EVOSKILLREC: SKILL-GENOME EVOLUTION FOR RECOMMENDER ARCHITECTURE DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern recommender systems advance not only by scaling data and parameters, but also by encoding task-specific inductive biases through architecture, including sparse feature interactions for click-through rate (CTR) prediction, temporal attention for sequential recommendation, and expert routing for multi-task learning. However, these biases are typically designed by human experts or searched within predefined operator spaces. Although recent LLM-driven code evolution expands this space, unconstrained edits often produce invalid or ineffective architectures, underuse established architecture design knowledge, and fail to preserve successful innovations for reuse. We introduce EVOSKILLREC, a promotion-and-reuse framework for cumulative recommender architecture evolution. It first decomposes recommenders into atomic executable skills and represents architectures as typed skill genomes, with each skill equipped with input–output types, semantic annotations, and implementation code. We then evolve models across different tasks through two coupled spaces: a constrained skill–space that mutates, recombines, specializes, and reuses validated skills, and an open-ended code–space in which LLM planners and synthesizers invent new skill modules using prior evolution traces and accumulated experience. An evolution controller evaluates candidates, diagnoses failures, retrieves relevant skills, promotes validated innovations into the skill library, and adaptively allocates the proposal budget between the two spaces. Extensive experiments on CTR prediction, multi-task learning, and multi-domain learning, including resource-constrained co-optimization of predictive quality and model FLOPs utilization in generative recommendation, consistently demonstrate the effectiveness of our proposed EVOSKILLREC. Code is available at `https://anonymous.4open.science/r/EvoSkillRec-BF55`.

1 INTRODUCTION

Recommender systems are large-scale prediction engines that estimate user preferences from historical interactions and side information. Their backbones have evolved substantially with the platforms they serve, moving from content-based filtering to deep ranking models and, more recently, to generative ranking models. Across every stage, however, many consequential advances have stemmed from a narrower, more identifiable source: ***architectural inductive bias***. In click-through-rate (CTR) prediction, the field converged on the view that sparse categorical fields should interact in structured, often high-order ways, motivating explicit factorization and feature-crossing modules such as DeepFM (Guo et al., 2017) and DCN (Wang et al., 2017). In target-aware ranking, a consensus emerged that different candidate items should activate different parts of a user's history, realized through attention-based interest extraction in DIN (Zhou et al., 2018) and later extended by DIEN (Zhou et al., 2019). In multi-task recommendation, the shared insight is that tasks should share representations without collapsing task-specific signal, as instantiated in MMoE (Ma et al., 2018a) and PLE (Tang et al., 2020). These examples support a unifying view: a recommender is not merely a stack of neural layers, but a composition of reusable inductive-bias units.

This view exposes a bottleneck. The first generation of strong recommender architectures, such as Wide & Deep (Cheng et al., 2016), DeepFM (Guo et al., 2017), DCN (Wang et al., 2017), and SASRec (Kang & McAuley, 2018), demonstrates the value of expert-designed inductive bias, but the design cycle remains slow and largely manual. Adapting to a new task, feature distribution, or

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

business scenario typically requires repeated human trial and error, and knowledge embedded in one model family is rarely represented in a form directly reusable by another task.

Recommender AutoML and neural architecture search (NAS) automate part of this burden by searching over operators, connectivity patterns, or loss functions rather than designing them manually. Automated neural interaction search discovers feature-interaction architectures for CTR prediction (Song et al., 2020); NASRec trains weight-sharing supernets over recommender architectures (Zhang et al., 2022); AutoLossGen searches loss-function programs for recommendation objectives (Li et al., 2022b); and recent work extends automated model design to joint model–hardware optimization (Zhang et al., 2024a). These approaches are valuable but share a structural limitation: the search space is still specified by humans. A NAS system can efficiently select among candidate operators but rarely invents a new module, objective, or routing mechanism outside its given grammar.

LLM-driven program evolution offers a possible escape from this fixed-space restriction. Program-search systems such as FunSearch (Romera-Paredes et al., 2024) and AlphaEvolve (Novikov et al., 2025) use language models to propose code in a broad, largely unconstrained program space, with evaluators selecting useful variants. Concurrent work has begun applying this idea to recommender systems: Self-EvolveRec uses LLM-based directional feedback to iteratively edit recommender code (Kim et al., 2026), and a related system for large-scale video recommendation coordinates offline and online LLM agents for autonomous optimization (Wang et al., 2026b). These works support an important finding: model design can become an iterative, evaluator-driven research process, with LLMs as the proposal source.

Direct LLM code evolution, however, is not yet sufficient for recommender architecture discovery, for two reasons. First, open-ended code edits, unconstrained by structural priors, tend to produce many invalid or low-value modifications when search ignores accumulated inductive-bias knowledge in the field. Second, even when an edit succeeds, its useful idea remains trapped in a one-off code diff: the next generation starts again from a codebase and feedback transcript rather than accumulated, validated recommender knowledge. A self-evolving recommender should not merely mutate; it should remember.

We propose **EVOSKILLREC**, a framework for skill-genome evolution in recommender architecture design and discovery. First, we treat ***recommender modules as skills***: EVOSKILLREC decomposes existing recommender models into atomic, executable modular skills, annotates each with semantic and inductive-bias knowledge, validates its functional correctness and reusability, and stores it in a library with its provenance, empirical performance history, and implementation. Second, we design a ***dual-space evolution process that jointly supports knowledge reuse and open-ended discovery*** for architecture search. In the skill space, EVOSKILLREC performs reliable genetic operations over known skills (add, replace, hybridize, and specialize) to exploit accumulated architectural knowledge. In the code space, EVOSKILLREC permits open-ended LLM invention when the existing skill library cannot readily address a diagnosed failure, producing a new module, fusion design, auxiliary objective, or routing mechanism grounded in the evolutionary trace. Successful inventions are not discarded after a single run: they are compiled back into skill cards and become part of the evolving genome for reuse in subsequent generations. Our contributions are as follows.

- We formulate recommender self-evolution as *skill-genome accumulation*: a model-to-skill compiler decomposes recommender code into validated, reusable architectural units curated into a growing library rather than discarded after use.
- We design a dual-space evolution algorithm that couples type-checked mutation over the skill library with open-ended code-space invention, closed by a cumulative evolution loop that distills successful inventions into the library.
- We show that EVOSKILLREC outperforms strong hand-designed and automated-design baselines across recommendation benchmarks, with skill-space and code-space evolution providing complementary, task-dependent gains.

2 RELATED WORK

AutoML and NAS for recommendation. Automated recommender design applies AutoML and NAS (Zoph & Le, 2017; Real et al., 2019) to architectures, feature interactions, embedding config-

urations, and training objectives. Examples include evolutionary architecture search in AutoCTR (Song et al., 2020), weight-sharing supernet search in NASRec (Zhang et al., 2022), loss-function synthesis from mathematical operators in AutoLossGen (Li et al., 2022b), and model–hardware co-design (Zhang et al., 2024a). EVOSKILLREC evolves both architectures and their *reusable building blocks*: it combines skill-space composition with code-space synthesis and incorporates validated modules into a growing skill library, allowing the set of available modules to grow during search.

LLM-driven program evolution. LLM-driven program evolution combines code generation with external evaluation: ELM uses LLMs as program mutation operators (Lehman et al., 2023), FunSearch discovers mathematical constructions and algorithmic heuristics (Romera-Paredes et al., 2024), and AlphaEvolve extends this paradigm to broader algorithmic and systems optimization (Novikov et al., 2025). In recommendation, Self-EvolveRec guides code evolution with directional feedback from user simulation and model diagnosis (Kim et al., 2026), while Wang et al. (2026b) coordinate offline and online agents for autonomous model optimization in large-scale video recommendation. These approaches retain programs, feedback, or experimental knowledge to guide subsequent iterations. EVOSKILLREC focuses on a complementary mechanism: *extracting, validating, and reusing* architectural modules as composable skills, so that successful code modifications enrich the building blocks available for subsequent architecture search.

3 PROBLEM SETTING

Let a recommender training instance be $(\mathcal{D}, \mathcal{T}, \mathcal{C})$, where $\mathcal{D}$ is the dataset, $\mathcal{T}$ specifies the task and its metrics (e.g., CTR prediction), and $\mathcal{C}$ specifies research constraints, such as latency and parameter budgets. A model $\mathcal{M}$ comprises code, parameters, data-processing assumptions, and training logic. Conventional AutoML and NAS solve

$$\mathcal{M}^\star = \arg\max_{\mathcal{M}\in\Omega} \ \mathcal{E}(\mathcal{M}; \mathcal{D}, \mathcal{T}, \mathcal{C}), \tag{1}$$

where human experts fix the search space Ω in advance. EVOSKILLREC instead maintains an evolving skill library $\mathcal{L}_t$ at generation t and searches jointly over compositions of existing skills and newly synthesized code:

$$\mathcal{M}_{t+1} \in \mathrm{Compose}(\mathcal{L}_t) \ \cup \ \mathrm{Invent}(\mathcal{L}_t, r_t), \tag{2}$$

where r_t diagnoses the current failure mode. When an LLM-coded candidate passes validation and survives selection, its validated, portable components are added to $\mathcal{L}_{t+1}$. Unlike Eq. 1, the objective of EVOSKILLREC is thus twofold: find a strong $\mathcal{M}^\star$ for the current task, and improve the search distribution for future tasks by accumulating validated architectural knowledge. As validated inventions are added to the skill library, future searches can reuse a richer set of model components.

The evaluation $\mathcal{E}$ can also consider multiple criteria. For example, Section 5.2 considers both predictive quality and hardware efficiency, with explicit constraints checked during candidate selection.

4 EVOSKILLREC: SKILL-GENOME EVOLUTION

We now describe EVOSKILLREC in detail, and Figure 1 gives the overall picture. Section 4.1 introduces the skill card abstraction together with the model-to-skill compiler and two-tier library that populate and grow the skill genome. Section 4.2 presents the dual-space evolution process that couples constrained skill-space recombination with open-ended code-space invention, and the adaptive budget that arbitrates between them. Section 4.3 closes the loop with the evolution controller that diagnoses failures, drives both spaces, and consolidates validated discoveries into evolutionary memory.

4.1 RECOMMENDER MODULES AS SKILLS

The *Agent Skills* framework packages procedural knowledge, instructions, and code into reusable, composable resources that agents can discover and load on demand (Zhang et al., 2025). EVOSKILLREC brings the same principle to recommender architecture design: it re-represents each established architecture as a set of atomic, executable *skills*, turning the inductive-bias knowledge otherwise buried in a monolithic, task-specific implementation into explicit, validated, reusable building blocks that an evolutionary search can retrieve, recombine, and extend.

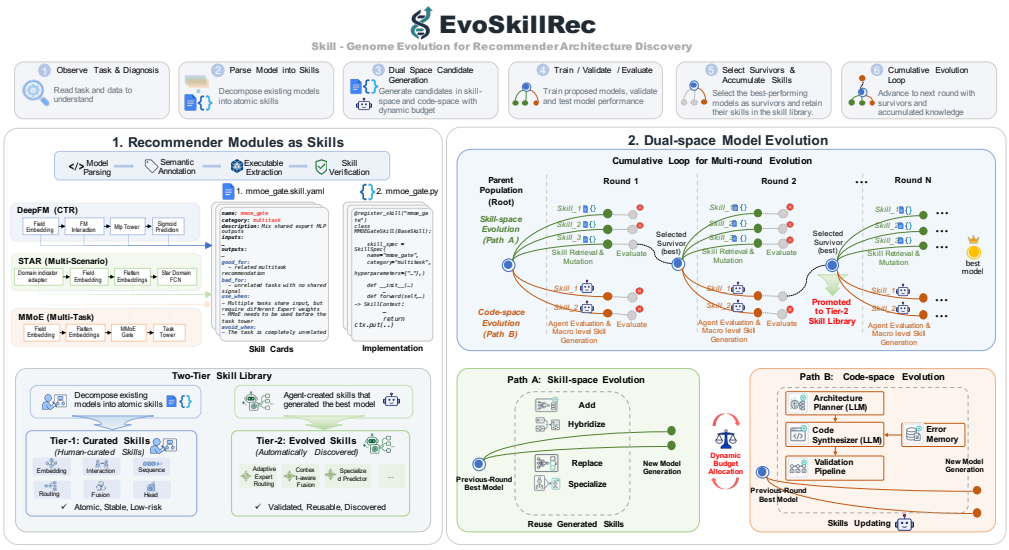

Figure 1: Overview of EVOSKILLREC. **(Left)** A model-to-skill compiler decomposes established recommender architectures into atomic, executable skills, documented by a skill card and admitted to the curated Tier-1 library only after validation. **(Right)** A model is a typed skill genome: under an adaptive budget, dual-space evolution proposes candidates by recombining library skills (skill space) or by LLM planning and synthesis of new modules (code space), while the cumulative evolution loop diagnoses candidates, selects survivors, and promotes validated inventions into Tier-2 for reuse in later generations.

