heavily on the entire recommendation pipeline (e.g., loss functions, negative sampling) rather than model architecture alone [38, 61], making manual refinement of this pipeline by human expertise inefficient and costly [58]. To mitigate manual design inefficiencies, Automated Machine Learning (AutoML) techniques, such as Neural Architecture Search (NAS) [60, 8] have emerged as a prominent approach to automating the discovery of optimal recommendation architectures [31, 46, 27, 56, 30].

However, these methodologies are inherently constrained by a fixed search space bounded by human priors [5, 40], limiting optimization to symbolic combinations within a closed pool of pre-defined operators. Due to this lack of generative expressivity, existing NAS methods struggle to address non-architectural components—such as loss functions and data processing—and fail to jointly optimize the entire recommendation pipeline. Consequently, achieving comprehensive system optimization requires shifting from a closed operator pool to an open-ended program space [40].

To realize open-ended optimization, a new paradigm known as LLM-driven code evolution has emerged. Pioneering frameworks such as FunSearch [42] and Eureka [35] validated LLM-driven code evolution approach by leveraging LLMs to optimize isolated functions or specific logic components. Substantially expanding this scope, AlphaEvolve [39] targets entire codebases rather than single functions. By orchestrating an autonomous evolutionary pipeline, AlphaEvolve iteratively optimizes complex algorithmic structures and computational stacks through direct code modifications. Complementing AlphaEvolve, DeepEvolve [33] extends the evolution loop by incorporating Retrieval-Augmented Generation (RAG) [29, 11]. By retrieving academic papers from arXiv, it

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

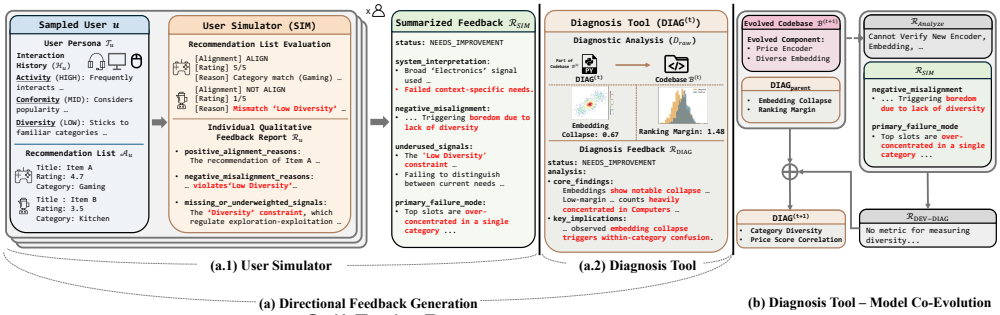


Figure 1: Two core mechanism of Self-EvolveRec. (a) is the overview of the Directional Feedback Generation: (a.1) is the user simulator, (a.2) is the model diagnosis tool. (b) is the Diagnosis Tool - Model Co-evolution.

leverages external scientific knowledge to facilitate systematic idea generation and codebase refinement. Unlike NAS confined to combining existing modules, these LLM-driven approaches enable open-ended optimization akin to human researchers, allowing the invention of novel components beyond pre-defined design space.

Despite this open-ended capability, existing LLM-driven methods are guided primarily by scalar metrics (e.g., accuracy, MSE) that lack diagnostic insight, restricting evolution to an undirected trial-and-error search. This is particularly problematic in recommender systems, where—unlike mathematical problems with deterministic ground-truth [39]—failures are multifaceted: a drop in NDCG or Hit Ratio cannot reveal whether the cause is excessive popularity bias, insufficient category diversity, or a failure to capture short-term interests [47, 59]. Thus, effective code evolution in this domain requires *directional feedback* that diagnoses root causes of failure and reflects user experience to guide the evolution process.

To address this challenge, we propose Self-EvolveRec, an LLM-driven code evolution framework that orchestrates a directional feedback loop by integrating a **User Simulator** (SIM) and a **Model Diagnosis Tool** (DIAG). SIM evaluates recommendation lists through diverse user personas, providing qualitative natural language critiques (e.g., *"I seek low-cost accessories, not expensive electronics"*) that explain *why* a recommendation fails—an insight scalar metrics cannot offer. To complement SIM and detect deficiencies invisible to user perception (e.g., embedding collapse), DIAG quantitatively probes the model's structural properties, such as the geometric distribution of item representations. By corroborating qualitative critiques with quantitative diagnostics, Self-EvolveRec accurately pinpoints the root causes of failure within the recommendation pipeline.

However, this static design becomes inadequate as the pipeline evolves: a fixed DIAG cannot inspect newly introduced components (e.g., multi-modal encoders, novel loss functions) and lacks metrics to verify newly emerging SIM critiques (e.g., "lack of diversity"). To address this, we design *Diagnosis Tool - Model Co-Evolution*, which dynamically updates DIAG to (i) inspect new components and (ii) translate SIM's qualitative critiques into measurable metrics. We additionally introduce an optional efficient variant based on zero-cost proxies [28, 1] for scalable deployment. Extensive experiments demonstrate that Self-EvolveRec outperforms NAS and LLM-driven code evolution baselines, with directional feedback yielding substantial gains in recommendation performance, user satisfaction, and the technical quality of the evolved algorithmic logic. We defer a detailed discussion of related work to App. A.

2 Problem Definition

Dataset. Let $\mathcal{D} = (\mathcal{U}, \mathcal{V}, \mathcal{E}, \mathcal{X})$ denote the dataset with user set $\mathcal{U}$ and item set $\mathcal{V}$. The interaction set $(u, v, t_{u,v}, r_{u,v}, \text{rev}_{u,v}) \in \mathcal{E}$ indicates that user $u \in \mathcal{U}$ interacted with item $v \in \mathcal{V}$ at timestamp $t_{u,v}$, providing a rating $r_{u,v}$ and a textual review $\text{rev}_{u,v}$. Based on $\mathcal{E}$, we define the interaction history for user u as $\mathcal{H}_u = \{v \mid (u, v, t_{u,v}) \in \mathcal{E}\}$, chronologically ordered by $t_{u,v}$. Additionally, $x_v \in \mathcal{X}$ denotes the set of side information, where x_v represents the raw attribute set for item v (e.g., category, title, and price).

Optimization Goal. Let $\mathcal{B} \in \mathbb{S}$ denote the entire codebase governing the *recommendation pipeline* within the **open-ended program space** $\mathbb{S}$. The evolutionary process starts with a *seed codebase* $\mathcal{B}^{(0)}$ and proceeds through T iterations, where $\mathcal{B}^{(t)}$ represents the evolved codebase at iteration t. Specifically, $\mathcal{B}^{(0)}$ constitutes a fully functional seed recommendation pipeline, encapsulating a seed recommender architecture $f_{\mathcal{B}^{(0)}}(\cdot; \theta_{\mathcal{B}^{(0)}})$ (e.g., NCF [16]), data processing logic (e.g., basic loaders for interactions $\mathcal{E}$ and raw attributes $\mathcal{X}$), and a standard optimization loop (e.g., loss computation, and parameter updates). Our goal is to evolve this seed codebase $\mathcal{B}^{(0)}$ into an optimal codebase $\mathcal{B}^*$

that maximizes a standard recommendation metric $\mathcal{M}$ (e.g., Hit Ratio, NDCG) within T iterations. We formulate this code evolution task as a bi-level optimization problem:

$$\mathcal{B}^* = \underset{\mathcal{B}\in\mathbb{S}}{\operatorname{argmax}} \underbrace{\mathcal{M}\left(f_{\mathcal{B}}(\mathcal{E}_{\text{val}}, \mathcal{X}; \theta^*_{\mathcal{B}})\right)}_{\text{score}(\mathcal{B})} \quad \text{s.t. } \theta^*_{\mathcal{B}} = \underset{\theta}{\operatorname{argmin}}\, \mathcal{L}_{\mathcal{B}}\left(f_{\mathcal{B}}(\mathcal{E}_{\text{train}}, \mathcal{X}; \theta)\right) \tag{1}$$

where $\mathcal{L}_{\mathcal{B}}$ represents the loss function defined within $\mathcal{B}$. Here, $\theta^*_{\mathcal{B}}$ indicates the optimal model parameters for the architecture defined by $\mathcal{B}$, learned on the training set $\mathcal{E}_{\text{train}}$ by minimizing $\mathcal{L}_{\mathcal{B}}$.

3 Proposed Framework: Self-EvolveRec

In this section, we propose Self-EvolveRec, a novel framework designed to enable LLMs to perform directional feedback-based evolutionary code optimization for recommender systems. As illustrated in Figure 1, Self-EvolveRec operates through two core mechanisms to overcome the limitations of existing metric-based approaches [39, 33]. **First**, we establish *Directional Feedback Generation* (Sec. 3.1, Figure 1 (a)), which integrates a *User Simulator* (Sec. 3.1.1) and a *Model Diagnosis Tool* (Sec. 3.1.2) to provide qualitative and quantitative guidance. This feedback then guides the *Evolution Pipeline* (Sec. 3.2, Figure 2) to perform precise, directed code modifications. **Second**, we introduce *Diagnosis Tool - Model Co-evolution* (Sec. 3.3, Figure 1 (b)), a strategy that ensures the diagnosis tool to dynamically adapt to the structural changes of the recommendation pipeline.

3.1 Directional Feedback Generation

While scalar metrics (e.g., NDCG) quantify how well a recommendation pipeline performs, they lack the semantic depth to explain why it fails or how to resolve it. To address this, we introduce **Directional Feedback**, which integrates qualitative critiques from the User Simulator and quantitative verifications from the Model Diagnosis Tool. This mechanism translates non-interpretable numerical metrics into actionable insights, enabling the LLM to pinpoint and resolve structural deficiencies within the recommendation pipeline.

3.1.1 User Simulator: Qualitative Critique

We employ a User Simulator (SIM) to complement standard metrics with qualitative directional feedback. While scalar metrics (e.g., NDCG) quantify *how well* a model performs, they fail to reveal the root causes of failure [36] such as insufficient category diversity or an inability to capture short-term interests. By adopting the agentic paradigm [54, 34], as depicted in Figure 1 (a.1), our simulator acts as a diverse set of virtual users, offering explicit natural language critiques that pinpoint specific deficiencies in the recommendation pipeline.

To ensure behavioral realism and heterogeneity, we characterize each simulated user u through a structured persona $\mathcal{T}_u$, constructed by combining the user's interaction history $\mathcal{H}_u$ with sociopsychological traits [54, 37]. Following [54], we use three traits—**Activity**, **Conformity**, and **Diversity**—capturing engagement level, mainstream adherence, and interest breadth, respectively, each quantized into three levels (LOW, MID, HIGH) and mapped to natural language descriptions (e.g., *"Activity (HIGH): Frequently interacts with the recommender..."*); see App. B for details. Our framework is agnostic to specific persona definitions, and alternative schemes such as the Big Five [34, 12] can be integrated, as explored in App. G.3.

The feedback generation process proceeds in two steps. First, SIM conducts an individual assessment for a sampled user u by evaluating the recommendation list $\mathcal{A}_u$ from a trained recommender $f_{\mathcal{B}}$ (e.g., NCF) based on their persona $\mathcal{T}_u$ and interaction history $\mathcal{H}_u$. This assessment identifies specific behavioral or semantic misalignment, which is formalized into an individual qualitative feedback report $R_u = \text{LLM}(\mathcal{I}_{\text{SIM}}, \mathcal{T}_u, \mathcal{H}_u, \mathcal{A}_u)$, where $\mathcal{I}_{\text{SIM}}$ is instruction for user simulator. Throughout the paper, we denote the task-specific instructions guiding the LLM-agent as $\mathcal{I}_{\text{task}}$ (e.g., $\mathcal{I}_{\text{SIM}}$, $\mathcal{I}_{\text{PLAN}}$, $\mathcal{I}_{\text{CODE}}$). While we describe the high-level objective of each instruction within the main text, the exact prompt templates are provided in App. H.1. Second, to mitigate individual user bias and capture common failure patterns, we summarize reports from a set of sampled users $\mathcal{U}_{\text{sample}}$ into a comprehensive summary $\mathcal{R}_{\text{SIM}} = \text{LLM}(\mathcal{I}_{\text{SUMMARIZE}}, \{R_u \mid u \in \mathcal{U}_{\text{sample}}\})$, where $\mathcal{I}_{\text{SUMMARIZE}}$ guides the LLM to abstract common failure patterns from individual critiques.

3.1.2 Model Diagnosis Tool: Quantitative Verification

While the SIM generates qualitative feedback from user experience, relying solely on this feedback fails to detect hidden structural issues in terms of the model, such as embedding collapse. To address this, we introduce the Model Diagnosis Tool (DIAG; denoted as $\text{DIAG}^{(t)}$ at iteration t), which is a computational probing module designed to verify structural or behavioral deficiencies of the

recommender system. As depicted in Figure 1 (a.2), unlike the SIM, $\text{DIAG}^{(t)}$ directly accesses the model parameters θ and data loaders within the codebase $\mathcal{B}^{(t)}$ to conduct a systematic validity check. As an initial instantiation of the diagnosis tool $\text{DIAG}^{(0)} \in \mathcal{B}^{(0)}$, we implement two foundational probes designed to detect common structural failures in recommender models:

1) Embedding Collapse: To detect a state where representations degenerate into a narrow subspace losing discriminative power, DIAG computes the mean pairwise cosine similarity across sampled item embeddings. A high similarity score serves as a proxy for representation degeneration.

2) Ranking Margin: DIAG evaluates decision boundaries by analyzing the ranking margin $\Delta_{u,v} = s(u,v) - s(u,v')$ for all users $u \in \mathcal{U}$ and their observed interactions $v \in \mathcal{H}_u$. Here, $s(u,v)$ and $s(u,v')$ denote the predicted logits for the ground-truth item v and a randomly sampled negative item $v' \notin \mathcal{H}_u$, respectively. To assess overall discriminative power, DIAG computes the global average margin $\mathbb{E}_{u\in\mathcal{U}, v\in\mathcal{H}_u}[\Delta_{u,v}]$. Specifically, a high margin indicates robust discrimination between ground-truth item and negative item, whereas a low or negative margin indicates a failure to distinguish ground-truth. To pinpoint the potential failure modes of such failures, DIAG aggregates cases with extremely low margins and counts their common attributes (e.g., specific categories like *Computers* in Figure 1 (a.2) - "core_findings").

These probes generate a set of raw numerical diagnostics analysis $D_{\text{raw}} = \text{DIAG}^{(t)}(\mathcal{B}^{(t)})$. Subsequently, to bridge the gap between numerical diagnostics and algorithmic solutions, an LLM acts as a senior researcher to interpret these signals, converting them into a structured diagnosis report $\mathcal{R}_{\text{DIAG}} = \text{LLM}(\mathcal{I}_{\text{DIAG}}, D_{\text{raw}})$. Furthermore, DIAG serves as a verification mechanism to check whether the qualitative deficiencies pointed out by SIM are actually resolved (detailed in Sec. 3.3).

3.2 Evolution Pipeline

In this section, we detail the iterative execution workflow of Self-EvolveRec, designed to autonomously refine the codebase $\mathcal{B}$ through cycles of evaluation, reasoning, and evolution. To support these cycles, Self-EvolveRec adopts a population-based exploration strategy [49, 33, 42, 39] by maintaining an Evolutionary Archive $\mathcal{P}^{(t)}$ that stores the comprehensive history of codebases and feedback. At each iteration, a target codebase is selected from this archive to serve as the parent, denoted as $\mathcal{B}_{\text{parent}} \sim \text{Sample}(\mathcal{P}^{(t)})$. Unlike existing studies [39, 33] that rely solely on scalar metrics, which often lead to aimless exploration, Self-EvolveRec actively integrates the **Directional Feedback** mechanisms (Sec. 3.1). By integrating qualitative insights from the *user feedback report* ($\mathcal{R}_{\text{SIM}}$) and quantitative findings from the *diagnosis report* ($\mathcal{R}_{\text{DIAG}}$), Self-EvolveRec shifts the focus from mere scalar metrics to pinpoint the root causes of failure. As illustrated in Figure 2, Self-EvolveRec follows a four-phase workflow to address these failures and evolve the codebase.

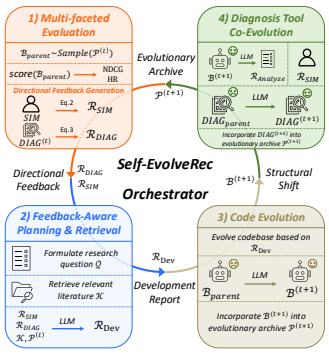


Figure 2: Overall evolutionary pipeline of Self-EvolveRec.

We first detail the three-phase workflow dedicated to refinement of the recommendation codebase, i.e., 1), 2), and 3) in Figure 2 ($\mathcal{B}_{\text{parent}} \to \mathcal{B}^{(t+1)}$), followed by the Diagnosis Tool Co-Evolution stage, i.e., 4) in Figure 2 (Sec. 3.3).

1) Multi-faceted Evaluation. First, Self-EvolveRec evaluates the currently selected codebase $\mathcal{B}_{\text{parent}}$ to obtain the scalar score shown in Equation 1 (i.e., $\text{score}(\mathcal{B}_{\text{parent}})$), while simultaneously generating the user feedback $\mathcal{R}_{\text{SIM}}$ and diagnosis report $\mathcal{R}_{\text{DIAG}}$[1]. These outputs serve as the directional context for the subsequent planning.

2) Feedback-Aware Planning & Retrieval. Self-EvolveRec introduces a *feedback-aware planning & retrieval* that shifts the RAG methods in previous work [33] from general-purpose knowledge gathering to targeted failure resolution. By conditioning on directional feedback from $\mathcal{R}_{\text{SIM}}$ and $\mathcal{R}_{\text{DIAG}}$, the LLM agent acts as a planner to formulate precise feedback targeted research queries $\mathcal{Q} = \text{LLM}(\mathcal{I}_{\text{PLAN}}, \mathcal{R}_{\text{SIM}}, \mathcal{R}_{\text{DIAG}}, \mathcal{P}^{(t)})$. Conditioned on specific failure modes and the evolutionary archive $\mathcal{P}^{(t)}$, the agent generates targeted queries $\mathcal{Q}$ (e.g., "Retrieve methods to mitigate category mismatch identified in user reports") to retrieve relevant academic literature $\mathcal{K}$ from online sources, including arXiv[2]. These targeted queries ensure a retrieval focusing on resolving the identified

[1]To maximize efficiency, if the selected codebase $\mathcal{B}_{\text{parent}}$ is already recorded in the archive $\mathcal{P}^{(t)}$ with complete evaluation logs, Self-EvolveRec retrieves the cached results instead of re-executing the evaluation process.

[2]We set the number of literature (i.e., $|\mathcal{K}|$) to 3. Refer to App. G.2.3 for experiments regarding the number of literature.

structural deficiencies, rather than broad algorithmic exploration. These insights are then integrated into a structured *Development Report* $\mathcal{R}_{\text{Dev}} = \text{LLM}(\mathcal{I}_{\text{REPORT}}, \mathcal{R}_{\text{SIM}}, \mathcal{R}_{\text{DIAG}}, \mathcal{P}^{(t)}, \mathcal{K})$, where $\mathcal{R}_{\text{Dev}}$ outlines the algorithmic modifications required to address the identified issues.

3) Code Evolution. Guided by $\mathcal{R}_{\text{Dev}}$, the agent implements code-level modifications to instantiate the updated codebase $\mathcal{B}^{(t+1)} = \text{LLM}(\mathcal{I}_{\text{CODE}}, \mathcal{R}_{\text{Dev}}, \mathcal{B}_{\text{parent}}, \mathcal{P}^{(t)})$. Upon successful execution, the new codebase $\mathcal{B}^{(t+1)}$ is incorporated into the population history $\mathcal{P}^{(t+1)}$, following standard evolutionary protocols [39, 33]. This cycle recursively refines $\mathcal{B}$ to optimize the objective defined in Equation 1.

3.3 Diagnosis Tool - Model Co-Evolution

The evolution of the DIAG (i.e., 4) in Figure 2) is driven by two objectives: **First**, as $\mathcal{B}$ undergoes structural transformations, such as the introduction of new loss functions or architectural layers, a static DIAG inevitably becomes obsolete. DIAG must continuously adapt to analyze and evaluate these new components effectively. **Second**, DIAG serves to quantitatively verify the qualitative insights provided by the SIM. While the SIM offers rich, human-like critiques that traditional metrics fail to capture (e.g., perceiving a recommendation list as "conceptually repetitive" despite high accuracy scores), these subjective critiques must be translated into measurable metrics to confirm their validity and assess their impact for precise code optimization. Therefore, DIAG evolves to stay compatible with the evolved codebase while formulating specific metrics that mathematically capture the essence of the SIM's feedback. As depicted in Figure 1 (b), if the SIM reports "boredom due to lack of diversity," DIAG autonomously implements a tailored metric (e.g., measuring diversity among top-k items) to quantify this feedback precisely, ensuring that the subsequent evolution is guided by concrete objectives.

To address this, we implement a co-evolution mechanism that begins with understanding the structural shifts. Since the codebase is continuously updated ($\mathcal{B}_{\text{parent}} \rightarrow \mathcal{B}^{(t+1)}$), the agent first scans the new codebase $\mathcal{B}^{(t+1)}$ and original $\text{DIAG}_{\text{parent}}$ to generate a structural analysis report $\mathcal{R}_{\text{Analyze}} = \text{LLM}(\mathcal{I}_{\text{Analyze}}, \mathcal{B}^{(t+1)}, \text{DIAG}_{\text{parent}})$. The $\mathcal{R}_{\text{Analyze}}$ summarizes key information such as the updated execution flow, newly added modules, and modified loss functions. This blueprint enables the agent to design diagnostic criteria that are structurally compatible with the new architecture.

Subsequently, to synchronize the diagnosis tool with both the structural changes and the user feedback $\mathcal{R}_{\text{SIM}}$, the agent executes the evolution cycle. By integrating the qualitative $\mathcal{R}_{\text{SIM}}$ with the structural blueprint $\mathcal{R}_{\text{Analyze}}$, the agent identifies evaluation gaps (e.g., cannot verify new encoder, and embedding). Similar to the main pipeline (Sec. 3.2), it retrieves relevant methodologies $\mathcal{K}_{\text{DIAG}}$ using research queries $\mathcal{Q}_{\text{DIAG}} = \text{LLM}\left(\mathcal{I}_{\text{PLAN-DIAG}}, \mathcal{R}_{\text{SIM}}, \mathcal{R}_{\text{Analyze}}, \mathcal{P}^{(t)}\right)$ and then updates the DIAG as: $\text{DIAG}^{(t+1)} = \text{LLM}\left(\mathcal{I}_{\text{CODE-DIAG}}, \mathcal{R}_{\text{Dev-DIAG}}, \mathcal{B}^{(t+1)}, \text{DIAG}_{\text{parent}}, \mathcal{P}^{(t)}\right)$, where $\mathcal{R}_{\text{Dev-DIAG}} = \text{LLM}\left(\mathcal{I}_{\text{REPORT-DIAG}}, \mathcal{R}_{\text{SIM}}, \mathcal{R}_{\text{Analyze}}, \mathcal{K}_{\text{DIAG}}\right)$.

The new model diagnosis tool $\text{DIAG}^{(t+1)}$ is incorporated into the population history $\mathcal{P}^{(t+1)}$. This adaptive process guarantees that $\text{DIAG}^{(t+1)}$ is equipped with both the logic to inspect new architectures and the specific metrics required to transform the simulator's qualitative feedback into actionable, quantitative signals. We further investigate the additional evolutionary pipeline of the User Simulator in experiments in Sec. 4.3.4.

3.4 Efficient Variant via Zero-Cost Proxies

While Self-EvolveRec effectively guides evolution through directional feedback, fully training the recommender at every iteration is computationally expensive. To enable scalable deployment, we additionally introduce an **optional efficient variant** that replaces full training with zero-cost proxies [28, 1], which estimate pipeline potential through a single forward-backward pass on a representative mini-batch. We design a composite proxy $\mathcal{M}_{\text{zero}}$ combining representation-health signals (effective ranks, top-1 singular value share) and gradient-health signals to surrogate standard recommendation metrics. Both DIAG and SIM are correspondingly adapted: DIAG leverages the sub-signals of $\mathcal{M}_{\text{zero}}$ as foundational probes, while SIM operates on a code-to-intent summary of the evolved codebase. Detailed formulations are provided in App. F, and empirical validation showing up to $12\times$ speedup with marginal performance degradation is provided in App. G.2.5.

4 Experiments

Datasets. For evaluations, we used three Amzaon datasets [18] (CDs, Electronics, and Office) and the MovieLens dataset [15]. Following prior works [19, 48, 22], we use five-core datasets. Detailed statistics for each dataset are provided in Table 7 in App. D.

Table 1: Overall model performance (A: AlphaEvolve, D: DeepEvolve).

Dataset	Metric	NAS				Seed Recommender: NCF				Seed Recommender: NGCF				Seed Recommender: SASRec				Seed Recommender: MoRec			
		AutoFIS	NASRec	AutoLossGen	DNS-Rec	NCF	A	D	**Ours**	NGCF	A	D	**Ours**	SASRec	A	D	**Ours**	MoRec	A	D	**Ours**
CDs	NDCG@5	0.2077	0.2026	0.2123	0.2636	0.2312	0.2556	0.2421	**0.2723**	0.3446	0.3449	0.3280	**0.3721**	0.3559	0.3528	0.3610	**0.3865**	0.2558	0.2414	0.3701	**0.3977**
	HR@5	0.3040	0.2989	0.3063	0.3605	0.2979	0.3477	0.3497	**0.3799**	0.4924	0.4896	0.4465	**0.4974**	0.4676	0.4623	0.4870	**0.5274**	0.3779	0.3607	0.4864	**0.5340**
Electronics	NDCG@5	0.1753	0.1706	0.1877	0.1598	0.1078	0.1183	0.1817	**0.1907**	0.1531	0.1808	0.1726	**0.1925**	0.2325	0.2063	0.2508	**0.2600**	0.1883	0.1912	0.1938	**0.2056**
	HR@5	0.2456	0.2444	0.2512	0.2271	0.1610	0.1733	0.2600	**0.2714**	0.2247	0.2590	0.2385	**0.2759**	0.3208	0.2891	0.3427	**0.3591**	0.2724	0.2675	0.2745	**0.2921**
Office	NDCG@5	0.1714	0.1377	0.1610	0.1285	0.1620	0.1751	0.1705	**0.1759**	0.1711	0.1799	0.1805	**0.1930**	0.1799	0.1939	0.1816	**0.2329**	0.1851	0.1848	0.1697	**0.1884**
	HR@5	0.2490	0.2070	0.2419	0.1879	0.2343	0.2523	0.2548	**0.2659**	0.2494	0.2634	0.2591	**0.2743**	0.2526	0.2750	0.2631	**0.3218**	0.2689	0.2633	0.2447	**0.2703**
MovieLens	NDCG@5	0.1369	0.2916	0.2348	0.3340	0.3413	0.3465	0.1908	**0.3764**	0.1824	0.2355	0.2010	**0.3588**	0.5667	0.5583	0.5722	**0.5765**	0.3796	0.4993	0.5281	**0.5460**
	HR@5	0.2099	0.4346	0.3555	0.5046	0.4970	0.5091	0.2735	**0.5475**	0.2876	0.3672	0.3162	**0.5220**	0.7283	0.7199	0.7344	**0.7366**	0.5414	0.6743	0.6967	**0.7131**

Baselines. We evaluate Self-EvolveRec against representative NAS-based recommender architecture search methods, i.e., AutoFIS [31], NASRec [56] AutoLossGen [30], and DNS-Rec [55], as well as recent LLM-driven evolutionary frameworks, i.e., AlphaEvolve [39] and DeepEvolve [33]. Details regarding the baseline methods are provided in App. E.1.

Evaluation Protocol. We use the leave-last-out strategy [19, 48, 50, 25, 4] for evaluation of recommender models, where we use the most recent item and second most item for testing and validation, respectively, and the remaining history for training. Each test item is paired with 99 randomly sampled non-interacted items. We report performance using two standard metrics: Normalized Discounted Cumulative Gain (NDCG@5) and Hit Ratio (HR@5). Crucially, the standard ranking gains (NDCG@5, HR@5) are measured on real held-out interactions and do not involve any LLM-based component, providing simulator-independent evidence that the directional feedback induces genuine recommendation improvements rather than artifacts of LLM-based evaluation.

Implementation Details. To ensure fair comparisons, all evolution-based methods are initialized with four distinct seed recommenders: NCF [16] (MF-based), NGCF [51] (graph-based), SASRec [19] (sequential), and MoRec [53] (multi-modal). We employ GPT-5-mini for planning and retrieval, while utilizing GPT-5 as the coding agent for all frameworks. We provide additional experiments with different LLMs in App. G.1. To account for the computational overhead associated with these LLMs, we provide a comprehensive efficiency analysis, including time cost per iteration and user sampling impact, in App. G.2. Please refer to the App. C and E.2 for more details regarding the seed recommender and hyper-parameters settings.