4.1.1 THE SKILL CARD ABSTRACTION

We represent each skill as

$$\mathcal{S} = (c, I, O, b, a, f, v, m), \tag{3}$$

where c is the implementation code; I and O specify its input and output interfaces; b describes its inductive bias; a gives its applicable tasks and usage conditions; f records relevant failure modes; v is its validation suite; and m stores empirical results and reuse history.

Each skill has a **skill card** that describes what the module does, when to use it, and which tensors it consumes and produces. These descriptions support retrieval, while the declared tensor names, shapes, and data types support compatibility checks when composing a model.

The card is paired with a PyTorch implementation derived from `BaseSkill`, which performs the actual computation. It reads the required inputs from a shared tensor context $\mathcal{X}$ and writes its outputs back under the declared keys. For example, the `fm_interaction` skill from DeepFM (Guo et al., 2017) reads `field_embeddings`, computes second-order feature interactions, and writes `fm_output`.

4.1.2 MODEL-TO-SKILL COMPILER

Using this skill representation, EVOSKILLREC builds its initial library from established open-source recommender models. These models provide working implementations of task-specific inductive biases, such as feature interactions, sequence modeling, and expert routing. Extracting their components makes this architectural knowledge available for retrieval and recombination during evolution.

The model-to-skill compiler takes architectures such as DeepFM (Guo et al., 2017), Wide & Deep (Cheng et al., 2016), DCN (Wang et al., 2017), DIN (Zhou et al., 2018), DIEN (Zhou et al., 2019), SASRec (Kang & McAuley, 2018), MMoE (Ma et al., 2018a), and PLE (Tang et al., 2020) as input.[1] It converts their components into skills through four stages. During *parsing*, static analysis and LLM assistance identify modules whose inputs, outputs, parameters, and call sites can be

[1]We use open-source implementations from Torch-RecHub (Datawhale, 2022) and UniRank (Li et al., 2026).

isolated. *Semantic annotation* uses an LLM to describe each module in a skill card, and *executable extraction* packages it as an independently testable `BaseSkill` module. Finally, *validation* tests each module and reconstructs the source model from its extracted skills to verify that its end-to-end behavior matches the original implementation. Only validated skills enter the initial library.

4.1.3 TWO-TIER SKILL LIBRARY

The skills extracted by the compiler form **Tier-1**, the initial curated part of the library $\mathcal{L}$. This tier remains fixed within each search run and provides reusable building blocks throughout evolution.

Tier-2 grows during evolution as the LLM generates new modules in code space. A generated module is promoted only after it passes validation and portability checks and its host candidate survives selection. Its skill card records the evaluation results and provenance that support promotion.

Both tiers share the same skill-card abstraction and named-tensor interface. Newly promoted skills can therefore be retrieved and composed alongside curated skills under the same compatibility checks, making discoveries from one generation available for reuse in later generations. Appendix A provides the full card specification, compiler checks, library inventories, and examples from both tiers.

4.2 DUAL-SPACE EVOLUTION

The goal of EVOSKILLREC is to discover new recommender architectures that improve task performance under the given constraints. Reusing validated skills lets this search build on established inductive biases and tested implementations. The current library may nevertheless lack modules needed to address a model's weaknesses, motivating the generation of new computational components.

We therefore couple two search spaces. *Skill space* explores new architectures by retrieving and recombining compatible skills from $\mathcal{L}$. *Code space* uses an LLM to synthesize new modules for diagnosed modeling needs. Once promoted to Tier-2, these modules become building blocks for further skill-space exploration, linking module invention to architecture discovery. The controller distributes a fixed number of candidates between the two spaces at each generation, allocating more to code space when validation progress stalls.

4.2.1 SKILL-SPACE EVOLUTION

We represent a model as a directed acyclic genome graph $G = (V, E)$, whose nodes are skill instances and edges connect named tensors. Given G and its diagnosis r_t, skill-space evolution retrieves relevant skills from $\mathcal{L}$ to address the diagnosed weaknesses, then generates candidate architectures through graph edits inspired by genetic mutation and recombination.

Retrieval first filters both library tiers by task compatibility and input availability: required tensors must be available in G, with keys $\mathcal{X}_G$, or supplied by one upstream producer. Eligible skills are ranked by

$$\text{score}(\mathcal{S}_i \mid G, r_t) = w_1\phi_{\text{task}}(\mathcal{S}_i, \mathcal{T}) + w_2\phi_{\text{io}}(\mathcal{S}_i, \mathcal{X}_G) + w_3\phi_{\text{ret}}(\mathcal{S}_i, r_t) + w_4\phi_{\text{bias}}(\mathcal{S}_i, r_t) + w_5\bar{m}(\mathcal{S}_i), \tag{4}$$

The five terms respectively measure task compatibility, input availability, textual similarity between the skill card and diagnosis r_t, whether the inductive bias addresses the diagnosed failure, and historical average validated improvement. We retain the top-n skills under a per-category cap.

Evolution then modifies G using four operators: **add** connects a retrieved skill to a compatible fusion layer; **replace** substitutes a skill with compatible interfaces while preserving its output key and downstream connections; **hybridize** joins branches at compatible fusion points; and **specialize** tunes hyperparameters without changing skill identity or wiring. Static checks reject edits violating tensor compatibility or acyclicity before training.

4.2.2 CODE-SPACE EVOLUTION

Skill-space evolution recombines available skills, so its architectural choices are constrained by the current library. Addressing a diagnosed weakness may require a module that the library does not

yet provide. Code-space evolution therefore uses an LLM to create new components and expand the search space. Because code generation and candidate evaluation are costly, useful discoveries are retained as reusable skills for later searches. To guide generation, the LLM receives a structured *design brief*

$$\mathcal{D}_t = \big(G_t,\ r_t,\ \mathcal{M}_{t-\Delta:t},\ \mathcal{X}_{G_t},\ \mathcal{C}\big), \tag{5}$$

combining the parent genome G_t, diagnosis r_t, recent search history $\mathcal{M}_{t-\Delta:t}$, available tensor keys $\mathcal{X}_{G_t}$, and constraints $\mathcal{C}$.

An architecture planner uses this brief to propose new mechanisms and their connections to G_t. Sketches with unavailable inputs, invalid wiring, or previously tried structures are discarded before a code synthesizer implements the remaining sketches as `nn.Module` components. These must match the planned structure and declared interfaces and pass safety, forward/backward, and portability checks before training.

Validated modules whose host candidates survive selection enter Tier-2 together with their skill cards. Subsequent generations can retrieve and compose them in skill space, turning code-space discoveries into building blocks for further architecture search.

4.2.3 Adaptive Budget Allocation

Skill space emphasizes *exploitation* by recombining existing skills, while code space emphasizes *exploration* by generating new modules. To balance reuse and invention in architecture discovery, we adapt the candidate quotas of the two spaces according to validation progress.

Each generation has a fixed budget of N candidates. For CTR, MTL, and MDL, we set $N = 8$ and initialize $(n_{\text{skill}}, n_{\text{code}}) = (6, 2)$. When the best validation score plateaus, the controller shifts slots from skill space to code space, subject to a maximum code quota and a minimum skill quota. A qualifying validation improvement restores the initial split. If code generation yields too few valid proposals, skill-space candidates fill the remaining slots.

4.3 Cumulative Evolution Loop

The cumulative evolution loop integrates the two search spaces and adaptive budget allocation into a self-evolving search process, using evaluation feedback to guide successive generations. Each generation diagnoses parents using memory $\mathcal{M}$ and proposes candidates in both spaces. Validated, unique candidates are trained and evaluated under a shared task protocol. Survivors selected by $\mathcal{E}$ become the next parents, and their validated code-space modules enter Tier-2 of $\mathcal{L}$. Memory $\mathcal{M}$ records successes and failures to guide subsequent generations. Appendix B details retrieval, validation, and skill promotion, with the full procedure in Algorithm 1.

5 Experiments

We evaluate EvoSkillRec through five research questions (RQs) concerning predictive performance, architecture evolution, the roles of the two search spaces, accuracy–efficiency trade-offs, and the effect of the LLM backbone.

5.1 Setup and Baselines

Tasks and datasets. We consider click-through rate (CTR) prediction on MovieLens-1M, Amazon Beauty, and Amazon Books; multi-task learning (MTL) on Census-Income; and multi-domain learning (MDL) on Amazon and MovieLens-1M. We additionally evaluate joint optimization of predictive quality and training efficiency on the industrial-scale QK-Video corpus.

Baselines. We compare against three categories of baselines: (i) manually designed architectures, comprising 7 CTR, 5 MTL, and 11 MDL models; (ii) NASRec (Zhang et al., 2022), which performs architecture search using a weight-sharing supernet; and (iii) OpenEvolve, an open-source LLM-driven code-evolution framework inspired by AlphaEvolve (Novikov et al., 2025). Task definitions, dataset statistics, preprocessing and splits, and the baseline catalog are provided in Appendix C.

Table 1: **Test AUC across CTR, MTL, and MDL benchmarks.** *Best Manual* is the strongest manually designed model on each dataset. Δ is the gain of EvoSkillRec over the strongest non-EvoSkillRec baseline in that column.

Method	CTR			MTL	MDL	
	MovieLens	Amazon Books	Amazon Beauty	Census	Amazon	MovieLens
Best Manual	0.7487[a]	0.6587[b]	0.6666[c]	0.9726[d]	0.7012[e]	0.7924[e]
NASRec	0.7361	0.6487	0.6632	0.9711	0.6870	0.7619
OpenEvolve	0.7521	0.6654	0.6760	0.9722	0.6903	0.7730
EvoSkillRec	**0.7522**	**0.6708**	**0.6845**	**0.9748**	**0.7033**	**0.7932**
Δ vs. best baseline	↑0.0001 (+0.01%)	↑0.0054 (+0.81%)	↑0.0085 (+1.26%)	↑0.0022 (+0.23%)	↑0.0021 (+0.30%)	↑0.0008 (+0.10%)

Best manual models by reported test AUC (full results in Appendix E.1): [a,b]Wide & Deep, [c]AutoInt, [d]AITM, [e]SAR-Net.