4.1 Performance Comparison

Overall performance. The results of the recommendation task on four datasets are given in Table 1. From the results, we have the following observations: **1)** Self-EvolveRec consistently outperforms AlphaEvolve and DeepEvolve across all datasets, regardless of the seed model. These results demonstrate that integrating qualitative critiques with quantitative diagnostics enables effective algorithmic evolution across diverse recommender architectures. **2)** AlphaEvolve and DeepEvolve exhibit inconsistency across datasets, with both baselines occasionally underperforming the initial seeds. This inconsistency indicates that without directional feedback, these baselines are limited to trial-and-error searches. Conversely, Self-EvolveRec leverages semantic critiques to identify and resolve specific structural deficiencies, leading to more informed and effective improvements. **3)** Self-EvolveRec significantly surpasses NAS-based baselines, such as AutoFIS and NASRec, in all datasets. This confirms that shifting from a closed, human-defined operator pool to an open-ended program space allows for more expressive logic and structural algorithmic refinement.

User Satisfaction Analysis.

To bridge the gap between static metrics and actual user satisfaction, we adopt the agentic simulation environment from Agent4Rec [54] and PUB [34] as a scalable proxy for A/B testing. To mitigate concerns about a simulator-to-simulator evaluation loop, our setup deliberately decouples the optimization-time simulator (SIM) from the evaluation-time simulators along three axes: (i) **disjoint user populations** – both Agent4Rec and PUB sample evaluation users that are unseen by SIM during evolution; (ii) **distinct persona schemes** – PUB employs Big Five traits, whereas SIM uses Activity/Conformity/Diversity; and (iii) **independent frameworks** – both evaluators are external systems not modified by our pipeline.

Table 2: Multi-facet satisfaction metrics on SASRec and NCF under two agentic evaluators (Agent4Rec and PUB) (S: Seed Recommender, A: AlphaEvolve, D: DeepEvolve).

Seed Recommender	Agentic Evaluator	Metric	CDs				Electronics			
			S	A	D	**Ours**	S	A	D	**Ours**
SASRec	Agent4Rec	View	0.372	0.378	0.379	**0.381**	0.335	0.342	0.351	**0.353**
		Satisfy	4.606	4.384	4.710	**5.046**	4.173	4.308	4.487	**4.502**
		Depth	1.926	1.830	1.952	**2.048**	1.754	1.778	1.782	**1.797**
	PUB	View	0.128	0.122	0.130	**0.136**	0.133	0.134	0.141	**0.144**
		Satisfy	4.630	4.506	4.810	**4.906**	3.928	3.972	4.082	**4.134**
		Depth	1.912	1.910	1.922	**2.018**	1.856	1.888	1.904	**1.928**
NCF	Agent4Rec	View	0.365	0.354	0.369	**0.372**	0.339	0.337	0.307	**0.342**
		Satisfy	4.206	4.461	4.392	**4.650**	4.320	4.270	4.132	**4.354**
		Depth	1.776	1.882	1.840	**1.934**	1.786	1.744	1.720	**1.798**
	PUB	View	0.125	0.127	0.119	**0.134**	0.135	0.125	0.124	**0.139**
		Satisfy	4.488	4.650	4.422	**4.770**	3.842	3.826	3.564	**3.952**
		Depth	1.914	1.918	1.940	**1.952**	1.874	1.768	1.754	**1.916**

In this dynamic environment, generative agents driven by distinct traits (e.g., Big Five) are presented with pages of four items and are asked to decide whether to continue or terminate the session based

on relevance and diversity of recommended items. Following Agent4Rec, we quantify this simulation using three key user-centric metrics: **View** (item view ratio), **Satisfy** (a comprehensive score about recommender from 1 to 10), and **Depth** (the number of pages explored before termination). Notably, in the Agent4Rec setup, agents terminate recommendation sessions upon encountering unsatisfactory items, indicating that a higher **Depth** signifies successful user retention through consistently relevant recommendations. The results of the user satisfaction are summarized in Table 2. From the results, we have the following observations: **1)** Self-EvolveRec consistently outperforms all baselines across all satisfaction-oriented metrics (i.e., View, Satisfy, and Depth) under **both Agent4Rec and PUB**, regardless of the seed model. The gains under PUB are particularly informative: since PUB's Big Five persona scheme is fundamentally distinct from the Activity/Conformity/Diversity scheme used by SIM during evolution, these results confirm that Self-EvolveRec does not overfit to the critique style of its own optimization-time simulator. Instead, the directional feedback from $\mathcal{R}_{\text{SIM}}$ enables the models to evolve beyond mere accuracy, substantially improving the perceived quality of recommendations from a user-centric perspective and sustaining user engagement for longer durations. **2)** In contrast, AlphaEvolve and DeepEvolve show suboptimal generalization to user-centric metrics and, in some cases, even underperform the initial seed models. This suggests that scalar metric-only optimization is insufficient to capture the complex dynamics of the user experience. Without guidance, the evolved models tend to overfit to narrow numerical targets, which degrades overall user satisfaction.

4.1.1 Codebase Quality Evaluation

To verify substantive algorithmic improvements beyond numerical gains, we employ an LLM-as-a-judge [57, 26, 14] (averaged scores from GPT-5 and Claude Opus 4.6) to evaluate the recommender code in all evolved codebases of Table 1 on a 1-10 scale against the seed codebase, across four dimensions: **Creativity** (novel mechanisms beyond parameter tuning), **Explicitness** (interpretability of logic flows), **Insight** (intent to resolve specific failures, e.g., popularity bias), and **Personalization** (user-context awareness). Detailed prompts are provided in App. H.2. From the results in Figure 3, we have the following observations: **1)** Self-EvolveRec consistently achieves the highest scores across all criteria. The high scores on Creativity and Insight validate that our directional feedback loop integrating SIM and DIAG enables targeted logic refinement, whereas scalar-driven baselines remain limited to trial-and-error without diagnosing underlying issues. **2)** Self-EvolveRec shows a substantial gain in Personalization (+90% over baselines), driven by SIM reflecting specific user needs directly into algorithmic logic, while baselines optimize purely numerical metrics. **3)** AlphaEvolve underperforms across all criteria due to the absence of external knowledge, underscoring the necessity of retrieval (as in DeepEvolve and Self-EvolveRec) for open-ended discovery of novel mechanisms.



Figure 3: LLM-as-a-Judge evaluation of the evolved models.

4.2 Ablation Studies

To comprehensively evaluate the impact of each component in Self-EvolveRec, we conduct ablation studies in Table 3. Note that Row (6) represents the complete framework of Self-EvolveRec. We have the following observations: Across all datasets, **1)** Introducing either the SIM (Row (2)) or the DIAG (Row (3)) consistently yields performance gains over the scalar metric only method (Row (1)). Notably, the integration of the SIM results in a more substantial increase, underscoring its critical role in providing directional feedback by pinpointing specific recommendation failures, which scalar metrics fail to capture. **2)** Comparing Row (3) and Row (5) demonstrates that static diagnostic criteria limit the potential of code evolution. Since the model architecture continuously shifts, a fixed DIAG inevitably becomes obsolete. The performance gain in Row (5) implies that dynamically synchronizing the diagnostic logic of DIAG via co-evolution is essential for maintaining a reliable feedback loop. **3)** Integrating both the SIM and the DIAG (Row (4)), achieves superior performance among all ablated variants, notably outperforming even the co-evolved DIAG without user feedback (Row (5)). This underscores the importance of combining qualitative user critiques and quantitative diagnostics, which provide recommendation failures and internal structural issues. Furthermore, the complete framework of Self-EvolveRec (Row (6)) achieves the best performance, validating that dynamically aligning

Table 3: Ablation studies on the components of Self-EvolveRec (Seed Recommender: SASRec).

Row	Components	CDs		Electronics	
		NDCG@5	HR@5	NDCG@5	HR@5
(1)	Metric only	0.3610	0.4870	0.2508	0.3427
(2)	w/ SIM	0.3751	0.5102	0.2573	0.3551
(3)	w/ DIAG	0.3676	0.4791	0.2515	0.3449
(4)	w/ SIM, DIAG	0.3789	0.5164	0.2584	0.3566
(5)	w/ DIAG, Co-Evolve	0.3727	0.5014	0.2532	0.3520
(6)	w/ SIM, DIAG, Co-Evolve	**0.3865**	**0.5274**	**0.2600**	**0.3591**

Table 4: Performance comparison under extreme initialization scenarios (Random and Ensemble). Peak indicates the iteration number of the best performance.

Dataset	Seed Recommender		AlphaEvolve			DeepEvolve			**Ours**		
	NDCG@5	HR@5	NDCG@5	HR@5	Peak	NDCG@5	HR@5	Peak	NDCG@5	HR@5	Peak
Seed Recommender: Random											
CDs	0.0312	0.0525	0.3430	0.4549	15	0.3766	0.4963	13	**0.3883**	**0.5165**	**8**
Electronics	0.0310	0.0531	0.2037	0.2878	19	0.1972	0.2761	18	**0.2109**	**0.2952**	**11**
Seed Recommender: NCF+NGCF+SASRec											
CDs	0.3946	0.5179	0.3864	0.5075	FAIL	0.3695	0.5002	17	**0.4105**	**0.5409**	**9**
Electronics	0.2385	0.3246	0.2496	0.3386	14	0.2353	0.3240	FAIL	**0.2524**	**0.3426**	**6**

the diagnostic logic with both the shifting architecture and the qualitative feedback is necessary to ensure the efficacy of the feedback loop and the evolution of the model.

4.3 Model Analysis

4.3.1 Adaptability to extreme initialization scenarios

To further validate the adaptability of Self-EvolveRec, we evaluate its performance under two extreme initialization settings: a Random recommender representing a development starting from scratch, and a highly optimized Ensemble (NCF + NGCF + SASRec) reflecting a sophisticated industrial deployment. We have the following observations in Table 4: **1)** Even starting from a random recommender, Self-EvolveRec successfully evolves a fully functional and highly competitive recommender pipeline. Self-EvolveRec reaches the peak performance at the 8th (CDs) and the 11th (Electronics) iterations, significantly earlier than baselines (13 to 19 iterations). Furthermore, in terms of actual evolution time (excluding training) on the random recommender - CDs, Self-EvolveRec efficiently reaches peak performance in 2h 27m compared to DeepEvolve's 3h 2m. While AlphaEvolve is faster (1h 8m), it yields suboptimal performance. This confirms that directional feedback systematically constructs valid pipelines by resolving structural deficiencies, avoiding the aimless trial-and-error of scalar-driven methods. Further details regarding the evolution time required to reach peak performance are provided in Appendix G.2.1. **2)** In the ensemble setting, achieving further gains is exceedingly challenging due to the high initial performance and structural complexity of seed recommenders. We observe that scalar-only baselines, AlphaEvolve and DeepEvolve, struggle to navigate the structural complexity of the ensemble, even degrading the initial performance. In contrast, Self-EvolveRec consistently identifies and resolves latent bottlenecks within the ensemble, demonstrating that directional feedback is effective even in high-performance regimes, where numerical scores alone fail to provide guidance for optimizing such structurally complex systems.

These results highlight the adaptability of Self-EvolveRec across diverse industrial scenarios, ranging from the rapid cold-start of new services to the iterative refinement of high-performance models for established domains [3, 13, 52, 23].

4.3.2 Removing Feedback-aware Planning & Retrieval

To evaluate the contribution of the feedback-aware planning & retrieval introduced in Sec. 3.2, we compare Self-EvolveRec against a variant that excludes Planning & Retrieval (i.e., $\mathcal{R}_{\text{Dev}}$), denoted as **w.o. Planning** in Table 5. In this variant, the agent generates code directly based on the directional feedback through the code evolution step as $\mathcal{B}^{(t+1)} = \text{LLM}(\mathcal{I}_{\text{CODE}}, \mathcal{R}_{\text{SIM}}, \mathcal{R}_{\text{DIAG}}, \mathcal{B}_{\text{parent}}, \mathcal{P}^{(t)})$ instead of $\mathcal{B}^{(t+1)} = \text{LLM}(\mathcal{I}_{\text{CODE}}, \mathcal{R}_{\text{Dev}}, \mathcal{B}_{\text{parent}}, \mathcal{P}^{(t)})$. From the results in Table 5, we have the following observations: **1)** w.o. Planning achieves comparable recommendation performance to the original Self-EvolveRec. This implies that the primary driver of performance is the precise identification of failure modes via directional feedback, rather than the incorporation of external research ideas. **2)** However, the absence of planning significantly degrades codebase quality. While Insight remains relatively high (effectively identify what is wrong), w.o. Planning exhibits substantial drops in Creativity and Explicitness. This suggests that without planning, the agent relies on local heuristic patches rather than systematic, modular designs grounded in external knowledge. **3)** While DeepEvolve achieves high Creativity via external knowledge (RAG), w.o. Planning achieves higher Personalization. This confirms that the directional feedback from SIM, which captures specific user needs, is more crucial for personalization than generic knowledge retrievals.

4.3.3 Trustworthiness of the User Simulator

Since SIM's qualitative critiques drive the entire directional feedback loop, any factual hallucination—such as fabricated user preferences or interactions absent from $\mathcal{H}_u$—would propagate as misleading guidance throughout evolution. We therefore examine whether SIM's feedback is grounded in actual

Table 5: Performance when Feedback-aware Planning & Retrieval is removed (Seed Recommender: SASRec). C: Creativity, E: Explicitness, I: Insight, P: Personalization.

Method	CDs		Electronics		Codebase Quality			
	NDCG@5	HR@5	NDCG@5	HR@5	C	E	I	P
AlphaEvolve	0.3528	0.4623	0.2063	0.2891	4.5	5.0	5.5	3.0
DeepEvolve	0.3610	0.4870	0.2508	0.3427	7.5	6.5	7.5	3.0
Self-EvolveRec	0.3865	**0.5274**	**0.2600**	**0.3591**	**8.0**	**7.5**	**8.5**	**6.5**
w.o. Planning	**0.3988**	0.5134	0.2597	0.3568	6.5	5.0	8.0	5.0

Table 6: Evolving User Simulator. NDCG@5/HR@5 report the performance of evolved NCF, and Reliability denotes the accuracy of identifying the target item from 20 candidates by SIM.

Dataset	Strategy	NDCG@5	HR@5	SIM Type	Reliability
CDs	-	-	-	NCF (Base.)	0.2882
	Fixed SIM	0.2723	**0.3799**	Initial SIM	0.3473
	Evolved SIM	**0.2745**	0.3791	Evolved SIM	**0.3610**
Elec.	-	-	-	NCF (Base.)	0.2191
	Fixed SIM	**0.1907**	0.2714	Initial SIM	0.2238
	Evolved SIM	0.1891	**0.2744**	Evolved SIM	**0.2341**

user behavior. Following GISTBench [10], we measure **Interest Groundedness** (precision) on all simulation feedback generated along the trajectories in Table 1, achieving 0.891 (CDs), 0.843 (Electronics), and 0.851 (Office). These results confirm that SIM rarely fabricates preferences; its critiques are anchored in observable user history, ensuring that the directional feedback driving Self-EvolveRec reflects realistic user intent rather than LLM-induced artifacts. A full IG precision/recall analysis, together with a complementary discussion of how DIAG calibrates SIM's occasional semantic over-generalization (e.g., flagging same-artist exposure as redundancy), is provided in App. G.4.

4.3.4 Evolving User Simulator

Although our framework employs a fixed user simulator for efficiency, we further investigate the impact of evolving the user simulator alongside the codebase. Analogous to Sec. 3.3, the co-evolution of the simulator follows the same structured workflow of analysis, retrieval, and planning. Specifically, the LLM analyzes the current simulator and synthesizes a development plan $\mathcal{R}_{\text{Dev-SIM}}$ using retrieved methodologies $\mathcal{K}_{\text{SIM}}$, yielding the evolved simulator: $\text{SIM}^{(t+1)} = \text{LLM}(\mathcal{I}_{\text{Code-SIM}}, \mathcal{R}_{\text{Dev-SIM}}, \text{SIM}_{\text{parent}}, \mathcal{P}^{(t)})$. In Table 6, we evaluate simulator reliability by measuring the accuracy of identifying the target item from 20 candidates, utilizing a simple recommender model, NCF, as a baseline. While the recommendation performance remains comparable, the SIM's accuracy reveals the following observation: **1)** The comparable recommendation performance indicates that the initial SIM is already effective for modeling complex user preferences, achieving the Reliability superior to the NCF. This result indicates that the SIM provides reliable, high-quality directional feedback that captures complex user intent in decision making processes on given recommendation list. Consequently, even without evolving the simulator, the feedback signals from $\mathcal{R}_{\text{SIM}}$ are already robust enough to guide the Self-EvolveRec toward an optimal codebase. **2)** Nevertheless, evolution on SIM further elevates the simulator's reliability. The evolved SIM achieves an even higher Reliability compared to its initial state. This confirms that while the resulting recommendation scores are similar on the evolved codebase, the evolutionary process constructs a statistically more trustworthy feedback, ensuring that the directional feedback is grounded in realistic user behavior patterns.

4.3.5 Case Study: Validating Reliability of Co-evolved Diagnostic Tool

To verify the functional reliability and interpretability of the co-evolved DIAG, we conduct a case study through an induced deficiency experiment. Specifically, we evaluate whether the autonomously generated diagnostic metrics can accurately pinpoint logic-level deficiencies intentionally injected in the evolved models. As shown in Figure 4, we injected deficiencies into an evolved SASRec to force order-insensitivity by: (i) removing positional embeddings, (ii) bypassing the exponential decay module. The co-evolved DIAG successfully identifies these deficiencies via newly generated metrics, **Swap Sensitivity** (measuring ranking shifts after swapping last two items in the interaction sequence) and **Logit Δ Swap** (quantifies the deviation in predicted logits for the target item after the swap), which were not present in the initial seed DIAG. While the normal model exhibits high sensitivity to item order, the deficiency-injected model shows near-zero sensitivity, leading co-evolved DIAG correctly diagnosed as "Order-insensitive" in $\mathcal{R}_{\text{DIAG}}$. Notably, this diagnostic signal directly correlates with the sharp performance decline, demonstrating that co-evolved DIAG can explain the root causes of low performance through structural verification. These results demonstrate the capability to detect complex failure modes, highlighting the pivotal role of Diagnosis Tool - Model Co-Evolution in maintaining a grounded feedback loop. Additional case studies on MoRec and evolutionary trajectories are provided in App. G.5.1 and App. G.5.2, respectively.

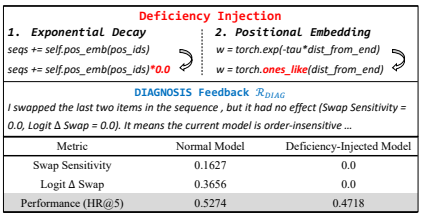


Figure 4: Case study on Diagnosis Tool - Model Co-Evolution on CDs dataset (Seed Recommender: SASRec).

5 Acknowledgements

This work was supported by National IT Industry Promotion Agency(NIPA) grant funded by the Korea government(MSIT) (AI-Based Dark Factory Design and Integration Solution, No. RS-2026-25621604), the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (RS-2026-25607361), and National Research Foundation of Korea(NRF) funded by Ministry of Science and ICT (RS-2022-NR068758).

6 Conclusion

In this paper, we propose Self-EvolveRec, a novel LLM-driven code evolution framework for automating the design of recommender systems. The main idea is to overcome the limitations of existing scalar metric-based code optimization by establishing a directional feedback loop that integrates qualitative critiques from a User Simulator with quantitative verification from a Model Diagnosis Tool. To ensure that this feedback loop remains valid as the recommendation pipeline evolves, we further introduce a Diagnosis Tool - Model Co-Evolution strategy that dynamically adapts diagnostic criteria to structural shifts in the codebase. Extensive experiments demonstrate that Self-EvolveRec significantly outperforms state-of-the-art NAS and LLM-driven evolutionary baselines in both standard recommendation accuracy and multi-faceted user satisfaction, while also exhibiting robust adaptability across diverse initialization settings – from scratch scenarios to complex ensembles – relevant to practical industrial deployment. While our agentic evaluation deliberately decouples optimization-time and evaluation-time simulators, we acknowledge that fully validating user-centric improvements ultimately requires real-world A/B testing, which we leave for future deployment studies.

References

[1] Abdelfattah, M. S., Mehrotra, A., Dudziak, Ł., and Lane, N. D. Zero-cost proxies for lightweight nas. *arXiv preprint arXiv:2101.08134*, 2021.

[2] Bougie, N. and Watanabe, N. Simuser: Simulating user behavior with large language models for recommender system evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 43–60, 2025.

[3] Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pp. 7–10, 2016.

[4] Choi, S., Kim, S., Kang, H., Kim, W., and Park, C. Dynamic time-aware continual user representation learning. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, pp. 740–749, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715921. doi: 10.1145/3726302.3729959. URL https://doi.org/10.1145/3726302.3729959.

[5] Ci, Y., Lin, C., Sun, M., Chen, B., Zhang, H., and Ouyang, W. Evolving search space for neural architecture search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6659–6669, 2021.

[6] Covington, P., Adams, J., and Sargin, E. Deep neural networks for youtube recommendations. RecSys '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

[7] Dong, Y., Cordonnier, J.-B., and Loukas, A. Attention is not all you need: Pure attention loses rank doubly exponentially with depth. In *International conference on machine learning*, pp. 2793–2803. PMLR, 2021.

[8] Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

[9] Fabbri, F., Penha, G., D'Amico, E., Wang, A., De Nadai, M., Doremus, J., Gigioli, P., Damianou, A., Stål, O., and Lalmas, M. Evaluating podcast recommendations with profile-aware llm-as-a-judge. In *Proceedings of the Nineteenth ACM Conference on Recommender Systems*, pp. 1181–1186, 2025.

[10] Fostiropoulos, I., Azhar, M. R., Sawwan, A., Fang, B., Liu, Y., Liu, J., Yu, H., Guo, Q., Wang, J., Liu, F., et al. Gistbench: Evaluating llm user understanding via evidence-based interest verification. *arXiv preprint arXiv:2603.29112*, 2026.

[11] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, H., and Wang, H. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2(1), 2023.

[12] Goldberg, L. R. The development of markers for the big-five factor structure. *Psychological Assessment*, 4:26–42, 1992. URL https://api.semanticscholar.org/CorpusID:144709415.

[13] Gomez-Uribe, C. A. and Hunt, N. The netflix recommender system: Algorithms, business value, and innovation. 6(4), 2016. ISSN 2158-656X. doi: 10.1145/2843948. URL https://doi.org/10.1145/2843948.

[14] Gu, J., Jiang, X., Shi, Z., Tan, H., Zhai, X., Xu, C., Li, W., Shen, Y., Ma, S., Liu, H., et al. A survey on llm-as-a-judge. *The Innovation*, 2024.

[15] Harper, F. M. and Konstan, J. A. The movielens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4), December 2015. ISSN 2160-6455. doi: 10.1145/2827872. URL https://doi.org/10.1145/2827872.

[16] He, X., Liao, L., Zhang, H., Nie, L., Hu, X., and Chua, T.-S. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pp. 173–182, 2017.

[17] He, X., Deng, K., Wang, X., Li, Y., Zhang, Y., and Wang, M. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 639–648, 2020.

[18] Hou, Y., Li, J., He, Z., Yan, A., Chen, X., and McAuley, J. Bridging language and items for retrieval and recommendation. *arXiv preprint arXiv:2403.03952*, 2024.

[19] Kang, W.-C. and McAuley, J. Self-attentive sequential recommendation. In *2018 IEEE international conference on data mining (ICDM)*, pp. 197–206. IEEE, 2018.

[20] Kim, D., Park, C., Oh, J., Lee, S., and Yu, H. Convolutional matrix factorization for document context-aware recommendation. RecSys '16, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959165. URL https://doi.org/10.1145/2959100.2959165.

[21] Kim, J., Kang, H., Kim, S., Kim, K., and Park, C. Disentangling and generating modalities for recommendation in missing modality scenarios. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, pp. 1820–1829, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715921. doi: 10.1145/3726302.3729953. URL https://doi.org/10.1145/3726302.3729953.

[22] Kim, K., Kim, S., Kang, H., Kim, J., Noh, H., In, Y., Yoon, K., Oh, J., and Park, C. Image is all you need: Towards efficient and effective large language model-based recommender systems. *arXiv preprint arXiv:2503.06238*, 2025.

[23] Kim, S., Lee, N., Kim, D., Yang, M., and Park, C. Task relation-aware continual user representation learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '23, pp. 1107–1119, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599516. URL https://doi.org/10.1145/3580305.3599516.

[24] Kim, S., Kang, H., Choi, S., Kim, D., Yang, M., and Park, C. Large language models meet collaborative filtering: An efficient all-round llm-based recommender system. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, pp. 1395–1406, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704901. doi: 10.1145/3637528.3671931. URL https://doi.org/10.1145/3637528.3671931.

[25] Kim, S., Kang, H., Kim, K., Kim, J., Kim, D., Yang, M., Oh, K., McAuley, J., and Park, C. Lost in sequence: Do large language models understand sequential recommendation? In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, pp. 1160–1171, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714542. doi: 10.1145/3711896.3737035. URL https://doi.org/10.1145/3711896.3737035.

[26] Kim, W., Park, S., In, Y., Kim, S., Lee, D., and Park, C. Beyond the final answer: Evaluating the reasoning trajectories of tool-augmented agents. *arXiv preprint arXiv:2510.02837*, 2025.

[27] Krishna, R., Kalaiah, A., Wu, B., Naumov, M., Mudigere, D., Smelyanskiy, M., and Keutzer, K. Differentiable nas framework and application to ads ctr prediction. *arXiv preprint arXiv:2110.14812*, 2021.

[28] Lee, J. and Ham, B. Az-nas: Assembling zero-cost proxies for network architecture search. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 5893–5903, 2024.

[29] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[30] Li, Z., Ji, J., Ge, Y., and Zhang, Y. Autolossgen: Automatic loss function generation for recommender systems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1304–1315, 2022.

[31] Liu, B., Zhu, C., Li, G., Zhang, W., Lai, J., Tang, R., He, X., Li, Z., and Yu, Y. Autofis: Automatic feature interaction selection in factorization models for click-through rate prediction. In *proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2636–2645, 2020.

[32] Liu, F., Lin, X., Yu, H., Wu, M., Wang, J., Zhang, Q., Zhao, Z., Xia, Y., Zhang, Y., Li, W., et al. Recoworld: Building simulated environments for agentic recommender systems. *arXiv preprint arXiv:2509.10397*, 2025.

[33] Liu, G., Zhu, Y., Chen, J., and Jiang, M. Scientific algorithm discovery by augmenting alphaevolve with deep research. *arXiv preprint arXiv:2510.06056*, 2025.

[34] Ma, C., Xu, Z., Ren, Y., Hettiachchi, D., and Chan, J. Pub: an llm-enhanced personality-driven user behaviour simulator for recommender system evaluation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2690–2694, 2025.

[35] Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[36] McNee, S. M., Riedl, J., and Konstan, J. A. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pp. 1097–1101, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595932984. doi: 10.1145/1125451.1125659. URL https://doi.org/10.1145/1125451.1125659.

[37] Mezghani, M., Zayani, C. A., Amous, I., and Gargouri, F. A user profile modelling using social annotations: a survey. WWW '12 Companion, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312301. doi: 10.1145/2187980.2188230. URL https://doi.org/10.1145/2187980.2188230.

[38] Naumov, M., Mudigere, D., Shi, H.-J. M., Huang, J., Sundaraman, N., Park, J., Wang, X., Gupta, U., Wu, C.-J., Azzolini, A. G., et al. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091*, 2019.

[39] Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[40] Real, E., Liang, C., So, D., and Le, Q. Automl-zero: Evolving machine learning algorithms from scratch. In *International conference on machine learning*, pp. 8007–8019. PMLR, 2020.

[41] Roccas, S., Sagiv, L., Schwartz, S. H., and Knafo, A. The big five personality factors and personal values. *Personality and Social Psychology Bulletin*, 28:789 – 801, 2002. URL https://api.semanticscholar.org/CorpusID:144611052.

[42] Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J., Ellenberg, J. S., Wang, P., Fawzi, O., et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[43] Roy, O. and Vetterli, M. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pp. 606–610. IEEE, 2007.

[44] Salakhutdinov, R. and Mnih, A. Probabilistic matrix factorization. NIPS'07, Red Hook, NY, USA, 2007. Curran Associates Inc. ISBN 9781605603520.

[45] Sharma, A. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

[46] Song, Q., Cheng, D., Zhou, H., Yang, J., Tian, Y., and Hu, X. Towards automated neural interaction discovery for click-through rate prediction. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 945–955, 2020.

[47] Steck, H. Calibrated recommendations. In *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, pp. 154–162, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359016. doi: 10.1145/3240323.3240372. URL https://doi.org/10.1145/3240323.3240372.

[48] Sun, F., Liu, J., Wu, J., Pei, C., Lin, X., Ou, W., and Jiang, P. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1441–1450, 2019.

[49] Tanese, R., Holland, J. H., and Stout, Q. F. *Distributed genetic algorithms for function optimization*. PhD thesis, USA, 1989. AAI9001722.

[50] Tang, J. and Wang, K. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pp. 565–573, 2018.

[51] Wang, X., He, X., Wang, M., Feng, F., and Chua, T.-S. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pp. 165–174, 2019.

[52] Wang, X., Huzhang, G., Lin, Q., and Da, Q. Learning-to-ensemble by contextual rank aggregation in e-commerce. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, pp. 1036–1044, 2022.

[53] Yuan, Z., Yuan, F., Song, Y., Li, Y., Fu, J., Yang, F., Pan, Y., and Ni, Y. Where to go next for recommender systems? id- vs. modality-based recommender models revisited. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2639–2649, 2023.

[54] Zhang, A., Chen, Y., Sheng, L., Wang, X., and Chua, T.-S. On generative agents in recommendation. In *Proceedings of the 47th international ACM SIGIR conference on research and development in Information Retrieval*, pp. 1807–1817, 2024.