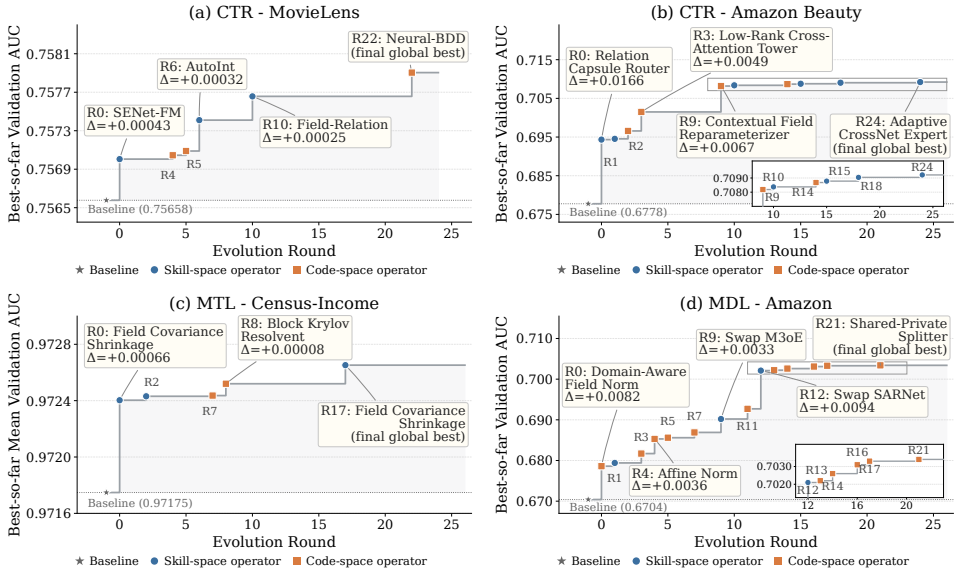

Figure 2: Best-so-far validation AUC over evolution rounds on four representative benchmarks. Each annotated improvement reports its operator and AUC gain, with color indicating the space that proposed the candidate.

Metrics and protocol. We report held-out test AUC, using the mean AUC across tasks for MTL and overall AUC across domains for MDL. On QK-Video, we additionally report model FLOPs utilization (MFU). Candidate and checkpoint selection uses validation metrics; held-out test metrics are used for reporting, not selection. QK-Video selection uses validation AUC and training MFU jointly. Training protocols, baseline adaptations, search and selection settings, and LLM and hardware configurations are detailed in Appendix D.

5.2 Results and Discussion

RQ1 (Effectiveness): Performance Across Recommendation Tasks. Table 1 compares EvoSkillRec with the best manual design, NASRec, and OpenEvolve across all three task families. Under the reported settings, NASRec scores below the strongest manual model on all six benchmarks. OpenEvolve exceeds the best manual result on all three CTR benchmarks but scores below it on MTL and MDL. EvoSkillRec has the highest reported test AUC in each column, with gains of 0.0001–0.0085 over the strongest alternative. These are point estimates from single runs, and the smallest margins do not establish a reliable ordering. The comparison concerns the evaluated implementations under the reported protocols. Because the methods differ in search spaces and proposal mechanisms, it does not isolate which design choice causes the performance differences. Per-dataset results are in Appendix E.1.

Table 2: Comparison with single-space variants. All values are test AUC; MTL reports the mean over tasks and MDL the overall AUC across domains.

Method	Components			CTR			MTL	MDL	
	Skill	Code	Reuse	MovieLens	Books	Beauty	Census	Amazon	MovieLens
Skill-space only	✓	✗	✓	0.7512	0.6647	0.6639	0.9737	0.7019	0.7924
Code-space only	✗	✓	✓	0.7520	0.6689	0.6840	0.9730	0.6807	0.7926
Dual-space w/o reuse	✓	✓	✗	0.7518	0.6657	0.6650	0.9740	0.6812	0.7920
Full model (EVOSKILLREC)	✓	✓	✓	**0.7522**	**0.6708**	**0.6845**	**0.9748**	**0.7033**	**0.7932**

Table 3: Candidate statistics in full-model runs, reported separately for skill-space and code-space proposals. Δ is validation AUC gain over the parent, and $\Delta_{\max}$ is the largest observed gain. Positive-gain rate is the fraction of evaluated candidates with $\Delta > 0$. Survivor yield is the fraction of all proposed candidates selected as survivors; duplicates and failed candidates count as non-survivors.

Benchmark	Positive-gain rate $P(\Delta > 0)$ (%)		Max. validation gain $\Delta_{\max}$		Survivor yield (%)		Code skills
	Skill	Code	Skill	Code	Skill	Code	promoted (#)
CTR-MovieLens	17.7	**25.7**	**+0.001470**	+0.001171	**40.9**	38.1	43
CTR-Beauty	**19.0**	18.9	**+0.016589**	+0.006988	**45.2**	21.6	16
CTR-Books	17.3	**18.1**	+0.005369	**+0.008282**	**44.1**	26.0	19
MTL-Census	29.0	**35.1**	**+0.000655**	+0.000229	**44.3**	35.1	18
MDL-Amazon	**56.2**	45.8	**+0.011990**	+0.009539	3.1	**62.5**	45
MDL-MovieLens	**31.6**	28.6	**+0.004594**	+0.001665	6.2	**50.6**	45

RQ2 (Evolution dynamics): How Do Architectures Evolve? Figure 2 plots best-so-far validation AUC against evolution round for four representative benchmarks, annotating improvements with their operators and the space that proposed each candidate.

Two patterns stand out. First, improvements arise from both spaces, with their timing varying across benchmarks and stages, even within CTR. This motivates keeping both mechanisms available throughout evolution. Second, synthesis can support subsequent reuse: on CTR-MovieLens, a branch invented at round 3 is reused by the round-6 candidate, which improves best-so-far validation AUC. A component from a non-incumbent candidate can therefore reappear in a later improving architecture. Appendix E.2 provides the complete generation-by-generation trajectories and per-dataset operator records.

RQ3 (Mechanism ablation): What Does Each Mechanism Contribute? We compare single-space variants and a dual-space variant without reuse (Table 2), and group candidates from full-model runs by their proposing space (Table 3). The full model's lead over all three variants across six benchmarks supports coupling the spaces with reuse. The mechanism statistics suggest complementary roles in search: skill-space proposals yield larger $\Delta_{\max}$ on five benchmarks, indicating substantial gains from composition, while code space's higher survivor yield on MDL suggests that synthesis sustains exploration when compatibility constraints and duplicate filtering restrict recombination. These contributions interact across generations: skill-space gains can build on earlier code-space inventions. On CTR-MovieLens, a round 3 invention improves validation AUC when reused in round 6, illustrating how promotion extends an invention's value beyond its original candidate. Ablation details and further analysis of the candidate statistics are provided in Appendix E.3.

RQ4: Multi-Objective Evolution. Generative recommenders now scale to billions of parameters (Zhou et al., 2025), so hardware efficiency is a design constraint and model FLOPs utilization (MFU) is reported alongside accuracy (Zhou et al., 2025; Zhang et al., 2026b) in multiple tech reports. We therefore test whether EVOSKILLREC can optimize a compound objective, seeding evolution from a RankMixer backbone (Zhu et al., 2025) on QK-Video for 30 rounds with an objective jointly rewarding AUC and MFU. Eligibility constraints are applied during candidate selection rather than folded into a scalarized reward. Most evolved candidates dominate the seed on *both* axes, so the two objectives improve together rather than trade off (Figure 3). Five eligible candidates lie on the Pareto front. Among these, only P1 (round 19) retains both improvements on the test set (Table 4); Figure 5 shows how the evolved architectures differ from the backbone. The front exposes the trade-off mechanism: compute-dense operators raise utilization, whereas low-rank routing preserves

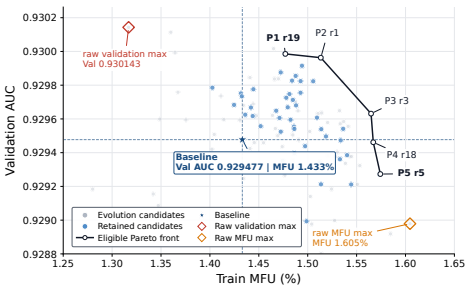

Figure 3: Accuracy–MFU trade-off on QK-Video. Grey points are all evolution candidates, blue points retained candidates, and the black curve the eligible Pareto front. Diamonds mark the raw validation-AUC and MFU maxima.

Model	Val. AUC	Test		Training MFU	
		AUC	ΔAUC ($\times 10^{-4}$)	MFU (%)	ΔMFU (pp)
Baseline	0.929 48	0.935 80	0.0	1.433	0.000
P1 (r19)	0.929 99	0.935 98	1.7	1.477	0.044
P2 (r1)	0.929 96	0.935 73	−0.7	1.513	0.080
P3 (r3)	0.929 63	0.935 70	−1.0	1.565	0.132
P4 (r18)	0.929 46	0.935 40	−4.0	1.567	0.134
P5 (r5)	0.929 27	0.935 61	−1.9	1.574	0.141

Table 4: Performance and training efficiency of the baseline (RankMixer) and eligible Pareto-front candidates. All Δ values are relative to the baseline; MFU deltas are in percentage points (pp). Architectural details for P1–P5 are shown in Figure 5; r_N denotes the round that produced each candidate.

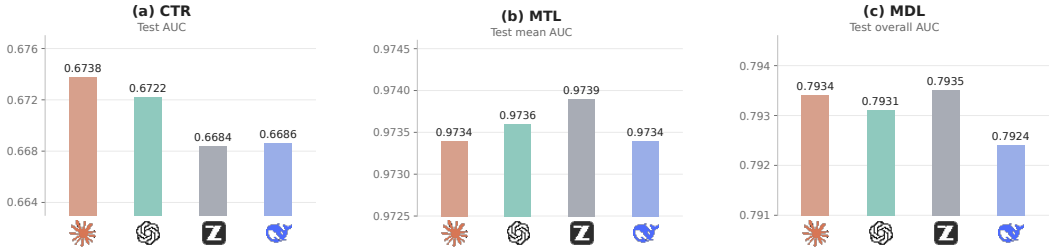

Figure 4: Effect of the LLM backbone. AUC of Claude Opus 4.8, GPT-5.6 (SOL), DeepSeek-V4-Pro and GLM-5.2 on CTR (Amazon-Books), multi-task learning (MTL, Census-Income) and multi-domain learning (MDL, MovieLens).

accuracy. Evolution is therefore effective for compound objectives, not only for single metrics; candidate-level details and the industrial-scale analysis are in Appendix E.4.

RQ5: Effect of the LLM Backbone. Because the backbone generates architectural proposals, its choice can affect which mechanisms evolution discovers. We compare Claude Opus 4.8, GPT-5.6 (SOL), DeepSeek-V4-Pro, and GLM-5.2 at their highest reasoning settings, with identical initial architectures, environments, and round budgets. Figure 4 reports test AUC on Amazon-Books (CTR), Census-Income (MTL), and MovieLens (MDL). We find no single model leads on all three tasks: Claude is strongest on CTR (0.6738), GLM on MTL (0.9739) and MDL (0.7935), where its 0.0001 margin over Claude on MDL is a tie; all runs are single-seed, so backbone gaps below $\sim 5 \times 10^{-4}$ AUC are ties as well. GLM is competitive with both closed-weight models and ahead of them on two tasks, so the gap between open- and closed-weight models is model-specific rather than categorical. DeepSeek is the weakest backbone overall, ranking last on MDL and jointly last on MTL, and its proposals also succeed at a consistently lower rate, so its deficit lies in how reliably it produces valid candidates, not only in their quality. Since differences between backbones are far smaller than the gain over the seed, the improvements above come mainly from the search procedure rather than from any single LLM. Per-backbone candidate tables are in Appendix E.5.

6 Conclusion

We presented EvoSkillRec, a skill-genome framework for recommender architecture discovery. It breaks established recommenders down into executable skills, evolves them through coupled skill-space recombination and code-space invention, and promotes validated discoveries back into a growing library. EvoSkillRec is the only method we compare that improves on the strongest hand-designed baseline on all six benchmarks, and it extends to accuracy–efficiency co-optimization in generative recommendation. The two search spaces are complementary, and future work will study cross-task skill transfer and industrial deployment.

7 AI USE STATEMENT

Large language models are themselves a component of EVOSKILLREC. During library construction, an LLM assists in semantically grouping candidate modules and generating skill-card annotations; during evolution, both the architecture planner and code synthesizer are implemented through LLM calls. The main experiments use GPT-5.5 via the Codex CLI, while the LLM-backbone study (Section 5.2, RQ5) evaluates four additional backbones: Claude Opus 4.8, GPT-5.6 (SOL), DeepSeek-V4-Pro, and GLM-5.2. AI tools also assist in manuscript preparation. We use ChatGPT to polish parts of the paper and to edit the text for clarity and readability.