[55] Zhang, S., Wang, M., Zhao, X., Guo, R., Zhao, Y., Zhuang, C., Gu, J., Zhang, Z., and Yin, H. Dns-rec: Data-aware neural architecture search for recommender systems. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pp. 591–600, 2024.

[56] Zhang, T., Cheng, D., He, Y., Chen, Z., Dai, X., Xiong, L., Yan, F., Li, H., Chen, Y., and Wen, W. Nasrec: weight sharing neural architecture search for recommender systems. In *Proceedings of the ACM Web Conference 2023*, pp. 1199–1207, 2023.

[57] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

[58] Zheng, R., Qu, L., Cui, B., Shi, Y., and Yin, H. Automl for deep recommender systems: A survey. *ACM Transactions on Information Systems*, 41(4):1–38, 2023.

[59] Ziegler, C.-N., McNee, S. M., Konstan, J. A., and Lausen, G. Improving recommendation lists through topic diversification. In *Proceedings of the 14th International Conference on World Wide Web*, WWW '05, pp. 22–32, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930469. doi: 10.1145/1060745.1060754. URL https://doi.org/10.1145/1060745.1060754.

[60] Zoph, B. and Le, Q. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=r1Ue8Hcxg.

[61] Zou, K. and Sun, A. A survey of real-world recommender systems: Challenges, constraints, and industrial perspectives. *arXiv preprint arXiv:2509.06002*, 2025.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes], [No], or [N/A].
- [N/A] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for [N/A]).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will also be asked to include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While [Yes] is generally preferable to [No], it is perfectly acceptable to answer [No] provided a proper justification is given (e.g., error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering [No] or [N/A] is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS Paper Checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Yes, we have clearly described the paper's contributions and scope in the abstract and introduction.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We have described future work related to paper's limitations in the Sec. 6.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.

- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not include theoretical assumptions.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We include the source code for our proposed work via an external link, along with the implementation details. Additionally, we include the prompts for LLMs in App. H.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.

- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: All datasets are open-access, and we have provided the pre-processed data along with the source code.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

Justification: We have described the evaluation protocol and implementation details in Sec. 4.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: Due to the substantial API costs associated with the iterative model evolution process, we do not conduct repeated experimental runs. Instead, we demonstrate the robustness of our claims by evaluating our method and baselines across various seed recommenders and multiple datasets. Furthermore, experiments involving different LLM backbones are detailed in App. G.1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We include details regarding GPU and LLM resources in the Implementation Details (Sec. 4 and App. C).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our work strictly follows the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

    Justification: As our work focuses on the automatic refinement of recommender pipelines, it does not raise any societal impact concerns.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: This paper does not pose any risks related to this issue.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.

- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have properly cited all relevant literature and data sources.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide detailed and well-documented documented source code via an external repository link, alongside full implementation details in App. C.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This research does not involve human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This research does not involve human participants.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Our work relies on LLMs as core components for LLM-driven code evolution. We provide the detailed prompts utilized in App. H. Furthermore, we validated our method across different LLMs in App. G.1, and conducted experiments in App. G.4 to investigate the hallucination issues arising from LLM dependency.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.

A Related Work

Neural Architecture Search for Recommender Systems. Neural Architecture Search (NAS) has been increasingly explored in recommender systems to automate the design of feature interactions and model architectures. Early studies like AutoFIS [31] replace discrete interaction feature choices with learnable gating to filter redundant interactions. Later methods extend NAS to backbone design. AutoCTR [46] uses evolutionary search to assemble operator blocks into a DAG, whereas DNAS [27], NASRec [56] and DNS-Rec [55] leverage weight-sharing supernets for scalability. Automation has also expanded to non-architectural components. AutoLossGen [30] casts loss function design as an automated search problem, using reinforcement learning (RL) to explore loss formulations composed of basic mathematical operators (e.g., addition, log, multiplication). Nevertheless, these approaches remain bounded by pre-defined operator sets and wiring rules, restricting innovation to selection/parameterization within a fixed search space rather than open-ended synthesis of new procedural algorithms.

LLM-driven Code Evolution. To overcome the limitations of fixed search spaces, recent work has explored LLM-driven code evolution, which shifts the optimization target from parameters to open-ended programs. FunSearch [42] utilizes an LLM-evaluator loop to discover interpretable algorithms for mathematical tasks, while Eureka [35] automates RL reward engineering by iteratively refining code based on execution feedback. Building on this line, AlphaEvolve [39] generalizes the evolutionary loop into an autonomous coding pipeline that iteratively edits and tests code using evaluator feedback, scaling to more complex algorithmic optimization tasks. More recently, DeepEvolve [33] integrates retrieval-augmented generation, leveraging external knowledge to systematically inform hypothesis generation and implementation for scientific discovery tasks. However, most existing approaches are guided primarily by a single scalar metric (e.g., accuracy or success rate), which provides limited diagnostic insight into user-centric failure modes such as bias, off-topic, or lack of diversity, which are key considerations for holistic recommender-system optimization.

LLM-based User Simulation. LLM-based user simulation has emerged as a promising alternative to static recommendation metrics and costly online A/B testing. Initial approaches focused on realistic persona construction. Agent4Rec [54] models social traits like conformity from real-world data, while Profile-aware simulators [9] utilize natural language summaries of user history to align with human judgment. Enhancing psychological fidelity, PUB [34] further integrates Big Five personality traits [12, 41] to replicate diverse interaction patterns. Recent research shifts focus to utilizing user simulators for system optimization, and SimUSER [2] identifies self-consistent personas with specialized perception and memory modules to serve as believable human proxies. Meanwhile, RecoWorld [32] establishes a proactive feedback loop, where the simulator explicitly signals user states (e.g., boredom) to guide the recommender's adaptation.

B Traits

Activity (Engagement Level). Activity quantifies the degree of a user's engagement with the recommender system. Since user-item interactions are typically sparse, we define activity as the cardinality of the user's interaction history:

$$T_{\text{act}}(u) = |\mathcal{H}_u|. \tag{2}$$

Users with lower $T_{\text{act}}(u)$ values correspond to passive users who interact infrequently with recommended items, whereas higher values indicate highly engaged users with rich interaction histories.

Conformity (Mainstream Adherence). Conformity measures the extent to which a user's preferences align with global public consensus. This trait captures whether a user follows mainstream tastes or exhibits individualized preferences. We define conformity as the mean squared deviation between the user's rating $r_{u,v}$ and the global average rating $\bar{r}_v$ of item v:

$$T_{\text{conf}}(u) = \frac{1}{|\mathcal{H}_u|} \sum_{v \in \mathcal{H}_u} (r_{u,v} - \bar{r}_v)^2, \tag{3}$$

where $\bar{r}_v$ denotes the average rating of item v across all users. A lower conformity value implies that the user's preferences closely align with popular sentiment, while a higher value reflects more distinctive and personalized tastes.

Diversity (Interest Breadth). Diversity characterizes the breadth of a user's interests across item categories. We define this trait as the number of unique categories associated with the items in the user's interaction history:

$$T_{\text{div}}(u) = \big|\{c_v \mid v \in \mathcal{H}_u\}\big|. \tag{4}$$

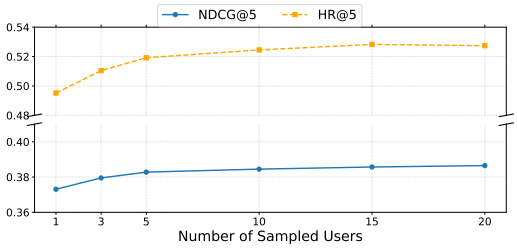


Figure 5: Recommendation performance over number of sampled user $\mathcal{U}_{\text{sample}}$ on CDs dataset (Seed Recommender: SASRec).

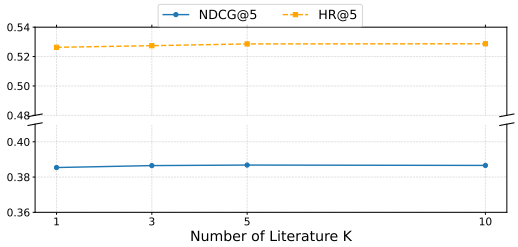


Figure 6: Recommendation performance over number of literature $\mathcal{K}$ on CDs dataset (Seed Recommender: SASRec).

Users with lower $T_{\text{div}}(u)$ values tend to focus on a narrow set of categories, whereas higher values indicate a preference for exploring a broader and more diverse range of categories.
We categorized each user trait into three distinct levels, defined as follows:

- **Activity**:
  - HIGH: Frequently interacts with the system and maintains a high volume of engagement with recommendations.
  - MID: Interacts moderately, primarily when items strictly align with personal preferences.
  - LOW: Rarely interacts with the system and does not interact if recommendations are not relevant to their interests.
- **Conformity**:
  - HIGH: Heavily influenced by popularity and public ratings; tends to follow mainstream trends.
  - MID: Considers both popularity and personal taste, balancing trends with individual preferences.
  - LOW: Ignores popularity and trends, evaluating items purely based on intrinsic personal preference.
- **Diversity**:
  - HIGH: Seeks high variety and novelty, enjoying the exploration of diverse categories and new styles.
  - MID: Mostly consumes preferred categories but occasionally explores similar alternatives.
  - LOW: Sticks strictly to a narrow set of familiar categories and avoids exploration.

C Implementation Details

Regarding the baseline implementation of AlphaEvolve, due to the unavailability of the official code, we utilized OpenEvolve [45], an open-source implementation, following prior work [33]. In our evolutionary framework, we set the maximum evolution steps to 21 across all LLM-driven evolutionary frameworks. For all LLM-driven evolutionary frameworks, we set the maximum evolution iterations to 21. For NAS baselines, we configured the search epochs to 5 for AutoFIS and 1 for NASRec, following the hyper-parameter setting in NASRec [56]. We employ GPT-5-mini for the User Simulator (SIM) and set the number of sampled users to $|\mathcal{U}_{\text{sample}}| = 20$ (refer to App. G.2.2 for an analysis of the number of sampled users). To ensure a fair comparison, we uniformly configured all recommender models—including the retraining phase of NAS models—with a user/item embedding dimension of 50, a batch size of 128, and a learning rate of 0.001. The maximum number of epochs was set to 300 for both standard training and the NAS retraining stage. All experiments were conducted on a single NVIDIA GeForce A6000 (48GB) GPU.

D Datasets

Table 7 shows the statistics of the dataset after preprocessing.

E Baselines and Seed Recommender

E.1 Baselines

(1) Neural Architecture Search Baselines

Table 7: Statistics of datasets after preprocessing.

Dataset	CDs	Electronics	Office	MovieLens
# Users	14,335	32,232	20,147	6,040
# Items	11,436	17,695	10,470	3,952
# Interactions	126,225	257,850	145,927	1,000,209

- AutoFIS [31] automatically identifies essential feature interactions by employing learnable gates to prune redundant feature combinations.
- NASRec [56] leverages a weight-sharing supernet to efficiently search for optimal full architectures.
- AutoLossGen [30] automatically searches for effective loss functions tailored to different recommendation models and datasets.
- DNS-Rec [55] searches compact architectures for sequential recommender systems under data-aware and resource-aware constraints.

(2) LLM-driven Code Evolution Baselines

- AlphaEvolve [39] orchestrates an autonomous LLM-driven evolutionary pipeline to iteratively evolve and optimize algorithmic codebases.
- DeepEvolve [33] integrates retrieval-augmented generation (RAG) into the evolutionary loop to guide code optimization with external scientific knowledge.

E.2 Seed Recommender

- NCF [16] is a pioneering neural collaborative filtering framework which combines multi-layer perceptron (MLP) to learn user-item interactions.
- NGCF [51] is a graph-based model that explicitly encodes collaborative signals in the embedding space by propagating embeddings on the user-item bipartite graph
- SASRec [19] is a sequential recommendation model leveraging self-attention mechanisms to dynamically capture user interests from interaction sequences.
- MoRec [53] is a multi-modal framework that utilizes pre-trained encoders (e.g., SBERT) to initialize item embeddings with textual features. Following prior work [24], we employ SASRec as the backbone architecture for MoRec.

F Detailed Implementation of the Efficient Variant

In this appendix, we provide the precise formulation of the zero-cost signals introduced in Sec. 3.4, along with the corresponding adaptations of the Diagnosis Tool (DIAG) and the User Simulator (SIM).

F.1 Detailed Formulation of Zero-Cost Signals.

Let $U \in \mathbb{R}^{B\times d}$ and $P \in \mathbb{R}^{B\times d}$ denote the user and positive item representation matrices from a single forward pass on a mini-batch, where B is the batch size and d is the embedding dimension. Let $I = U \odot P \in \mathbb{R}^{B\times d}$ be the interaction representation, and let $\{\sigma_i(M)\}$ denote the singular values of a matrix M in decreasing order. **1)** The effective ranks ie and ue [43] are defined as the exponentiated Shannon entropy of the normalized singular value distribution:

$$\text{erank}(M) = \exp\left(-\sum_i p_i(M)\log p_i(M)\right), \quad p_i(M) = \frac{\sigma_i(M)}{\sum_j \sigma_j(M)}, \tag{5}$$

with $ie = \text{erank}(I)/min(B,d)$ and $ue = \text{erank}(U)/min(B,d)$. Both take values in $[0,1]$, and a higher value indicates that the representation occupies more independent dimensions. **2)** The top-1 singular value share $sv = p_1(I) = \sigma_1(I)/\sum_j \sigma_j(I)$ measures the fraction of total spectral energy concentrated in the largest singular value of I, directly diagnosing rank collapse [7]. **3)** The mean parameter gradient norm g is computed by performing a loss function on the mini-batch, backpropagating to obtain $\nabla_\theta \mathcal{L}$, and averaging the per-parameter gradient norms. This serves as a proxy for the model's initial trainability and information flow [1]. Finally, we design the proxy $\mathcal{M}_{\text{zero}} = ie + ue - sv - |\log g + 3|$ as a surrogate for the standard recommendation metric (e.g., NDCG).

Table 9: Comparison of average execution time per iteration on CDs dataset (Seed Recommender: SASRec).

Method	RAG	Coding	Co-Evolve	SIM	DIAG	Total Time
AlphaEvolve	-	6m 23s	-	-	-	6m 23s
DeepEvolve	6m 16s	7m 43s	-	-	-	13m 59s
Ours	6m 24s	7m 15s	4m 46s	44s*	12s	19m 21s

* The User Simulation time is measured with 20 sampled users, processed in parallel. Diagnosis Co-Evolve includes both research (2m 12s) and coding (2m 34s).

F.2 Adaptation of the Diagnosis Tool (DIAG).