REFERENCES

Census-Income (KDD). UCI Machine Learning Repository, 2000. DOI: https://doi.org/10.24432/C5N30T.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

Zheng Chai, Huizhi Yang, Di Chen, Lele Yu, Xiang Sun, Qin Ren, Bo Han, Hui Lu, Xionghang Xie, Yaocheng Tan, Xijun Xiao, Sijun Zhang, Wenlin Zhao, Shiru Ren, Peng Xu, Yuchao Zheng, and Di Wu. Longer: Scaling up long sequence modeling in industrial recommenders. *arXiv preprint arXiv:2505.04421*, 2025.

Jianxin Chang, Chenbin Zhang, Yiqun Hui, Dewei Leng, Yanan Niu, Yang Song, and Kun Gai. PEPNet: Parameter and embedding personalized network for infusing with personalized prior information. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 2023.

Bo Chen, Yichao Wang, Zhirong Liu, Ruiming Tang, Wei Guo, Hongkun Zheng, Weiwei Yao, Muyu Zhang, and Xiuqiang He. Enhancing explicit and implicit feature interactions via information sharing for parallel deep CTR models. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, pp. 7–10, 2016.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

Datawhale. Torch-rechub. `https://github.com/datawhalechina/torch-rechub`, 2022. A PyTorch-based recommender system framework providing easy-to-use and extensible solutions.

Qin Ding, Kevin Course, Linjian Ma, Jianhui Sun, Rouchen Liu, Zhao Zhu, Chunxing Yin, Wei Li, Dai Li, Yu Shi, Xuan Cao, Ze Yang, Han Li, Xing Liu, Bi Xue, Hongwei Li, Rui Jian, Daisy Shi He, Jing Qian, Matt Ma, Qunshu Zhang, and Rui Li. Bending the scaling law curve in large-scale recommendation systems. *arXiv preprint arXiv:2602.16986*, 2026.

Huan Gui, Ruoxi Wang, Ke Yin, Long Jin, Maciej Kula, Taibai Xu, Lichan Hong, and Ed H Chi. Hiformer: Heterogeneous feature interactions learning with transformers for recommender systems. *arXiv preprint arXiv:2311.05884*, 2023.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: A factorization-machine based neural network for ctr prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pp. 1725–1731, 2017.

Mingming Ha, Guanchen Wang, Linxun Chen, Xuan Rao, Yuexin Shi, Tianbao Ma, Zhaojie Liu, Yunqian Fan, Zilong Lu, Yanan Niu, Han Li, and Kun Gai. Unimixer: A unified architecture for scaling laws in recommendation systems. *arXiv preprint arXiv:2604.00590*, 2026.

F. Maxwell Harper and Joseph A. Konstan. The MovieLens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems*, 5(4):1–19, 2015.

Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web (WWW)*, 2016.

Xu Huang, Hao Zhang, Zhifang Fan, Yunwen Huang, Zhuoxing Wei, Zheng Chai, Jinan Ni, Yuchao Zheng, and Qiwei Chen. Mixformer: Co-scaling up dense and sequence in industrial recommenders. *arXiv preprint arXiv:2602.14110*, 2026a.

Yunwen Huang, Shiyong Hong, Xijun Xiao, Jinqiu Jin, Xuanyuan Luo, Zhe Wang, Zheng Chai, Shikang Wu, Yuchao Zheng, and Jingjian Lin. Hyformer: Revisiting the roles of sequence modeling and feature interaction in ctr prediction. *arXiv preprint arXiv:2601.12681*, 2026b.

Yuchen Jiang, Jie Zhu, Xintian Han, Hui Lu, Kunmin Bai, Mingyu Yang, Shikang Wu, Ruihao Zhang, Wenlin Zhao, Shipeng Bai, Sijin Zhou, Huizhi Yang, Tianyi Liu, Wenda Liu, Ziyan Gong, Haoran Ding, Zheng Chai, Deping Xie, Zhe Chen, Yuchao Zheng, and Peng Xu. Tokenmixer-large: Scaling up large ranking models in industrial recommenders. *arXiv preprint arXiv:2602.06563*, 2026.

Wang-Cheng Kang and Julian McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining*, pp. 197–206. IEEE, 2018.

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

Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. In *Handbook of Evolutionary Machine Learning*, pp. 331–366. Springer, 2023.

Honghao Li, Xianquan Wang, Zibin Zhang, Yi Zhang, Kangyi Lin, and Yiwen Zhang. UniRank: Benchmarking Ranking Models for Unified Sequential Modeling and Feature Interaction. *arXiv preprint arXiv:2607.19987*, 2026.

Jinyun Li, Huiwen Zheng, Yuanlin Liu, Minfang Lu, Lixia Wu, and Haoyuan Hu. AdaptDHM: Adaptive distribution hierarchical model for multi-domain CTR prediction. *arXiv preprint arXiv:2211.12105*, 2022a.

Kaiyuan Li, Yanhua Cheng, Dongdong Mao, Yanxiang Zeng, Yongxiang Tang, Chao Wang, Xialong Liu, and Peng Jiang. Infnet: A task-aware information flow network for large-scale recommendation systems. *arXiv preprint arXiv:2508.11565*, 2025a.

Xiaopeng Li, Fan Yan, Xiangyu Zhao, Yichao Wang, Bo Chen, Huifeng Guo, and Ruiming Tang. HAMUR: Hyper adapter for multi-domain recommendation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM)*, 2023.

Xiaopeng Li, Jingtong Gao, Pengyue Jia, Yichao Wang, Wanyu Wang, Yejing Wang, Yuhao Wang, Huifeng Guo, and Ruiming Tang. Scenario-wise rec: A multi-scenario recommendation benchmark. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management (CIKM)*, 2025b. doi: 10.1145/3746252.3761137.

Zelong Li, Jianchao Ji, Yingqiang Ge, and Yongfeng Zhang. Autolossgen: Automatic loss function generation for recommender systems. *arXiv preprint arXiv:2204.13160*, 2022b.

Mingyang Liu, Yong Bai, Zhangming Chan, Sishuo Chen, Xiang-Rong Sheng, Han Zhu, Jian Xu, and Xinyang Chen. Est: Towards efficient scaling laws in click-through rate prediction via unified modeling. *arXiv preprint arXiv:2602.10811*, 2026.

Jiaqi Ma, Zhe Zhao, Xinyang Yi, Jilin Chen, Lichan Hong, and Ed H. Chi. Modeling task relationships in multi-task learning with multi-gate mixture-of-experts. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1930–1939, 2018a.

Xiao Ma, Liqin Zhao, Guan Huang, Zhi Wang, Zelin Hu, Xiaoqiang Zhu, and Kun Gai. Entire space multi-task model: An effective approach for estimating post-click conversion rate. In *Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR)*, 2018b.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton van den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2015.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP)*, 2019.

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

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pp. 4780–4789, 2019.

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

Qijie Shen, Wanjie Tao, Jing Zhang, Hong Wen, Zulong Chen, and Quan Lu. SAR-Net: A scenario-aware ranking network for personalized fair recommendation in hundreds of travel scenarios. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

Xiang-Rong Sheng, Liqin Zhao, Guorui Zhou, Xinyao Ding, Binding Dai, Qiang Luo, Siran Yang, Jingshan Lv, Chi Zhang, Hongbo Deng, and Xiaoqiang Zhu. One model to serve all: Star topology adaptive recommender for multi-domain CTR prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

Qingquan Song, Dehua Cheng, Hanning Zhou, Jiyan Yang, Yuandong Tian, and Xia Hu. Towards automated neural interaction discovery for click-through rate prediction. *arXiv preprint arXiv:2007.06434*, 2020.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp. 1161–1170, 2019.

Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. Progressive layered extraction (ple): A novel multi-task learning (mtl) model for personalized recommendations. In *Proceedings of the 14th ACM Conference on Recommender Systems*, pp. 269–278, 2020.

Fangye Wang, Guowei Yang, Xiaojiang Zhou, Song Yang, and Pengjie Wang. Query-mixed interest extraction and heterogeneous interaction: A scalable ctr model for industrial recommender systems. *arXiv preprint arXiv:2602.09387*, 2026a.

Haochen Wang, Yi Wu, Daryl Chang, Li Wei, and Lukasz Heldt. Self-evolving recommendation system: End-to-end autonomous model optimization with llm agents. *arXiv preprint arXiv:2602.10226*, 2026b.

Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & cross network for ad click predictions. In *Proceedings of the ADKDD'17*, pp. 1–7, 2017.

Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference (WWW)*, 2021.

Dongbo Xi, Zhen Chen, Peng Yan, Yinger Zhang, Yongchun Zhu, Fuzhen Zhuang, and Yu Chen. Modeling the sequential dependence among audience multi-step conversions with multi-task learning in targeted display advertising. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining (KDD)*, 2021.

Jun Xiao, Hao Ye, Xiangnan He, Hanwang Zhang, Fei Wu, and Tat-Seng Chua. Attentional factorization machines: Learning the weight of feature interactions via attention networks. *arXiv preprint arXiv:1708.04617*, 2017.

Xuanhua Yang, Xiaoyu Peng, Penghui Wei, Shaoguo Liu, Liang Wang, and Bo Zheng. AdaSparse: Learning adaptively sparse structures for multi-domain click-through rate prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management (CIKM)*, 2022.

Yantao Yu, Sen Qiao, Lei Shen, Bing Wang, and Xiaoyi Zeng. Beyond dense connectivity: Explicit sparsity for scalable recommendation. *arXiv preprint arXiv:2604.08011*, 2026.

Guanghu Yuan, Fajie Yuan, Yudong Li, Beibei Kong, Shujie Li, Lei Chen, Min Yang, Chenyun Yu, Bo Hu, Zang Li, Yu Xu, and Xiaohu Qie. Tenrec: A large-scale multipurpose benchmark dataset for recommender systems. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, volume 35, 2022.

Barry Zhang, Keith Lazuka, and Mahesh Murag. Equipping agents for the real world with Agent Skills. Anthropic Engineering Blog, 2025. URL `https://www.anthropic.com/engineering/equipping-agents-for-the-real-world-with-agent-skills`.

Ruifeng Zhang, Zexi Huang, Zikai Wang, Ke Sun, Bohang Zheng, Zhen Ouyang, Huimin Xie, Phil Shen, Junlin Zhang, Wentao Guo, and Qinglei Wang. Zenith: Scaling up ranking models for billion-scale livestreaming recommendation. *arXiv preprint arXiv:2601.21285*, 2026a.

Tunhou Zhang, Dehua Cheng, Yuchen He, Zhengxing Chen, Xiaoliang Dai, Liang Xiong, Feng Yan, Hai Li, Yiran Chen, and Wei Wen. Nasrec: Weight sharing neural architecture search for recommender systems. *arXiv preprint arXiv:2207.07187*, 2022.

Tunhou Zhang, Dehua Cheng, Yuchen He, Zhengxing Chen, Xiaoliang Dai, Liang Xiong, Yudong Liu, Feng Cheng, Yufan Cao, Feng Yan, Hai Li, Yiran Chen, and Wei Wen. Towards automated model design on recommender systems. *arXiv preprint arXiv:2411.07569*, 2024a.

Zhaoqi Zhang, Haolei Pei, Jun Guo, Tianyu Wang, Yufei Feng, Hui Sun, Shaowei Liu, and Aixin Sun. Onetrans: Unified feature interaction and sequence modeling with one transformer in industrial recommender. In *Proceedings of the ACM Web Conference 2026*, pp. 8162–8170, 2026b.

Zijian Zhang, Shuchang Liu, Jiaao Yu, Qingpeng Cai, Xiangyu Zhao, Chunxu Zhang, Ziru Liu, Qidong Liu, Hongwei Zhao, Lantao Hu, Peng Jiang, and Kun Gai. M3oE: Multi-domain multi-task mixture-of-experts recommendation framework. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, 2024b.