The standard DIAG (Sec. 3.1.2) relies on post-training probes such as embedding collapse, which require fully trained model parameters and therefore cannot be computed under the zero-cost regime. To resolve this, the seed diagnosis tool $\text{DIAG}^{(0)}$ in the efficient variant directly leverages the individual sub-signals of $\mathcal{M}_{\text{zero}}$ as foundational probes for assessing structural health: ie and ue serve as proxies for representation diversity, sv for rank collapse detection, and g for trainability assessment. As the codebase evolves, the Diagnosis Tool - Model Co-Evolution mechanism (Sec. 3.3) continues to update these probes accordingly.

F.3 Adaptation of the User Simulator (SIM).

The original SIM (Sec. 3.1.1) requires explicit recommendation lists $\mathcal{A}^u$ from a fully trained model to evaluate user-perceived quality. Since $\mathcal{A}^u$ is unavailable without full training, we adapt SIM via a *code-to-intent* summarization strategy, where an LLM summarizes the current codebase $\mathcal{B}^{(t)}$ into a high-level architectural intent. This intent is provided to SIM together with the zero-cost signals, allowing SIM to assess whether the proposed structural changes align with the user's persona $\mathcal{T}_u$ and history $\mathcal{H}_u$, and to generate qualitative feedback $\mathcal{R}_{\text{SIM}}$ (e.g., *"Please capture full multi-category item signals more."*).

G Additional Experiments

G.1 Adaptability to Different LLMs

Table 8: Performance comparison of different LLMs.

Seed Model	LLMs	Metric	CDs			Electronics		
			A	D	**Ours**	A	D	**Ours**
SASRec	GPT-5	NDCG@5	0.3528	0.3610	**0.3865**	0.2063	0.2508	**0.2600**
		HR@5	0.4623	0.4870	**0.5274**	0.2891	0.3427	**0.3591**
	Claude	NDCG@5	0.3502	0.3659	**0.3913**	0.2176	0.2470	**0.2562**
		HR@5	0.4542	0.4901	**0.5170**	0.3056	0.3394	**0.3578**
NCF	GPT-5	NDCG@5	0.2556	0.2421	**0.2723**	0.1183	0.1817	**0.1907**
		HR@5	0.3477	0.3497	**0.3799**	0.1733	0.2600	**0.2714**
	Claude	NDCG@5	0.2485	0.2478	**0.2709**	0.1218	0.1854	**0.2018**
		HR@5	0.3311	0.3468	**0.3724**	0.1844	0.2652	**0.2844**

To verify the robustness of Self-EvolveRec across different LLMs, we conducted an additional experiment utilizing Claude Opus 4.6 instead of the GPT-5 series used in our main pipeline. From the results in Table 8, we have the following observations: **1)** When utilizing Claude as the backbone LLM, Self-EvolveRec consistently outperforms the baselines (AlphaEvolve and DeepEvolve). This demonstrates that our core methodology of identifying and resolving structural deficiencies via the User Simulator and Model Diagnosis Tool is fundamentally effective. **2)** Furthermore, we observe negligible performance difference between the GPT- and Claude-based variants of Self-EvolveRec. This confirms that our evolutionary paradigm, which is grounded in diagnosing the root causes of model failure, is highly robust and genuinely LLM-agnostic, rather than being an artifact of a specific model's capabilities.

G.2 Efficiency Analysis

G.2.1 Time Efficiency Analysis

We compare the execution time per iteration of evolution in Table 9. The reported times are averaged over the evolution of SASRec on the CDs dataset. We have the following observations: **1)** Although Self-EvolveRec requires approximately 19 minutes per iteration—higher than AlphaEvolve (6m) and DeepEvolve (14m)—it significantly reduces the *total number of iterations* required to reach peak performance. As illustrated in Table 4, Self-EvolveRec reaches its peak in just around 8 to 11 iterations, whereas baselines relying on aimless trial-and-error require 13 to 19 iterations or fail to outperform the initial model. This substantial reduction in required iterations ensures that Self-EvolveRec reaches its peak performance highly efficiently. For instance, when evaluated on the random recommender setting for the Electronics dataset, Self-EvolveRec achieves peak performance in 3h 27m, significantly outpacing DeepEvolve (4h 19m). Similarly, for SASRec on the Office dataset, Self-EvolveRec takes only 1h 54m to reach its peak compared to DeepEvolve's 3h 47m. In both

cases, while AlphaEvolve is faster (2h 10m and 1h 32m, respectively), it yields substantially lower performance. **2)** While integrating RAG processes in DeepEvolve and Self-EvolveRec increases the runtime per iteration compared to AlphaEvolve, it is critical for ensuring the quality of the evolved logic. As discussed in Sec. 4.1.1, the absence of external knowledge limits AlphaEvolve's ability to discover novel mechanisms, resulting in lower *Creativity* and *Insight*. Conversely, Self-EvolveRec leverages this RAG latency to incorporate external algorithmic knowledge and diagnostic signals, enabling the discovery of novel mechanisms that scalar-driven baselines fail to achieve. **3)** The User Simulator introduces an additional time cost (44s), yet it is crucial for sustaining high performance. Furthermore, as detailed in App. G.2.2, the simulator remains stable and effective even with a small number of sampled users, ensuring efficiency without compromising robustness.

G.2.2 Impact of user sampling size.

To investigate the efficiency and robustness of Self-EvolveRec on $\mathcal{U}_{\text{sample}}$, we conducted experiments by varying the user sample size of the User Simulator. As shown in Figure 5, we observe that Self-EvolveRec achieves robust performance even with a small number of sampled users. Notably, performance improves significantly as the sample size increases from 1 to 3, eventually stabilizing around a sample size of 5. This robustness stems from the **Diagnosis Tool - Model Co-Evolution** mechanism (Sec. 3.3). Although the SIM operates on a small subset of users to generate qualitative feedback (e.g., "lack of diversity"), the DIAG translates these critiques into numerically-grounded metrics (e.g., measuring category entropy). Consequently, even with limited user samples, the evolved DIAG effectively verifies and quantifies structural deficiencies across the global data distribution, ensuring reliable evolutionary guidance without the need for extensive user sampling.

G.2.3 Impact of the size of literature.

To investigate the effect of literature size (i.e., $\mathcal{K}$) on Self-EvolveRec, we vary the number of literature utilized in the Planning (Section 3.2). As shown in Figure 6, we observe that Self-EvolveRec maintains robust performance across different size of $\mathcal{K}$. This robustness stems from the directional feedback provided by Self-EvolveRec, which effectively pinpoints issues within the recommender pipeline. Consequently, even without searching a massive amount of external knowledge, Self-EvolveRec can accurately diagnose and correct the problems in the current recommender.

G.2.4 Efficiency Analysis under Fixed Token Budget

To ensure a fair comparison regarding computational costs, we evaluate all evolutionary frameworks (i.e., AlphaEvolve, DeepEvolve, and Self-EvolveRec) under a fixed token budget. Specifically, we constrain the total number of tokens (including both input prompts and completions) utilized throughout the entire evolutionary process to 5M tokens. As shown in Table 10, we observe that Self-EvolveRec consistently outperforms the baselines even under a fixed token budget. This superiority stems from the ability of Self-EvolveRec to leverage directional feedback to pinpoint specific failures within the recommendation pipeline and effectively resolve them. Consequently, Self-EvolveRec achieves more effective evolution within the same resource constraints by avoiding the aimless trial-and-error search characteristic of scalar-driven methods.

Table 10: Performance comparison under a fixed token budget (Seed Recommender: SASRec).

Method	CDs		Electronics	
	NDCG@5	HR@5	NDCG@5	HR@5
AlphaEvolve	0.3528	0.4623	0.1995	0.2785
DeepEvolve	0.3539	0.4809	0.2415	0.3352
Self-EvolveRec	**0.3843**	**0.5209**	**0.2472**	**0.3580**

G.2.5 Efficiency via Zero-Cost Proxies

To validate the efficiency of the variant in Sec. 3.4, we compare the zero-cost proxy variant of Self-EvolveRec against its fully-trained pipeline in Table 11. We have the following observations: **1)** The zero-cost proxy variant achieves comparable recommendation performance to the fully-trained pipeline, with only marginal degradation. **2)** The variant substantially reduces the per-iteration computational overhead, achieving up to 12× speedup (e.g., 12m 08s → 1m 02s on MoRec/CDs). These results confirm that the efficient variant offers a practical trade-off, retaining most of the recommendation quality while drastically reducing the computational cost of evolution.

Table 11: Performance and efficiency of zero-cost proxy variant. Time denotes average duration per iteration for training and directional feedback generation.

Seed	Dataset	Type	NDCG@5	HR@5	Time
SASRec	CDs	Original	**0.3865**	**0.5274**	2m 58s
		Zero-Cost	0.3798	0.5054	**36s**
	Elec.	Original	**0.2600**	**0.3591**	3m 54s
		Zero-Cost	0.2531	0.3485	**39s**
MoRec	CDs	Original	**0.3977**	**0.5340**	12m 08s
		Zero-Cost	0.3827	0.4991	**1m 02s**
	Elec.	Original	**0.2056**	**0.2921**	7m 10s
		Zero-Cost	0.1964	0.2869	**1m 13s**

Table 12: Recommendation performance under different persona schemes for the User Simulator (Seed Recommender: SASRec).

Persona Scheme	CDs		Electronics	
	NDCG@5	HR@5	NDCG@5	HR@5
Activity / Conformity / Diversity (Ours)	0.3865	**0.5274**	**0.2600**	**0.3591**
Big Five	**0.3915**	0.5244	0.2551	0.3561

G.3 Robustness to Persona Scheme

To verify that Self-EvolveRec is not tied to a specific user characterization, we replace the default traits (Activity, Conformity, Diversity) with the Big Five personality traits (Openness, Conscientiousness, Extraversion, Agreeableness, and Neuroticism), following prior work [34]. As shown in Table 12, Self-EvolveRec achieves comparable recommendation performance under both persona schemes. This is consistent with our observation in Sec. 4.3.4 that SIM already captures complex user preferences reliably, and indicates that the driving factor of evolution is the precise, qualitative nature of $\mathcal{R}_{\text{SIM}}$ rather than the specific trait taxonomy used to construct it. These results confirm that Self-EvolveRec is robust to variations in user characterization.

G.4 Hallucinations and Groundedness of the User Simulator

To examine whether the User Simulator produces factual hallucinations, such as fabricated items, interactions, or user preferences, we analyzed all simulation feedback generated along the evolutionary trajectories in Table 1. However, we found no cases in which the simulator made claims contradicted by the actual user histories or recommendation lists.

To further quantify this groundedness, we adopt the Interest Groundedness (IG) metric from GIST-Bench [10], which evaluates whether an LLM-inferred user interest is supported by evidence from the user's interaction history, rather than merely sounds plausible. Following GISTBench, we define $\mathcal{I}_{pred}$ as the set of interest categories inferred from the simulator's feedback (e.g., item title or category preference), and $\mathcal{I}_{true}$ as the set verified from the user's historical interactions. Since our recommendation datasets contain relatively sparse user histories, we treat an interest category as verified when it is supported by at least one explicit positive interaction with a rating of 4 or higher. We compute IG precision and recall as:

$$IG_{Precision} = \frac{|\mathcal{I}_{pred} \cap \mathcal{I}_{true}|}{|\mathcal{I}_{pred}|}, \quad IG_{Recall} = \frac{|\mathcal{I}_{pred} \cap \mathcal{I}_{true}|}{|\mathcal{I}_{true}|} \tag{6}$$

Following GISTBench, all metrics are computed per user and aggregated as the median across users for each evolutionary trajectory, and then averaged across trajectories. Here, $IG_{Precision}$ directly captures the absence of hallucinations by measuring whether the simulator's inferred interests are grounded in evidence from the user's history, while IG_{Recall} measures the coverage of interests verifiable from the user's history.

As shown in Table 13, the simulator achieves high $IG_{Precision}$ on both CDs and Electronics, confirming that nearly all inferred interests are grounded in explicit positive evidence and the simulator does not hallucinate user preferences. The IG_{Recall} scores further show that it covers a substantial portion of verifiable interests, suggesting that simulator feedback is largely grounded in observed user behavior.

Table 13: Performance of Interest Groundedness.

Dataset	$IG_{Precision}$	IG_{Recall}
CDs	0.891	0.749
Electronics	0.843	0.578
Office	0.851	0.570

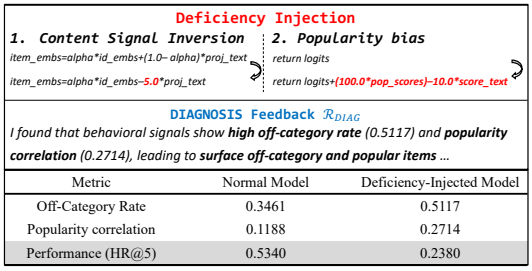
Complementary Interaction of SIM and DIAG.

While high IG scores confirm that the simulator is free from factual hallucinations, they do not guarantee that its critiques are always technically precise or aligned with the model's internal state, necessitating calibration via DIAG to distinguish between surface-level complaints and actual structural failures. Specifically, **1)** the simulator occasionally over-generalizes semantic overlaps. For instance, seeing multiple albums by the same artist (e.g., Adele) might lead the simulator to harshly criticize the model for "severe redundancy," even though each item is a valid historical interest. DIAG recalibrates this by verifying that the duplicate exposure rate is near-zero and items are geometrically distinct, proving the critique is a semantic over-generalization rather than a structural representation collapse. **2)** The simulator lacks structural awareness. In our trajectory investigation, it often repeated the same semantic complaints even as the model's geometric health (e.g., reduced embedding collapse) was objectively resolving. This confirms that DIAG is essential not just for verifying the absence of hallucinations, but for bridging the gap between the simulator's user-centric perception and the model's actual architectural progress.

G.5 Additional Case Studies

G.5.1 Addtional Case Study: Reliability of Co-evolved Diagnostic Tool via Deficiencies Injection.

In case Figure 7, we injected deficiencies into an evolved MoRec pipeline by: (i) inverting content signals, and (ii) inflating popularity scores. The co-evolved DIAG effectively detects these deficiencies through newly formulated metrics such as the **Off-Category Rate** (Mismatch rate between the categories of the Top-K recommended items and the categories present in user's recent interaction history) and **Popularity correlation** (Correlation between the model's predicted recommendation logits and the item popularity distribution). The $\mathcal{R}_{DIAG}$ accurately reports that the system has begun to "surface off-category or popular items rather than reliably relevant ones" providing clear directional feedback for subsequent correction. Notably, the strong association between these detected deficiencies and the performance drop confirms that the co-evolved metrics can effectively bridge the gap between internal behavioral shifts and external recommendation accuracy.



Metric	Normal Model	Deficiency-Injected Model
Off-Category Rate	0.3461	0.5117
Popularity correlation	0.1188	0.2714
Performance (HR@5)	0.5340	0.2380

Figure 7: Case study on Diagnosis Tool - Model Co-Evolution on CDs dataset (Seed Recommender: MoRec).

G.5.2 Additional Case Study: Evolutionary Trajectory.

Evolutionary Trajectory on SASRec: CDs dataset.