Guorui Zhou, Xiaoqiang Zhu, Chenru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. Deep interest network for click-through rate prediction. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1059–1068, 2018.

Guorui Zhou, Na Mou, Ying Fan, Qi Pi, Weijie Bian, Chang Zhou, Xiaoqiang Zhu, and Kun Gai. Deep interest evolution network for click-through rate prediction. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence*, pp. 5941–5948, 2019.

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

Yifeng Zhou, Yuehong Hu, Zhixiang Feng, Junwei Pan, Kaihui Wu, Hanyong Li, Shangyu Zhang, Shudong Huang, Zhangbin Zhu, Chengguo Yin, Haijie Gu, and Jie Jiang. Tokenformer: Unify the multi-field and sequential recommendation worlds. *arXiv preprint arXiv:2604.13737*, 2026.

Jie Zhu, Zhifang Fan, Xiaoxie Zhu, Yuchen Jiang, Hangyu Wang, Xintian Han, Haoran Ding, Xinmin Wang, Wenlin Zhao, Zhen Gong, Huizhi Yang, Zheng Chai, Zhe Chen, Yuchao Zheng, Qiwei Chen, Feng Zhang, Xun Zhou, Peng Xu, Xiao Yang, Di Wu, and Zuotao Liu. Rankmixer: Scaling up ranking models in industrial recommenders. *arXiv preprint arXiv:2507.15551*, 2025.

Jieming Zhu, Jinyang Liu, Shuai Yang, Qi Zhang, and Xiuqiang He. Open benchmarking for click-through rate prediction. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM)*, 2021.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Appendix Contents

A Skill Representation and Library Construction **16**

A.1 Skill Card Fields and Tensor Interfaces 16

A.2 Model-to-Skill Compilation and Validation 18

A.3 Curated Library for CTR, MTL, and MDL 18

A.4 Curated Library for Generative Ranking 20

A.5 Examples of Curated and Generated Skill Cards 21

B Details of the Cumulative Evolution Loop **23**

B.1 Skill Retrieval and Compatibility Filtering 23

B.2 Typed Skill-Space Operators 23

B.3 Code-Space Planning and Validation 23

B.4 Adaptive Proposal Budget 24

B.5 Cumulative Evolution Loop and Skill Promotion 24

C Tasks, Datasets, and Baselines **25**

C.1 Tasks and Evaluation Objectives 25

C.2 Datasets and Data Splits 25

C.3 Baseline Models and Source Architectures 26

D Implementation and Evaluation Protocol **28**

D.1 Training Protocols and Initial Architectures 28

D.2 Search Budgets and Survivor Selection 30

D.3 LLM Settings for Planning and Code Generation 31

D.4 Hardware Setup and MFU Measurement 31

D.5 Implementation of the Automated Search Baselines 32

E Supplementary Experimental Results **34**

E.1 RQ1: Predictive Performance Across Recommendation Tasks 34

E.2 RQ2: Architecture Evolution Across Generations 36

E.3 RQ3: Contributions of the Search Spaces and Skill Reuse 40

E.4 RQ4: Accuracy–Efficiency Trade-offs on QK-Video 41

E.5 RQ5: Effect of the LLM Backbone 44

A SKILL REPRESENTATION AND LIBRARY CONSTRUCTION

Every reusable module in the skill library comes with a skill card: a machine-readable YAML manifest that records what the module computes, which named tensors it consumes and produces, and how it can be retrieved and composed with other skills. The library has two tiers. Tier-1 consists of curated components obtained by decomposing existing recommendation models, together with supporting adapters and training objectives. Tier-2 consists of modules synthesized by the LLM during Code Space exploration and retained for later reuse. Both tiers use the same named-tensor interface, so the planner can treat them uniformly; Tier-2 cards additionally store the validation and promotion evidence collected when the module was admitted. The following sections describe the card interface, the compilation and validation process, the curated libraries, and examples from both tiers.

A.1 SKILL CARD FIELDS AND TENSOR INTERFACES

Table 5 groups the fields of a skill card by function. Two serialization conventions coexist in the implementation. Curated cards usually identify a skill by `name` and declare its interface with `inputs` and `outputs`. Generated cards also carry a `skill_id` and declare the interface with `input_signature` and `output_signature`. The two conventions encode the same identity and interface information. Not every field is mandatory: the table lists the field groups that the library supports, and a compact curated card may leave its full tensor contract to the `SkillSpec` registered in Python.

Tensor shapes and compatibility requirements. A tensor contract binds a name to a shape and a dtype. For example, `field_embeddings` has shape $[B, F, D]$, where B is the batch size, F the number of feature fields, and D the embedding dimension. Constructor arguments are supplied by the genome node that invokes the skill; the example invocation in a card may use placeholders such as `${embedding_dim}`. Because dimensions are symbolic, a skill can be re-instantiated under a different feature schema, provided its declared constraints hold. This concerns the module definition only: trained weights are not assumed to transfer, and compatibility is checked for each concrete schema rather than guaranteed for all of them. Similarly, the upstream and downstream neighbors listed in a card serve only as retrieval hints. A connection in an actual genome is valid only when the instantiated tensor shapes and context keys match.

Table 5: Fields of a skill card, grouped by function. The Validation and Promotion groups appear only in Tier-2 (generated) cards.

Field group	**Serialized fields**	**Content and purpose**
Identity and role	`name`, `skill_id`, `skill_name`, `category`, `description` / `function_description`	Stable identifier, functional category, and a natural-language description of the module.
Tensor interface	`inputs`, `outputs`; or `input_signature`, `output_signature`	Name, shape, and dtype of each input and output tensor; generated signatures may add a semantic annotation.
Modeling context	`task_types` / `applicable_tasks`, `inductive_bias`, `failure_signatures`, `failure_modes_addressed`	Applicable tasks, modeling assumptions, known interface failures, and the modeling weaknesses the skill is meant to address.
Retrieval	`retrieval`: `summary`, `use_when`, `avoid_when`, `task_types`, `architecture_roles`, `input_modalities`, `output_semantics`, `objectives`, `model_families`, `aliases`, `query_examples`	Descriptions and search terms used to match a modeling need to a reusable component.
Composition	`composition`: `requires`, `produces`, `common_upstream`, `common_downstream`, `example_genome_fragment`	Context keys read and written, typical neighboring skills, and an example invocation with constructor parameters.
Usage and cost	`constraints`, `cost`, `good_for`, `bad_for`, `compatible_with`, `mutation_roles`	Optional restrictions, computational cost, compatibility hints, and suggested mutation roles.
Implementation and provenance	`source`; or `implementation_path`, `class_name`, `source_proposal_id`, `parent_skills`, `mutation_lineage`	Tier-1: original implementation and extraction type. Tier-2: location of the generated code and its proposal lineage.
Validation (Tier-2)	`validation_status`, `validation_results`, `portability`	Results of static, forward/backward, and shape checks; where available, tests under other feature schemas and the resulting reuse scope.
Promotion (Tier-2)	`promotion_status`, `promoted_at`, `candidate_id`, `candidate_status`, `candidate_metrics`, `architecture_fingerprint`, `code_hash`	Evidence from the candidate genome that contained the skill, plus identifiers that trace the retained architecture and code.

A.2 Model-to-Skill Compilation and Validation

Parsing extracts candidate modules (embedding tables, field encoders, feature-crossing blocks, sequence encoders, attention aggregators, fusion layers, towers, gates, experts, losses, and samplers) by combining static code analysis with LLM-assisted semantic grouping, and accepts a module only if its inputs, outputs, parameters, and call sites can be cleanly isolated from the rest of the model. *Semantic annotation* writes the card fields: function, inductive bias, applicable tasks, failure modes, assumptions, and nearest known model family. *Executable extraction* rewrites the code into a typed `BaseSkill` file exposing exactly the identified inputs, outputs, and core execution logic. *Validation* applies general tests (importability, deterministic construction, shape preservation, differentiability, and serialization), and the strongest test, reconstruction: the compiler rebuilds each seed model from its extracted skills and verifies that end-to-end behavior matches the original implementation; when exact equivalence cannot be established, the skill construction is flagged and the whole process is revisited.

A.3 Curated Library for CTR, MTL, and MDL

The curated library contains **94 skills in 11 categories**. Table 6 reports the full size of each category and shows up to four skill identifiers per category; ellipses indicate omitted entries. Skills are grouped by the `category` field of their cards, and each skill is counted once even if it serves several tasks or model families. Categories are defined by functional role, and two of them need a brief note. `scenario` collects multi-domain components, and `generative` collects semantic-ID and generative ranking components. The name `generative` refers to the modeling paradigm, not to how a skill was obtained: every skill in this table is a curated Tier-1 skill. Fusion adapters are listed under `utility`.

Table 6: Curated Tier-1 skills by category. The Size column gives the full number of skills in each category. Up to four identifiers are shown per category; ... denotes omitted entries. Identifiers and category assignments follow the stored skill manifests.

Category	Size	Role	Example skill identifiers
embedding	4	Categorical and sequential feature embeddings.	field_aware_embedding_adapter, field_embedding, sequence_field_embedding, shared_sequence_embedding
interaction	11	Explicit feature interactions, attention, and gating.	afm_attention_pooling, autoint_attention, bilinear_interaction, cen_feature_gate, ...
tower	2	Dense representation transformations.	mlp_tower, shared_mlp_tower
sequence	19	Sequence encoding, interest modeling, and temporal features.	dien_interest_evolution, din_target_attention, exact_leakyrelu_transformer_encoder, full_augru_evolution, ...
matching	7	Candidate scoring, retrieval, and negative sampling.	all_item_scoring_adapter, candidate_dot_logits, dot_product_logits, in_batch_negative_head, ...
multitask	10	Shared experts, task routing, and task-specific prediction.	aitm_transfer, esmm_chain_head, exact_attention_transfer, mmoe_gate, ...
scenario	11	Domain conditioning, adaptation, and prediction.	adaptdhm_cluster_tower, adasparse_pruned_tower, domain_indicator_adapter, domain_select, ...
generative	8	Quantization, semantic IDs, and generative recommendation.	exact_residual_quantizer_stack, exact_t5_encoder_decoder, kmeans_sinkhorn_initialization, normalized_item_dot_head, ...
head	6	Prediction logits, probabilities, and language-model outputs.	binary_ctr_head, linear_logit, sasrec_pairwise_logits, sequence_lm_head, ...
loss	9	Training objectives and regularization.	bce_loss, covariance_regularizer, dien_auxiliary_interest_loss, hinge_loss, ...
utility	7	Tensor adaptation, normalization, and fusion.	additive_fusion, concat_fusion, dense_feature_path, field_sum, ...
Total	**94**	11 categories; 42 identifiers shown	

A.4 CURATED LIBRARY FOR GENERATIVE RANKING

Table 7 lists the base skills obtained by decomposing the 15 source architectures described in Section C.3. The compiler breaks each model into reusable components and keeps the generic modules they depend on, giving **92 skills in 10 categories**. This library is **maintained separately from the library used in the CTR, MTL, and MDL experiments**. We build it **specifically for generative ranking models**. These models work on unified token sequences and rely on token mixing, attention and model-specific backbone blocks, so they need different operators from classical feature-interaction models. **Interaction (25), embedding (17) and attention (16)** make up **63%** of the library. **54 skills are model-specific** and carry their source model's prefix. The other **38 are shared building blocks**, such as cross networks, pooling, activations and PyTorch primitives.

Table 7: Example skills for generative ranking from the full inventory of 92 skills across 10 categories. The # column reports the full number of skills in each category; up to four identifiers are shown per category, with . . . indicating omitted entries. Model-specific skills are prefixed with their source model.