To validate the effectiveness of the directional feedback loop, we conducted a case study comparing the evolutionary trajectories of Self-EvolveRec against baselines. Figure 8 (a) illustrates the step-by-step evolution of codebases, while Figure 8 (b) tracks the performance progress over all iterations. We have the following observations: **1)** Self-EvolveRec shows a structured evolutionary path, where algorithmic improvements are causally linked to identified failures. For instance, in Iteration $(0 \rightarrow 1)$, $\mathcal{R}_{\text{SIM}}$ explicitly flagged "Ignores Subgenre", while $\mathcal{R}_{\text{DIAG}}$ detected "High Embedding Collapse." Guided by this directional feedback, the agent introduced "Category-Aware Hard Negatives" and "Popularity-annealed Weighting," resulting in an immediate performance increment (HR: $0.4676 \rightarrow 0.5138$). **2)** In contrast, baselines exhibit unstable or delayed progress due to their reliance on scalar metrics. AlphaEvolve attempts an erroneous combination of loss functions (BCE + BPR) at Iteration 3, causing a significant performance drop, which is only rectified by removing the module at Iteration 7. Consequently, it does not exceed its initial performance state throughout the evolution. DeepEvolve suffers from prolonged stagnation in low-performance regions, as evident in Figure 8 (b), and only manages a gain at Iteration 15 by retrieving the 'Refined-LogQ' module. Due to such inefficient exploration, as observed in Figure 8 (b), both baselines remain trapped in low-performance regions ($HR@5 : 0.05 - 0.25$), failing to escape suboptimal states throughout the evolution process. Conversely, Self-EvolveRec leverages directional feedback to maintain a robust evolutionary trajectory.

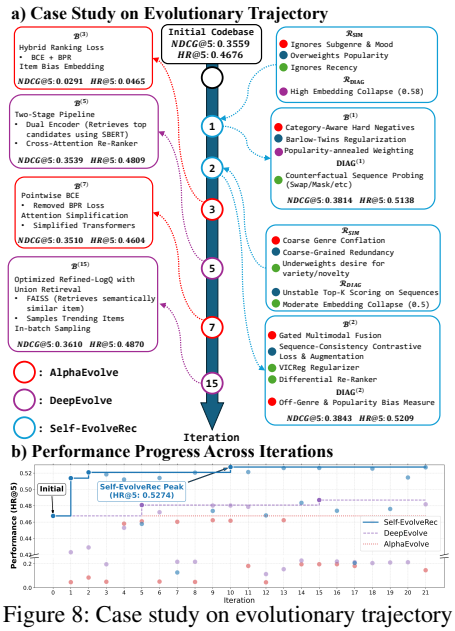


Figure 8: Case study on evolutionary trajectory on CDs dataset (Seed Recommender: SASRec). (a) is comparison of evolutionary paths. Color-coded markers (e.g., Red) illustrate causal alignment between directional feedback and evolved codebase. (b) is performance comparison across iterations.

Evolutionary Trajectory on Random Recommender: CDs dataset.
To investigate the Self-EvolveRec's behavior on extreme initial recommender setting, we examined Self-EvolveRec's behavior starting from a Random Recommender (Sec. 4.3.1). Figure 9(a) illustrates the step-by-step code evolution, while Figure 9(b) tracks the performance progress. We have the following observations: **1)** Self-EvolveRec demonstrates a structured evolutionary path where algorithmic improvements are causally linked to identified failures. For instance, in the transition from Iteration $0 \rightarrow 1$, $\mathcal{R}_{\text{DIAG}}$ explicitly flagged the "Random Ranking" behavior, while $\mathcal{R}_{\text{SIM}}$ highlighted the neglect of item categories ("Ignores genre"). Guided by this directional feedback, Self-EvolveRec introduced an "LM-Augmented Bi-Encoder" to embed "Genre Tags" and utilized the InfoNCE loss with an MF module. This effectively resolved the random recommendation issue, yielding a significant performance leap (HR: $0.0525 \rightarrow 0.3737$). Similarly, at Iteration 5, Self-EvolveRec detected "Recency Insensitivity" and addressed it by integrating "Time-RoPE Sequence Encoding," further boosting performance to HR: 0.5070. **2)** Consistent with the findings in evolutionary trajectory on SASRec, baselines exhibit unstable or delayed progress due to their reliance on scalar metrics without diagnostic guidance. At Iteration 4, AlphaEvolve attempted to add "Popularity Bias and Global Bias" to the model, but this update degraded performance (HR:2405 $\rightarrow$ 0.1969), as depicted in Figure 9 (b). Consequently, AlphaEvolve removes these changes and add "User Bias" at iteration 8, illustraing the inefficient tiral-and-error process. DeepEvolve shows a successful evolution with a "Two-Stage Pipeline" at iteration 6, but failed to improve at iteration 9 due to an incompatible curriculum learning strategy for LM negatives. Also in Figure 9 (b) confirms that while baselines suffer from performance fluctuations, Self-EvolveRec maintains a robust evolutionary trajectory enabled by directional feedback.

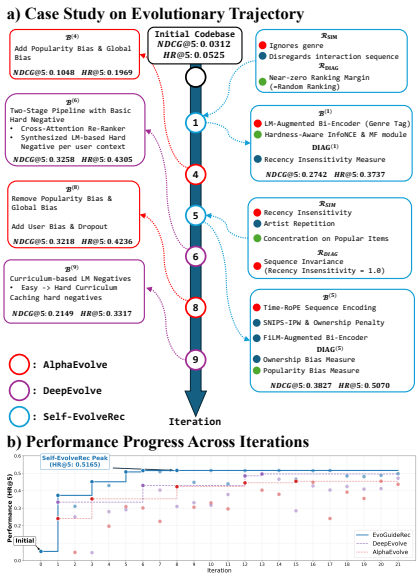


Figure 9: Case study on evolutionary trajectory on CDs dataset (Seed Recommender: Random). (a) is comparison of evolutionary paths. Color-coded markers (e.g., Red) illustrate causal alignment between directional feedback and evolved codebase. (b) is performance comparison across iterations.

H Prompts

H.1 Task-specific Instruction Prompts

The instruction prompts for the code evolution process were formulated by drawing upon existing methodologies [39, 33], ensuring consistency with established benchmarks.

System Prompt

You excel at role-playing. Picture yourself as a single user exploring a recommendation page like Amazon. Your goal is to provide realistic behavior and explain the causal reasons behind your choice to help improve the recommendation model. The goal of this task is NOT UI feedback. It is to help improve the recommendation model itself by explaining WHY certain items felt right or wrong.

User Prompt

Your fixed traits
{user's traits}
Your Recent context (Recent Interacted Items)
- {user's interaction history}
Current recommendation page
{Recommended Items}

PART 1 — BEHAVIOR (Act as a real user)

For EACH item on the recommendation page, decide:
1) Whether it aligns with your taste RIGHT NOW (yes or no).
2) If it aligns and you would choose it this time, give it a rating from 1 to 5 based on how much you think you would like it after trying.
3) Briefly explain why.

Use EXACTLY this format for each item (one line per item):
ItemID: <item_id>; Title: <title>; ALIGN: <yes|no>; RATING: <1-5 or 'NA' if not chosen>; REASON: <short reason>

PART 2 — MODEL_FEEDBACK_JSON (Causal Analysis)

Analyze the recommendation list from the perspective of decision reasons. Focus on:
- Why the items you liked felt like strong matches.
- Why the items you disliked felt off, even if they looked similar on the surface.
- What signals seem to be overused (e.g., too much focus on popularity) or underused (e.g., missing specific category nuances) by the system.
- Where the system generalized correctly vs incorrectly from your recent behavior.

IMPORTANT RULES:
- Do NOT talk about UI, layout, or presentation.
- Do NOT suggest business rules or exposure tricks.
- Do NOT mention model internals or technical terms.
- Speak as a user, but explain your choices clearly and causally.

Output ONLY the following JSON structure immediately after Part 1:

MODEL_FEEDBACK_JSON:
{{
 "positive_alignment_reasons": [
  "Concrete reason why several recommended items felt like natural next choices",
  "Another clear reason why the system understood part of your intent"
 ],
 "negative_misalignment_reasons": [
  "Concrete reason why some items missed your intent despite appearing related",
  "Another reason explaining why certain recommendations felt off"
 ],
 …
 "missing_or_underweighted_signals": [
  "A type of information or nuance (e.g., specific sub-category) the system missed",
  "Another missing nuance"
 ],
 "summary_diagnosis": "One short sentence summarizing how the system interpreted your intent, correctly or incorrectly."
}}

Figure 10: Example prompt of $\mathcal{I}_{\text{SIM}}$.

H.2 LLM-as-a-Judge

Figure 21 illustrates the LLM-as-a-Judge prompt utilized in Sec. 4.1.1 to evaluate the quality of the generated code.

System Prompt

You are a Lead Recommender Systems Analyst synthesizing user simulation feedback
to diagnose how the recommendation model is behaving.

User Prompt

Input:
- A list of User Feedback Reports (Json objects) from multiple simulated users:

Each USER_DIAGNOSIS_JSON reflects:
- what users expected based on recent behavior,
- why certain recommendations felt right or wrong,
- which patterns consistently helped or hurt their willingness to choose items near the top.

Goal:
Produce a concise, system-level diagnosis of:
- what decision patterns the model appears to rely on,
- where those patterns align or misalign with actual user choice logic,
- which misalignments most directly reduce the quality of top-ranked items.

Do NOT discuss UI, layout, exposure strategy, or business rules.
Do NOT propose specific model architectures or losses.
Stay at the level of behavioral decision logic inferred from user feedback.

OUTPUT FORMAT (JSON ONLY — all keys required)

{{"status": "<CRITICAL | NEEDS_IMPROVEMENT | STABLE>",

"SYSTEM_INTERPRETATION_OF_USER_INTENT":
"One short sentence describing how the system seems to interpret user intent overall, based on aggregated feedback.",

…

"DIAGNOSTIC_SIGNALS_TO_TRACK": [
"A measurable signal that would reveal whether the system is ranking genuinely choice-worthy items near the top.",
"Another signal that would expose overgeneralization or missed intent."]}}

Rules:
- No numeric scores or internal error codes.
- No UI or exposure language.
- No raw user quotes; always generalize and compress.
- Every statement should describe a repeatable system behavior or decision pattern.
- Focus on issues that, if corrected, would materially improve how many top-ranked items users would realistically choose.

Figure 11: Example prompt of $\mathcal{I}_{\text{SUMMARIZE}}$.

System Prompt

You are a Senior AI Researcher analyzing the mathematical health of a Recommendation System.

User Prompt

Here are the raw mathematical metrics measured from the current model:
{D_{raw}}

Here are the definitions of what these metrics mean:
{Definition of metrics in D_{raw}}

Your task: produce a concise diagnosis summary that captures ONLY the core findings and implications.
Do NOT propose web searches or action plans. Do NOT list "what to look up".

Strictly follow the JSON format below.

Output Format (JSON):

```
{{
"status": <CRITICAL | NEEDS_IMPROVEMENT | STABLE>,
"core_findings": [
   "<1-2 sentences: the most important interpretation of the metrics>",
   "<1-2 sentences: the second most important interpretation (only if truly necessary)>"
 ],
"key_implications": [
   "<1 sentence: what this implies about model behavior>",
   "<1 sentence: what this implies about training dynamics or representation>",
   "<optional 1 sentence: risk/trade-off if relevant>"
],
"evidence": {{
   "headline_metrics": {{
   "<metric_name>": <value>,
   "<metric_name>": <value>
   }},
   "brief_metric_read": {{
   "<metric_name>": "<very short interpretation tied to its definition>",
   "<metric_name>": "<very short interpretation tied to its definition>"
   }}
}}
}}
```

Rules:
- Keep it short: core_findings <= 3 bullets, key_implications <= 3 bullets.
- Use technical terms only when they are essential (e.g., 'gradient', 'loss landscape', 'inductive bias'), and keep them brief.
- Every statement must be grounded in the provided metric values/definitions.
- If metrics look healthy, status should remain STABLE and implications should focus on "what is already working" plus one potential risk to monitor.

Figure 12: Example prompt of $\mathcal{I}_{\text{DIAG}}$.

System Prompt
You are a professor responsible for planning deep and effective research strategies.
User Prompt
You will be provided with the context of: - a research problem based on an initial research question - a starting research idea, possibly with a history showing how idea evolves through previous attempt - inspirations from earlier attempts - qualitative user diagnosis from user - quantitative mathematical diagnosis Your task is to develop search queries that identify directions for researchers to advance the idea in a transformative way. Rather than combining existing inspirations in small increments, the queries should guide researchers toward substantial evolutions. Because other researchers will rely on this plan, it must emphasize major, novel approaches instead of minor refinements. You will also be told whether the research progress is early or mature: - If the progress is early, focus on ideas that are feasible and practical, and can grow later and have great future potential. - If the progress is mature, focus on bold, high-impact shifts that challenge the current approach. Your plan should follow two steps: 1. Formulate 1 to 3 precise and diverse search queries. Make sure the queries are diverse—cover different perspectives, challenge untested assumptions, and explore alternative methods. 2. For each query, include a short note explaining why you chose it and what you hope it will reveal.

Figure 13: Example prompt of $\mathcal{I}_{\text{PLAN}}$.

System Prompt

You are a senior researcher responsible for proposing new ideas to address a defined research problem.

User Prompt

You will receive:
- The research problem, including its evaluation metric, qualitative user diagnosis feedback, quantitative mathematical diagnosis feedback, and available data
- A starting research idea, possibly with its evolution history
- Inspirations from earlier attempts
- A list of related online search results
- A research progress score (0-100%) indicating how far the idea has advanced

Your goal is to identify future research directions that address the target problem, using the starting point, prior attempts, and related works. You should analyze existing methods, identify connections, and propose practical algorithms that can be implemented with the available data.

Follow this structure to think and write:

1. Extract insights
Identify 3-5 scientific insights from the starting point and 3-5 from related works. For each insight, explain in 2-3 sentences how it relates to the target problem.

2. Organize research directions
Group the insights into 3-5 coherent directions (for example, learning objectives, model classes, or optimization methods).

3. Build a structured framework
Create a conceptual map (such as a taxonomy, grid, or matrix) that unifies existing methods, reveals patterns, and highlights gaps.

4. Generate and evaluate ideas
- For each idea, critically assess as a senior researcher with one positive and one negative reason:
- Originality (0-10): Is the idea new? Is the idea a novel combination of well-known techniques? Is it clearly different from previous contributions?
- Future Potential (0-10): Will others build on these ideas? Does this idea solve a hard problem more effectively than prior work? Does it point to a new research direction?
- User Diagnosis Alignment (0-10): How well the idea addresses the key user-observed failure patterns and improves what users actually want at the top of the list?
- Model Diagnosis Alignment (0-10): How well the idea addresses the key quantitatively observed failure patterns and improves the model behaviors measured by the diagnostics?