Category	#	Role	Example skill identifiers
`embedding`	17	Feature embeddings and model-specific feature-to-token transformations.	`est_tokenization`, `feature_embedding`, `feature_embedding_dict`, `hemix_tokenization`, . . .
`interaction`	25	Explicit feature interactions, token mixing, and token-wise feed-forward or expert layers.	`bilinear_interaction`, `bilinear_interaction_v2`, `compressed_interaction_net`, `cross_interaction`, . . .
`attention`	16	Self-, cross-, and target-aware attention, sequence transduction, and channel recalibration.	`din_attention`, `est_cross_attention`, `hemix_hetero_attention`, `hiformer_attention`, . . .
`tower`	3	MLP towers and mixed feed-forward transformations.	`est_mixed_ffn`, `mlp_block`, `onetrans_mixed_ffn`
`block`	10	Composite model-specific backbone blocks.	`est_block`, `hiformer_block`, `hyformer_block`, `infnet_block`, . . .
`pooling`	3	Masked sum/average aggregation and top-k pooling.	`kmax_pooling`, `masked_average_pooling`, `masked_sum_pooling`
`activation`	2	Adaptive and smooth nonlinear activations.	`dice`, `gelu`
`head`	2	Linear logits and per-task output activations.	`logistic_regression`, `output_activation_list`
`primitive`	5	Basic PyTorch layers and containers.	`dropout`, `embedding`, `identity`, `linear`, . . .
`utility`	9	Query processing, sequence representation, aggregation, broadcast, and fusion.	`hyformer_query_boosting`, `hyformer_query_decoding`, `hyformer_query_generation`, `hyformer_sequence_representation`, . . .
Total	**92**	10 categories	

A.5 Examples of Curated and Generated Skill Cards

We show one card from each tier: a component decomposed from an existing model (Tier-1) and a module generated by the Code Space LLM pipeline (Tier-2). Both listings are excerpts of stored manifests. Some fields are omitted for space; the retained fields keep their original values and nesting.

Curated skill example: factorization-machine interaction. The `fm_interaction` skill isolates the second-order interaction branch of DeepFM by wrapping the existing FM layer. The card (Listing 1) records the source location and extraction type, the second-order inductive bias, the tensor interface, and a minimal invocation.

```
name: fm_interaction
category: interaction
description: Compute second-order factorization-machine interactions over
     field embeddings.
inputs:
  - name: field_embeddings
    shape: "[batch_size, num_fields, embedding_dim]"
    dtype: float32
outputs:
  - name: fm_output
    shape: "[batch_size, 1]"
    dtype: float32
task_types: [ctr, ranking]
inductive_bias: [second_order_feature_interactions]
failure_signatures: [rank_not_3]
constraints:
  input_rank: 3
cost:
  parameters: 0
  compute: "O(batch_size * num_fields * embedding_dim)"
source:
  file: torch_rechub/basic/layers.py
  class_or_function: FM
  extraction_type: wrapped
retrieval:
  summary: "Use this for low-cost second-order feature interactions over
    field embeddings in CTR/ranking models."
  task_types: [ctr, ranking]
  architecture_roles: [interaction, second_order, factorization_machine]
  input_modalities: [field_embeddings]
  output_semantics: [interaction_logit, fm_output]
composition:
  requires: [field_embeddings]
  produces: [fm_output]
  common_upstream: [field_embedding]
  common_downstream: [concat_fusion, binary_ctr_head]
  example_genome_fragment:
    skill: fm_interaction
    params:
      input_key: field_embeddings
      output_key: fm_output
```

Listing 1: Tier-1 card example: FM interaction extracted from an existing implementation.

Generated skill example: covariance-based field mixing. The generated field-mixing skill in Listing 2 was proposed while exploring a PLE-based multi-task genome. The design hypothesis was that suppressing field correlations that conflict across tasks, before the representation is shared, would reduce negative transfer. The module first normalizes the field embeddings and estimates cross-field correlations from the current batch. Two small networks map these statistics to the factors of a field-routing matrix, and a learned gate interpolates between this matrix and the identity. Channel projections and a residual connection complete the update. Both the input and the output

are field_embeddings with shape $[B, F, D]$, so the module can be inserted between the feature path and the flattening step without changing any downstream interface. The card (Listing 2) is shown as follows:

```
skill_id: macro_field_covariance_shrinkage_v1
name: macro_field_covariance_shrinkage_v1
category: routing
task_types: [multitask]
failure_modes_addressed: [negative_transfer_between_tasks]
input_signature:
  - name: field_embeddings
    shape: [batch_size, num_fields, embedding_dim]
    dtype: float32
output_signature:
  - name: field_embeddings
    shape: [batch_size, num_fields, embedding_dim]
    dtype: float32
implementation_path: recskill/generated_skills/
    macro_field_covariance_shrinkage_v1.py
class_name: FieldCovarianceShrinkage
source_proposal_id: proposal_field_covariance_shrinkage_v1
validation_status: passed
validation_results:
  static_check: {passed: true, issues: []}
  unit_tests:
    passed: true
    checks: {import: true, instantiate: true, forward: true, backward:
    true}
    issues: []
  shape_tests:
    passed: true
    output_shapes: {field_embeddings: [2, 3, 16]}
    issues: []
  portability_tests:
    passed: true
portability:
  scope: schema_agnostic
  reuse_enabled: true
  constraints: {min_num_fields: 1}
composition:
  requires: [field_embeddings]
  produces: [field_embeddings]
  example_genome_fragment:
    skill: macro_field_covariance_shrinkage_v1
    params:
      covariance_rank: 8
      embedding_dim: ${embedding_dim}
      num_fields: ${num_fields}
      shrinkage_hidden_dim: 16
promotion_status: promoted
promoted_at: '2026-08-04T17:21:03+00:00'
candidate_id: round1_code_macro_field_covariance_shrinkage_v1_p0
candidate_status: survivor
candidate_metrics:
  validation_mean_auc: 0.9720098642670008
architecture_fingerprint: 1835d961a9028a03a224
```

Listing 2: Tier-2 card example: a generated and promoted field-mixing module.

Because the two cards follow the same tensor contract, the planner can wire either skill to any compatible neighbor. The tiers differ in provenance and in the evidence attached to it. A Tier-1 card records the source component and how it was extracted; a Tier-2 card additionally records the generated implementation, its verification results, and the candidate through which it entered the library.

B Details of the Cumulative Evolution Loop

This section follows the generation process: retrieve compatible skills, apply typed operators or synthesize new modules, allocate the proposal budget, and retain validated candidates and reusable skills.

B.1 Skill Retrieval and Compatibility Filtering

Skill-space retrieval is rule-based. *Hard filters* discard any skill that is incompatible with the current task, whose required input keys the parent genome cannot supply directly or through a one-hop upstream producer, whose implementation is not loadable, that belongs to a category excluded from direct search (e.g., losses, objectives, terminal heads), or that would duplicate a skill already present in the genome. The survivors are ranked by Eq. 4, whose terms are: ϕ_{task} (task-type compatibility with $\mathcal{T}$), ϕ_{io} (availability of the required input keys in $\mathcal{X}_G$ or one hop upstream), ϕ_{ret} (textual similarity between the card's retrieval block and the diagnosis r_t), ϕ_{bias} (whether the skill's inductive bias is known to counter the failure signature in r_t), and $\bar{m}(\mathcal{S}_i)$ (the skill's historical average validated improvement); the weights $w_1, \ldots, w_5$ are fixed across tasks. The top-n scoring skills form the pool from which the genome is evolved, subject to a per-category cap that prevents the pool from collapsing onto a single skill category. No vector index (e.g., FAISS, Annoy, Milvus) is used over skill representations: retrieval relies only on skill-card metadata, string matching against the diagnosis, and tensor-key compatibility.

B.2 Typed Skill-Space Operators

Table 8 specifies the four skill-space operator categories of Section 4.2.1 with their preconditions. Every precondition is checked statically against the genome DAG and the skill cards' typed interfaces before a candidate is constructed, so a proposal that violates one is rejected without consuming any training budget.

Table 8: Typed mutation operators used by skill-space evolution.

Operator	**Preconditions**	**Example**
Add	The new skill's required input keys are present in $\mathcal{X}_G$ or producible by one upstream hop, and its output key is accepted by the fusion layer.	Insert a target-aware attention branch before the tower.
Replace	Input and output signatures are compatible, and the output key is preserved so downstream wiring is untouched.	Replace mean pooling with target-aware attention; replace the FM branch with identity (removal).
Hybridize	Two branches expose compatible fusion points and the merged graph remains acyclic and typed.	Combine a DeepFM interaction branch with a DIN interest branch.
Specialize	Scenario or task metadata is available, and the change alters only hyperparameters, not `skill_id` or wiring.	Attach a domain-specific adapter; widen or narrow a branch under a different capacity budget.

B.3 Code-Space Planning and Validation

Design brief. Equation 5 bundles five inputs. The parent genome G_t fixes the nodes, edges, and per-node interfaces that a proposal must respect. The diagnosis r_t records the observed failure modes together with the node each is attributed to. The memory window $\mathcal{M}_{t-\Delta:t}$ is read in both directions: promoted skills show which directions work, while failed proposals serve as negative examples. The tensor keys $\mathcal{X}_{G_t}$ bound what a proposal may read, since only keys already present in the tensor context are admissible, so a module consumes intermediate tensors rather than raw features. The constraints $\mathcal{C}$ are the deployment or research constraints of Section 3. A proposal must also be a macro-level change wired in one of four ways: replacing a node, inserting between two nodes, branching into the fusion layer, or replacing that layer. The architecture planner's sketches describe structure rather than code: the missing mechanism, the innovation lane it belongs to (interaction,

routing, fusion, embedding, or sequence), the failure mode it targets, the nodes and keys it touches, its wiring, and its parameter budget.

Code-space screening. A sketch is discarded if the parent genome cannot supply its declared inputs, if its wiring is not one of the permitted macro-level transformations, or if it duplicates a structure already tried. Synthesized code is then checked in five ways: (1) static analysis rejects non-importable code and any file, network, subprocess, or dynamic-evaluation capability; (2) the declared skill-card contract must match the realized interface and tensor shapes; (3) the code must realize the macro-level change its sketch declared; (4) the module must complete forward and backward passes with valid gradients on synthetic inputs; and (5) a portability test requires it to remain executable as the number and type of input fields change.

B.4 Adaptive Proposal Budget

The controller divides a fixed candidate budget between skill-space recombination and code-space invention. It increases the code-space quota when progress stalls, while retaining a minimum skill-space quota. The per-run quotas, plateau thresholds, and reset rule are specified in Section D.2.

B.5 Cumulative Evolution Loop and Skill Promotion

Algorithm 1 combines the two proposal spaces with candidate validation, evaluation, survivor selection, and library updates. Training and selection settings are specified in Sections D.1 and D.2.

Algorithm 1 Cumulative evolution loop. $\mathcal{O} = \{\text{ADD}, \text{REPLACE}, \text{HYBRIDIZE}, \text{SPECIALIZE}\}$ is the skill-space operator set (Section 4.2.1), $\mathcal{P}_t$ the skills synthesized in code space, $G_{\mathcal{S}}$ the candidate genome hosting skill $\mathcal{S}$, and Top_K the K highest-scoring elements of a set; $\ell(G)$ and $o(G)$ record a candidate's lineage and validation outcome.