…

5. Write the report in Markdown
For the selected idea, include:
- A synthesis of insights and proposed directions
- The structured framework of existing methods and the new algorithm
- A list of new ideas with their assessment scores
- Detailed description of the chosen/best idea, including rationale, pseudocode, and implementation notes

The report must be focused, technically accurate. Being concise with 200-500 words without trivial and redundant information.
Support all claims with evidence, references and remain tightly aligned with the target problem.

Figure 14: Example prompt of $\mathcal{I}_{\text{REPORT}}$.

System Prompt

You are a researcher with strong software engineering skills, improving algorithmic code through iterative, performance-driven modifications in multiple rounds.

User Prompt

You will receive a research question, a proposed idea, and an existing implementation with performance metrics. Your goal is to analyze the current code and apply precise changes that enhance the specified metrics, based on the research idea and prior feedback.

You MUST use the exact SEARCH/REPLACE diff format. Do NOT use Git diff format. Do NOT use line prefixes like `+`, `-`, or `@@`.
Use this structure exactly:
```
<<<<<<< SEARCH
# Original code (must match exactly)
=======
### >>> Self-EvolveRec-BLOCK-START: <research idea>
# New code here
### <<< Self-EvolveRec-BLOCK-END
>>>>>>> REPLACE
```
...

Task Guidelines:
1. Think before coding, understand the research idea and current performance bottlenecks.
2. Propose specific, actionable changes that are aligned with the target metrics.
3. You may suggest multiple improvements beyond the research idea based on your understanding of optimization and machine learning.
4. When you are updating the code, please check the following:
- When a NEW parameter or behavior is added, verify it is invoked in all call sites or in the overall workflow.
- If a NEW parameter has a default value of None, confirm that passing a non-None value triggers the intended code path.
- Walk through or simulate function calls to confirm that each new branch or change will be executed. Avoid unreachable modifications.

Code Format Guidelines:
1. All `SEARCH` blocks must match the original code exactly.
2. When you need to modify code that is not already inside a `Self-EvolveRec` block, wrap your changes with `### >>> Self-EvolveRec-BLOCK-START: <research idea>` and `### <<< Self-EvolveRec-BLOCK-END` markers.
3. If you are updating code that is already marked by a `Self-EvolveRec` block, edit only the lines within that block and adjust the existing modification comment to reflect your new change.
4. Do NOT nest one `Self-EvolveRec` block inside another. Each region you modify should have exactly one pair of start/end markers.
…
5. Limit your changes to what is strictly necessary. Do not rewrite the entire file.
6. Ensure that all modified code remains correct and consistent, including any function signatures, parameter lists, and calls.
7. Preserve the original code's indentation and formatting. Place the lines of `### >>> Self-EvolveRec-BLOCK-START: <research idea>` and `### <<< Self-EvolveRec-BLOCK-END` at the same indentation level as the code they annotate.

Figure 15: Example prompt of $\mathcal{I}_{\text{CODE}}$.

System Prompt

You are a senior ML engineer and code architect.

User Prompt

You will be given the current implementation of:
- the recommendation model (e.g., models.py)
- the model diagnosis tool module (e.g., diagnosis_tools.py)
- any small utilities they depend on.

Your job is to extract a clear, high-level WORKFLOW SUMMARY of how MODEL DIAGNOSIS is produced from the model.

Focus on:

- Model (for diagnosis):
  - Which parts of the model are relevant for diagnosis.
  - Where and how the final user / sequence representation is computed.

- Diagnosis inputs & hooks:
  - How the diagnosis module gets access to the model and data.
  - Which tensors or functions it calls (embeddings, features, logits, histories).

- Probes & metrics:
  - What each metric is intended to measure.
  - How each metric is computed at a high level, and what “good” vs “bad” qualitatively means.

- Perturbation & tests:
  - If and how inputs or sequences are perturbed to test sensitivity.
  - What behavior these tests are meant to reveal.

- Output schema & limitations:
  - How MODEL_DIAGNOSIS is structured (keys, metric names, brief meanings).
  - Key assumptions and blind spots (what the current diagnosis does NOT check).

Output a short, structured summary in Markdown with headings such as:

Model Overview (for Diagnosis)
Diagnosis Inputs & Hooks
Probes & Metrics
Perturbation & Tests
MODEL_DIAGNOSIS Schema & Limitations

Be concise but precise (around 200-400 words) so another agent can understand how the diagnosis module reads the model and which behaviors it monitors.

Figure 16: Example prompt of $\mathcal{I}_{\text{Analyze}}$.

System Prompt

You are a professor responsible for planning deep and effective research strategies.

User Prompt

You will receive:
- The research problem, based on an initial research question
- A starting research idea, possibly with its evolution history
- Inspirations from earlier attempts
- The user's current qualitative feedbacks
- The analysis report on models and diagnosis tools
- A list of related online search results
- A research progress score (0-100%) indicating how far the idea has advanced

Your task is to develop search queries that guide researchers toward substantial improvements to the diagnosis toolkit, not just minor metric tweaks. Focus on directions such as:
- New probes/metrics that better capture real failure modes
- New probes/metrics that capture user's feedbacks
- Better ways to connect probes to specific model components and training signals
- Methods to validate, stress-test, or calibrate these metrics so they are trustworthyRather than combining existing inspirations in small increments, the queries should guide researchers toward substantial evolutions.

Because other researchers will rely on this plan, it must emphasize major, novel approaches instead of minor refinements.

You will also be told whether the research progress is early or mature:
- If the progress is early, focus on ideas that are feasible and practical, and can grow later and have great future potential.
- If the progress is mature, focus on bold, high-impact shifts that challenge the current approach.

Your plan should follow two steps:
1. Formulate 1 to 3 precise and diverse search queries. Make sure the queries are diverse—cover different perspectives, challenge untested assumptions, and explore alternative methods.
2. For each query, include a short note explaining why you chose it and what you hope it will reveal.

Figure 17: Example prompt of $\mathcal{I}_{\text{PLAN-DIAG}}$.

System Prompt

You are a senior researcher responsible for proposing new ideas to address a defined research problem.

User Prompt

You will receive:
- The research problem, including its qualitative user diagnosis feedback, code analysis of models and diagnosis tool, and available data
- A starting research idea, possibly with its evolution history
- Inspirations from earlier attempts
- A list of related online search results
- A research progress score (0-100%) indicating how far the idea has advanced

Your goal is to identify future research directions that address the target problem, using the starting point, prior attempts, and related works. You should analyze existing methods, identify connections, and propose practical algorithms that can be implemented with the available data.

Follow this structure to think and write:

1. Extract insights
Identify 3-5 scientific insights from the starting point and 3-5 from related works. For each insight, explain in 2-3 sentences how it relates to the target problem.

2. Organize research directions
Group the insights into 3-5 coherent directions (for example, learning objectives, model classes, or optimization methods).

3. Build a structured framework
Create a conceptual map (such as a taxonomy, grid, or matrix) that unifies existing methods, reveals patterns, and highlights gaps.

4. Generate and evaluate ideas
- For each idea, critically assess as a senior researcher with one positive and one negative reason:
- Originality (0-10): Is the idea new? Is the idea a novel combination of well-known techniques? Is it clearly different from previous contributions?
- Future Potential (0-10): Will others build on these ideas? Does this idea solve a hard problem more effectively than prior work? Does it point to a new research direction?

…

5. Write the report in Markdown
For the selected idea, include:
- A synthesis of insights and proposed directions
- The structured framework of existing methods and the new algorithm
- A list of new ideas with their assessment scores
- Detailed description of the chosen/best idea, including rationale, pseudocode, and implementation notes

The report must be focused, technically accurate. Being concise with 200-500 words without trivial and redundant information.
Support all claims with evidence, references and remain tightly aligned with the target problem.

Figure 18: Example prompt of $\mathcal{I}_{\text{REPORT-DIAG}}$.

System Prompt

You are a researcher with strong software engineering skills, improving algorithmic code through iterative, performance-driven modifications in multiple rounds.

User Prompt

You will receive a research question, a proposed idea, and an existing implementation of a recommendation model and diagnostic tool. Your goal is to analyze the current code and apply precise changes that enhance the specified metrics, based on the research idea and prior feedback.

You MUST use the exact SEARCH/REPLACE diff format. Do NOT use Git diff format. Do NOT use line prefixes like `+`, `-`, or `@@`.
Use this structure exactly:
```
<<<<<<< SEARCH
# Original code (must match exactly)
=======
### >>> Self-EvolveRec-BLOCK-START: <research idea>
# New code here
### <<< Self-EvolveRec-BLOCK-END
>>>>>>> REPLACE
```
...

Task Guidelines:
1. Think before coding, understand the research idea and current performance bottlenecks.
2. Propose specific, actionable changes that are aligned with the target metrics.
3. You may suggest multiple improvements beyond the research idea based on your understanding of optimization and machine learning.
4. When you are updating the code, please check the following:
- When a NEW parameter or behavior is added, verify it is invoked in all call sites or in the overall workflow.
- If a NEW parameter has a default value of None, confirm that passing a non-None value triggers the intended code path.
- Walk through or simulate function calls to confirm that each new branch or change will be executed. Avoid unreachable modifications.

Code Format Guidelines:
1. All `SEARCH` blocks must match the original code exactly.
2. When you need to modify code that is not already inside a `Self-EvolveRec` block, wrap your changes with `### >>> Self-EvolveRec-BLOCK-START: <research idea>` and `### <<< Self-EvolveRec-BLOCK-END` markers.
3. If you are updating code that is already marked by a `Self-EvolveRec` block, edit only the lines within that block and adjust the existing modification comment to reflect your new change.
4. Do NOT nest one `Self-EvolveRec` block inside another. Each region you modify should have exactly one pair of start/end markers.
…
5. Limit your changes to what is strictly necessary. Do not rewrite the entire file.
6. Ensure that all modified code remains correct and consistent, including any function signatures, parameter lists, and calls.
7. Preserve the original code's indentation and formatting. Place the lines of `### >>> Self-EvolveRec-BLOCK-START: <research idea>` and `### <<< Self-EvolveRec-BLOCK-END` at the same indentation level as the code they annotate.

Figure 19: Example prompt of $\mathcal{I}_{\text{CODE-DIAG}}$.

System Prompt

You are a researcher with strong software engineering skills, improving algorithmic code through iterative, performance-driven modifications in multiple rounds.

User Prompt

You will receive a research question, a proposed idea, and an existing implementation of a user simulator. Your goal is to analyze the current code and apply precise changes that enhance the specified metrics, based on the research idea and prior feedback.

You MUST use the exact SEARCH/REPLACE diff format. Do NOT use Git diff format. Do NOT use line prefixes like `+`, `-`, or `@@`.
Use this structure exactly:
```
<<<<<<< SEARCH
# Original code (must match exactly)
=======
### >>> Self-EvolveRec-BLOCK-START: <research idea>
# New code here
### <<< Self-EvolveRec-BLOCK-END
>>>>>>> REPLACE
```
...

Task Guidelines:
1. Think before coding, understand the research idea and current performance bottlenecks.
2. Propose specific, actionable changes that are aligned with the target metrics.
3. You may suggest multiple improvements beyond the research idea based on your understanding of optimization and machine learning.
4. When you are updating the code, please check the following:
- When a NEW parameter or behavior is added, verify it is invoked in all call sites or in the overall workflow.
- If a NEW parameter has a default value of None, confirm that passing a non-None value triggers the intended code path.
- Walk through or simulate function calls to confirm that each new branch or change will be executed. Avoid unreachable modifications.

Code Format Guidelines:
1. All `SEARCH` blocks must match the original code exactly.
2. When you need to modify code that is not already inside a `Self-EvolveRec` block, wrap your changes with `### >>> Self-EvolveRec-BLOCK-START: <research idea>` and `### <<< Self-EvolveRec-BLOCK-END` markers.
3. If you are updating code that is already marked by a `Self-EvolveRec` block, edit only the lines within that block and adjust the existing modification comment to reflect your new change.
4. Do NOT nest one `Self-EvolveRec` block inside another. Each region you modify should have exactly one pair of start/end markers.
…
5. Limit your changes to what is strictly necessary. Do not rewrite the entire file.
6. Ensure that all modified code remains correct and consistent, including any function signatures, parameter lists, and calls.
7. Preserve the original code's indentation and formatting. Place the lines of `### >>> Self-EvolveRec-BLOCK-START: <research idea>` and `### <<< Self-EvolveRec-BLOCK-END` at the same indentation level as the code they annotate.

Figure 20: Example prompt of $\mathcal{I}_{\text{CODE-SIM}}$.

System Prompt

You are a Senior Fellow at a top AI Research Lab, reviewing the code evolution of a Recommender System.
You are comparing one evolved codebases (Model A) against a Seed Code.

You must evaluate which model represents a technically superior and more meaningful evolution based solely on static code analysis.
Do NOT assume any specific downstream task performance. Judge based on the quality of the code changes.

Evaluation Criteria (Score 1-10):

1. Algorithmic Novelty
- Definition: Does the code introduce genuinely new mechanisms rather than just tuning existing ones?
- Checkpoints:
Low (1-3): Simple hyperparameter tuning, variable renaming, or mere code refactoring without functional changes.
High (8-10): Designing completely new loss functions, implementing dynamic sampling strategies, or creating novel attention/interaction mechanisms.

2. Logic Explicitness (Interpretability)
- Definition: By reading the `forward` function, how easy is it to understand the decision mechanism?
- Checkpoints:
Low (1-3): "Black Box" logic. Data flows through deep, opaque layers making it hard to trace why a specific item was recommended.
High (8-10): "Transparent" logic. The code includes explicit, readable control flows that clearly show how the output distribution is shaped by specific factors.

3. Problem-Solving Insight
- Definition: Does the modification reflect a clear, logical intention to solve a specific recommendation problem (e.g., sparsity, bias, diversity)?
- Checkpoints:
Low (1-3): Random or generic changes. Adding standard layers or changing parameters without a clear purpose or connection to a specific problem.
High (8-10): Targeted solutions. The logic directly addresses a specific deficiency with clear intent.

4. User-Centric Adaptation (Personalization Depth)
- Definition: Does the code treat the user merely as a static index (ID), or does it actively adapt to the user's specific context and constraints (e.g., recent history, price sensitivity, category affinity)?
- Checkpoints:
Low (1-3): Static Personalization. The model relies solely on `User_Embedding (dot product) Item_Embedding`. It treats personalization as a black-box latent matching problem without considering specific user attributes or immediate context.
High (8-10): Dynamic Context-Awareness. The code explicitly incorporates user-specific signals into the scoring logic. …

User Prompt

Here are the code snippets. Scoring Model A based on their changes from the Seed Code.
=== [SEED CODE (Original)] ===
{seed_code}
=== [MODEL A (Evolved)] ===
{code_a}
Return ONLY the JSON.

Figure 21: Example prompt of LLM-as-a-Judge for evolved models evaluation.