Require: seed genome G_0; initial skill library $\mathcal{L}_0$; instance $(\mathcal{D}, \mathcal{T}, \mathcal{C})$; generations T; budget N; survivor size K and tolerance δ; plateau window k and threshold ϵ
Ensure: best genome $G^\star$; grown library $\mathcal{L}_T$; memory $\mathcal{M}_T$

$s(G_0) \leftarrow \mathcal{E}(G_0; \mathcal{D}, \mathcal{T}, \mathcal{C})$; $\Sigma_0 \leftarrow \{G_0\}$; $\mathcal{M}_0 \leftarrow \{(G_0, s(G_0))\}$; $G^\star \leftarrow G_0$
for $t = 1$ **to** T **do**
  $r_t \leftarrow \text{DIAGNOSE}(\Sigma_{t-1}, \mathcal{M}_{t-1})$ ▷ *failure modes attributed to genome nodes*
  $(n_{\text{skill}}, n_{\text{code}}) \leftarrow \text{ALLOCATE}(N, \mathcal{M}_{t-1}, \epsilon, k)$ ▷ *shift toward code space on a plateau*
  $\mathcal{S}_t \leftarrow \text{RETRIEVE}(\mathcal{L}_{t-1}, \Sigma_{t-1}, r_t)$ ▷ *top-n skills under Eq. 4*
  $\mathcal{Q}_t^{\text{skill}} \leftarrow \{\, \text{op}(G, \mathcal{S}) \;:\; G \in \Sigma_{t-1},\ \mathcal{S} \in \mathcal{S}_t,\ \text{op} \in \mathcal{O} \,\}$, $|\mathcal{Q}_t^{\text{skill}}| = n_{\text{skill}}$
  $\big(\mathcal{P}_t, \mathcal{Q}_t^{\text{code}}\big) \leftarrow \text{SYNTHESIZE}\big(\text{PLAN}(\mathcal{D}_t)\big)$, $|\mathcal{Q}_t^{\text{code}}| = n_{\text{code}}$ ▷ *$\mathcal{D}_t$ from Eq. 5*
  $\mathcal{Q}_t \leftarrow \text{VALIDATE}\big(\mathcal{Q}_t^{\text{skill}} \cup \mathcal{Q}_t^{\text{code}}\big)$ ▷ *validation and fingerprint de-duplication*
  $s(G) \leftarrow \mathcal{E}(G; \mathcal{D}, \mathcal{T}, \mathcal{C})$ for every $G \in \mathcal{Q}_t$ ▷ *one training protocol per generation*
  $\bar{s}_t \leftarrow \text{mean}_{G \in \mathcal{Q}_t}\, s(G)$; $\Sigma_t \leftarrow \text{Top}_K\big\{\, G \in \mathcal{Q}_t \;:\; s(G) \geq \bar{s}_t - \delta \,\big\}$ ▷ *survivor selection*
  $\mathcal{L}_t \leftarrow \mathcal{L}_{t-1} \cup \big\{\, \mathcal{S} \in \mathcal{P}_t \;:\; G_{\mathcal{S}} \in \Sigma_t \,\big\}$ ▷ *sedimentation into Tier-2*
  $\mathcal{M}_t \leftarrow \mathcal{M}_{t-1} \cup \big\{\, \big(G, s(G), r_t, \ell(G), o(G)\big) : G \in \mathcal{Q}_t^{\text{skill}} \cup \mathcal{Q}_t^{\text{code}} \,\big\}$
  $G^\star \leftarrow \arg\max_{G \in \{G^\star\} \cup \Sigma_t} s(G)$
end for
return $G^\star,\ \mathcal{L}_T,\ \mathcal{M}_T$

Promotion bookkeeping. A candidate enters the survivor set Σ_t if its primary metric is at least the generation mean minus a tolerance $\delta \geq 0$ ($\delta = 0$ in the CTR, MTL, and MDL runs) and it ranks in the top-K after sorting. Only generated skills carried by a survivor are promoted. This rule retains modules used by selected candidates; it does not by itself measure each module's marginal contribution. A promoted card records the candidate metrics and status that triggered promotion, the architecture details, and the originating proposal and parent node, and the skill is registered in the index the compiler reads so that later generations retrieve it exactly like a Tier-1 skill.

C Tasks, Datasets, and Baselines

C.1 Tasks and Evaluation Objectives

We evaluate our framework on four representative recommendation tasks, covering single-objective prediction, multi-task learning, multi-domain learning, and joint quality–efficiency optimization.

- **Click-Through Rate (CTR) Prediction.** Given a user–item pair with its contextual features, CTR prediction estimates the probability that the user interacts with (e.g., clicks) the item, which is formulated as binary classification. The model is trained with binary cross-entropy (BCE) loss and evaluated by AUC and LogLoss, following common CTR benchmarks (Zhu et al., 2021).
- **Multi-Task Learning (MTL).** MTL jointly optimizes several related prediction objectives with a single model, where the tasks share part of the model's parameters so that knowledge transfers between them (Caruana, 1997; Ma et al., 2018a). The key challenge is to balance useful sharing against negative transfer between tasks. The overall loss is the (weighted) sum of per-task BCE losses, and we report the AUC of each task.
- **Multi-Domain Learning (MDL).** Also known as multi-scenario recommendation, MDL serves samples from several domains (scenarios) with one unified model. The domains share the same feature space and label, but their data distributions differ (Sheng et al., 2021). The model must capture both what the domains share and what is specific to each domain. Following Li et al. (2025b), we report AUC for each domain and overall.
- **Multi-Objective Optimization: AUC Co-optimized with MFU.** Beyond predictive quality, industrial recommenders must also use hardware efficiently. In this task, a searched architecture is judged jointly by (i) its predictive quality (AUC) and (ii) its Model FLOPs Utilization (MFU) (Chowdhery et al., 2023). MFU is the ratio of achieved training FLOP/s to the hardware's theoretical peak FLOP/s. The goal is to find architectures that improve AUC without lowering MFU (or that reach a better AUC–MFU Pareto front).

Table 9: Dataset statistics. For the six accuracy benchmarks, *# models* is the number of manually designed architectures in the comparison set. For QK-Video, it is the number of UniRank seed models used to compile the generative ranking skill library. Split sizes are numbers of interactions.

Task	# Models or seeds	Dataset	# Interactions		
			Train	Val	Test
CTR	7	MovieLens	700.15K	100.02K	200.04K
		Beauty	1.20M	171.26K	342.51K
		Books	13.76M	1.97M	3.93M
MTL	5	Census	199.52K	49.88K	49.88K
MDL	11	Amazon	658.83K	82.35K	82.35K
		MovieLens	800.17K	100.02K	100.02K
Multi-objective	15	QK-Video	394.65M	50.08M	48.58M

C.2 Datasets and Data Splits

Table 9 summarizes the statistics of all datasets used in the four tasks. The details are as follows.

- **CTR task.**
  - **MovieLens** (Harper & Konstan, 2015): a movie-rating dataset collected by GroupLens, containing user demographic features and movie attributes. We use the MovieLens-1M version.

  - **Amazon Beauty** and **Amazon Books** (McAuley et al., 2015; He & McAuley, 2016): two category subsets of the Amazon product review corpus, containing user ratings together with item metadata. Books is the largest CTR dataset in our study (over 19M interactions), so it tests whether searched architectures scale to large data. All three datasets are split into training, validation, and test sets with a ratio of 7:1:2.
- **MTL task.**
  - **Census-Income (KDD)** (cen, 2000): a UCI benchmark extracted from the 1994–1995 U.S. Current Population Surveys, with about 300K records and 40 demographic and employment attributes. It is widely used to evaluate multi-task recommenders (Ma et al., 2018a; Tang et al., 2020).
- **MDL task.** Both datasets follow the preprocessing protocol of the Scenario-Wise Rec benchmark (Li et al., 2025b), with an 8:1:1 split.
  - **Amazon** (Ni et al., 2019): three product categories (*Clothing*, *Beauty*, and *Health*) are treated as three domains.
  - **MovieLens** (Harper & Konstan, 2015): interactions are divided into three domains by user age group (*1–24*, *25–34*, and *35+*).
- **Multi-objective task.**
  - **QK-Video** (Yuan et al., 2022): the largest sub-dataset of the Tenrec benchmark, collected from Tencent's QQ Kandian video-recommendation platform. It contains about 493M user–video interactions. Its industrial scale makes it well suited to measuring training efficiency (MFU) together with predictive quality.

C.3 Baseline Models and Source Architectures

For each task, the manually designed architectures listed below serve as manual baselines (Table 9).

CTR baselines (7 models).

- **Wide&Deep** (Cheng et al., 2016): jointly trains a wide linear model (for memorization) and a deep MLP (for generalization).
- **DeepFM** (Guo et al., 2017): replaces the wide part of Wide&Deep with a factorization machine (FM). The FM and the deep network share one embedding layer, so no manual feature engineering is needed.
- **DCN** (Wang et al., 2017): adds a cross network that explicitly models bounded-degree feature interactions, in parallel with a deep network.
- **DCN-V2** (Wang et al., 2021): improves DCN with a more expressive cross layer, and supports both stacked and parallel structures.
- **EDCN** (Chen et al., 2021): strengthens parallel CTR models by sharing information between the explicit and implicit interaction branches through bridge and regulation modules.
- **AFM** (Xiao et al., 2017): extends FM with an attention network that learns a separate importance weight for each second-order feature interaction.
- **AutoInt** (Song et al., 2019): learns high-order feature interactions automatically with stacked multi-head self-attention layers.

MTL baselines (5 models).

- **Shared-Bottom** (Caruana, 1997): all tasks share the bottom layers, and each task has its own tower on top.
- **MMoE** (Ma et al., 2018a): replaces the shared bottom with a set of shared experts, combined by task-specific gating networks.
- **PLE** (Tang et al., 2020): separates task-specific experts from shared experts and extracts representations progressively across levels, which reduces negative transfer and the seesaw phenomenon.

- **ESMM** (Ma et al., 2018b): models post-view CTR and CTCVR over the entire sample space, which addresses sample selection bias and data sparsity in conversion-rate estimation.
- **AITM** (Xi et al., 2021): models the sequential dependence among multi-step conversions with an adaptive information-transfer module based on attention.

MDL baselines (11 models).

- **Shared-Bottom** (Caruana, 1997), **MMoE** (Ma et al., 2018a), and **PLE** (Tang et al., 2020): multi-task architectures adapted to MDL by treating each domain as a task.
- **STAR** (Sheng et al., 2021): a star-topology network whose shared centered parameters are combined element-wise with domain-specific parameters. It also uses partitioned normalization and an auxiliary network.
- **SAR-Net** (Shen et al., 2021): a scenario-aware ranking network that transfers users' cross-scenario interests through scenario-specific attention and mixes scenario-specific and shared experts.
- **AdaSparse** (Yang et al., 2022): learns adaptively sparse network structures for each domain with lightweight domain-aware neuron-level weighting factors.
- **AdaptDHM** (Li et al., 2022a): an adaptive distribution hierarchical model that clusters samples into distributions dynamically and routes each sample to the matching expert, without relying on predefined domain labels.
- **EPNet** and **PPNet** (Chang et al., 2023): the two components of PEPNet. EPNet personalizes embeddings with domain-side prior information, and PPNet personalizes the DNN parameters of each layer with user/item-side prior information via gate networks.
- **HAMUR** (Li et al., 2023): a hyper adapter framework. A shared hyper-network dynamically generates the parameters of domain-specific adapters that plug into backbone models.
- **M3oE** (Zhang et al., 2024b): a multi-domain, multi-task mixture-of-experts framework with shared, domain-specific, and task-specific experts, fused through a two-level structure whose fusion weights are found by AutoML.

Source architectures for generative ranking (15 models). For this setting, the seed-model set comprises the 15 generative ranking architectures listed by the UniRank benchmark (Li et al., 2026) [2]: OneTrans, RankMixer, Zenith, HyFormer, MixFormer, TokenMixer, HiFormer, INFNet, EST, LONGER, HeMix, UniMixer, TokenFormer, UltraHSTU, and SSR. These architectures are the source models from which the generative ranking skill library is compiled, and each is described below. RankMixer (Zhu et al., 2025) is the seed backbone used to initialize the reported co-evolution run; the other 14 seed models are source architectures rather than separately evaluated baselines in the AUC–MFU table.

- **OneTrans** (Zhang et al., 2026b): a single causal Transformer over one token stream of sequential and non-sequential features. Sequential tokens share parameters, non-sequential tokens have their own, and the KV cache is reused across requests.
- **RankMixer** (Zhu et al., 2025): a hardware-aware design that replaces self-attention with parameter-free multi-head token mixing and per-token FFNs. A Sparse-MoE variant scales it to 1B parameters.
- **Zenith** (Zhang et al., 2026a): scales a small set of high-dimensional Prime Tokens with Token Fusion and Token Boost, at little extra inference latency.
- **HyFormer** (Huang et al., 2026b): alternates Query Decoding, where global tokens decode long behaviour sequences, and Query Boosting, which mixes tokens across features.
- **MixFormer** (Huang et al., 2026a): scales dense interaction and sequence modelling together in one shared-parameter block with a query mixer, cross-attention and output fusion.
- **TokenMixer** (Jiang et al., 2026): scales token mixing to 15B parameters with mixing-and-reverting, inter-layer residuals, auxiliary losses and a sparse per-token MoE.

[2] https://github.com/salmon1802/UniRank

- **HiFormer** (Gui et al., 2023): heterogeneous self-attention with feature-specific projections, made servable by low-rank approximation and pruning.
- **INFNet** (Li et al., 2025a): per-group hub tokens gather context by cross-attention and broadcast it back to feature tokens, giving task-aware interaction at linear cost.
- **EST** (Liu et al., 2026): treats all raw inputs as one sequence, with lightweight cross-attention for heterogeneous features and content-sparse attention for behaviours.
- **LONGER** (Chai et al., 2025): models ultra-long user sequences with global tokens, token merging through inner Transformers, and hybrid attention.
- **HeMix** (Wang et al., 2026a): extracts user interests with dynamic and fixed queries, and replaces self-attention with a HeteroMixer block for multi-granularity feature interaction.
- **UniMixer** (Ha et al., 2026): makes rule-based token mixing a learnable operator that covers attention-, TokenMixer- and FM-style interaction.
- **TokenFormer** (Zhou et al., 2026): unifies multi-field and sequential tokens with bottom-full/top-sliding attention and multiplicative gating, which prevents representation collapse.
- **UltraHSTU** (Ding et al., 2026): co-designs the input sequence, sparse attention and topology of HSTU for better training and inference scaling.
- **SSR** (Yu et al., 2026): replaces dense connectivity with filter-then-fuse blocks, which apply sparse dimension-level filtering over parallel views and then fuse densely within each subspace.

D Implementation and Evaluation Protocol

The EvoSkillRec experiments fall into two groups. The first group runs accuracy-oriented architecture search on the CTR, multi-task (MTL), and multi-domain (MDL) benchmarks, where every candidate is **trained on a single GPU** and selected by validation AUC. The second group runs AUC–MFU co-evolution on the industrial-scale QK-Video corpus, where every candidate is **trained with multi-GPU** data parallelism and selected jointly on validation AUC and measured hardware efficiency. The two groups share the search procedure and the LLM pipeline, but differ in data scale, training recipe, and selection rule. All settings are fixed within a search run and are identical for every candidate in that run. The two automated baselines are described separately in Section D.5.

D.1 Training Protocols and Initial Architectures

Shared recipe The EvoSkillRec candidate models are implemented in PyTorch and trained from scratch with Adam at a learning rate of 10^{-3}, without learning-rate schedules or warmup. Each prediction target is trained with binary cross-entropy; for multi-task datasets the losses of all tasks are summed with equal weights. Table 10 compares the remaining settings of the two experiment groups.

CTR, MTL, and MDL benchmarks. Table 11 lists the dataset-specific settings. Training stops when the selection metric does not improve for the given number of epochs, and the best checkpoint is restored before test evaluation. The selection metric is validation AUC for CTR, the mean validation AUC of the income and marital-status tasks for Census-Income, and the global validation AUC over all domains for MDL (per-domain AUC is reported separately). CTR datasets are split chronologically into 70%/10%/20% for training, validation, and test. Census-Income uses the standard prepared splits with 32 categorical and 7 continuous features. The MDL datasets use a random 80%/10%/10% split with three domains.

The EvoSkillRec search starts from a standard model for each task (Table 12): DeepFM for CTR, Shared-Bottom for MTL, and SAR-Net for MDL on both MovieLens and Amazon. The CTR seed uses hidden widths $(256, 128, 64)$, ReLU, and dropout 0.2. The MTL seed uses a shared backbone of $(128, 64)$, task heads with one hidden layer of width 32, ReLU, and dropout 0.1. On both MDL datasets, the SAR-Net seed uses eight shared experts and two domain-specific experts of dimension 16, with no dropout. These values define only the starting point; evolution is free to change the structure and the architecture-specific parameters.

Table 10: Training settings shared by EVOSKILLREC candidates in each experiment group. MDL: multi-domain learning.

Setting	CTR / MTL / MDL benchmarks	AUC–MFU co-evolution
Optimizer	Adam, lr 10^{-3}	Adam, lr 10^{-3}
Weight decay	dataset-specific (Table 11)	0
Batch size	dataset-specific (Table 11)	8,192 per GPU
Epochs	dataset-specific, with early stopping	1
Gradient clipping	-	global norm 10
Precision	FP32	BF16 autocast
Embedding dimension	16 (CTR, MDL); 8 (MTL)	16
Model selection	best validation AUC	constrained Pareto selection on validation mean AUC and training MFU
Reported metrics	AUC, logloss	AUC, gAUC, logloss, training MFU
Random seed	2022	2026

Table 11: Dataset-specific training settings on the CTR, MTL, and MDL benchmarks. Epochs is the maximum number of epochs; patience is the early-stopping patience in epochs.

Task	Dataset	Batch size	Epochs	Weight decay	Patience
CTR	MovieLens-1M	1,024	5	3×10^{-4}	3
CTR	Amazon-Beauty	8,192	2	3×10^{-4}	2
CTR	Amazon-Books	8,192	2	3×10^{-4}	2
MTL	Census-Income	2,048	5	10^{-4}	3
MDL	MovieLens	4,096	1	10^{-5}	3
MDL	Amazon	4,096	1	10^{-5}	3

Table 12: Seed architecture for each task and dataset. Every run initializes the genome with an established architecture and evolves from there.

Task	**Dataset**	**Seed architecture**
CTR prediction	MovieLens	DeepFM
	Amazon-Beauty	DeepFM
	Amazon-Books	DeepFM
Multi-task learning	Census	Shared-Bottom
Multi-domain learning	Amazon (5-core)	SAR-Net
	MovieLens-1M	SAR-Net

AUC–MFU co-evolution. Table 13 summarizes the industrial-scale dataset used here. The user ID is the grouping key for gAUC. Every candidate is trained for one epoch and evaluated once at the end of the epoch. The search starts from RankMixer, which on QK-Video uses three RankMixer blocks with expansion ratio 2, four non-sequential tokens of dimension 128, and task towers of widths $(128, 64)$. User behavior sequences are truncated or padded to length 100 and processed by target attention with hidden widths $(256, 128)$, Dice activation, and no dropout or batch normalization. The resulting model has 141.7M parameters, of which 140.0M are embeddings and 1.66M are dense.

Table 13: Industrial-scale datasets used for AUC–MFU co-evolution, together with the RankMixer token dimension used on each dataset. Sample counts are taken from the parquet metadata.

Dataset	QK-Video
Train	394,647,318
Validation	50,079,027
Test	48,579,958
Tasks	4
Feature fields	10
Token dim	128

D.2 Search Budgets and Survivor Selection

Each EvoSkillRec generation allocates a fixed proposal budget of N candidate slots, split between skill-space and code-space proposals. Skill-space proposals apply one of four operators (add, replace, hybridize, specialize) to retrieved skill cards. Code-space proposals are macro-level modules produced by the LLM and wired into the parent genome either between two existing nodes or in place of one node, with input and output shapes preserved. Before training, each generated module is checked for syntax, importability, tensor-shape compatibility, and genome validity. Candidates whose architecture fingerprint matches a previously evaluated architecture, including those stored in the evolution memory, are skipped. A code-space module becomes a reusable skill only after its candidate survives selection and passes the portability check.

Budget controller. The split of N between the two spaces is adjusted online. If the best validation score has not improved by at least 10^{-3} for two consecutive generations, the controller moves budget from skill space to code space, up to the maximum code quota; any qualifying improvement resets it to the initial split. N itself never changes. When the LLM returns fewer valid proposals than its quota, skill-space candidates fill the remaining slots. Table 14 lists the per-run values.

Table 14: Evolution settings for each EvoSkillRec search run. B/B denotes Amazon-Beauty and Amazon-Books, which share one configuration. Quotas are given as numbers of candidates per generation.

	CTR		MTL	MDL		Co-evo.
Setting	ML-1M	B/B	Census	ML	Amazon	QK-Video
Generations	25	25	25	25	25	30
Candidate budget N	8	8	8	8	8	6
Survivors kept K	4	3	4	4	4	2
Initial skill / code quota	6/2	6/2	6/2	6/2	6/2	4/2
Code quota increment	2	1	2	1	1	2
Maximum code quota	6	6	6	6	6	4
Minimum skill quota	2	2	2	2	2	2
Stagnation patience	2	2	2	2	2	2

Survivor selection. On the CTR, MTL, and MDL benchmarks, survivor selection follows Algorithm 1. Let $s(G)$ denote the task's validation AUC metric and $\bar{s}_t = \mathrm{mean}_{G \in \mathcal{Q}_t} s(G)$ the generation mean. The survivor set Σ_t consists of the top-K candidates by $s(G)$ among those satisfying $s(G) \geq \bar{s}_t - \delta$, where δ is the tolerance in Algorithm 1. These survivors become the next generation's parents, and only generated skills carried by a survivor are promoted to Tier-2. If no candidate survives, the current parents are kept. In co-evolution, selection is made on a constrained Pareto

front of validation mean AUC and end-to-end training MFU. A candidate is eligible only if its validation AUC is within 0.001 of the incumbent best and its training MFU is at least 95% of the baseline's, with a stable MFU measurement (see Section D.4). Candidates without a valid hardware measurement are discarded. Test metrics are never used for selection in either setting.

D.3 LLM Settings for Planning and Code Generation

Code-space proposals are generated in two stages. A planner receives the parent genome, task and tensor dimensions, recent validation and hardware metrics, records of failed candidates, the current budget state, and the evolution history, and returns a set of architecture sketches. A synthesizer turns selected sketches into JSON proposals, each containing a PyTorch module, its tensor signature, the wiring location in the genome, and skill metadata. The LLM is called separately for each parent that receives code-space budget, so the total number of calls per generation depends on the controller state. Generated modules may not read or write files, spawn processes, or access the network. If the LLM fails to return a valid proposal after the allowed attempts, no template-based fallback is used and the slot is filled from skill space. See Table 15 for our settings.

Table 15: LLM settings. Sketches and proposals are requested per parent that receives code-space budget. Decoding parameters not set by our wrapper follow the provider defaults.

Setting	CTR / MTL / MDL	Co-evolution
Backend / model	Codex CLI, GPT-5.5	Codex CLI, GPT-5.5
Stages	planner $\rightarrow$ synthesizer	planner $\rightarrow$ synthesizer
Sketches requested	8 (ML-1M CTR), 12 (others)	8
Proposals requested	3 (B/B CTR), 2 (others)	2
Attempts per stage	1 (CTR), 2 (MTL, MDL)	2
Timeout per call	300 s	900 s
Output format	one JSON object per call	one JSON object per call
Temperature / top-p / max tokens	provider default	provider default

D.4 Hardware Setup and MFU Measurement

All experiments run on a single node with eight NVIDIA A800-SXM4 GPUs (80 GB each) and two Intel Xeon Platinum 8358 CPUs. On the CTR, MTL, and MDL benchmarks, each candidate is trained on one GPU, and up to eight candidates are evaluated in parallel. In co-evolution, each candidate is trained with distributed data parallelism on six GPUs.

Measuring model FLOPs utilization. MFU is the ratio between achieved training FLOP/s and the BF16 dense peak of the A800 (312 TFLOP/s per GPU). FLOPs per sample are counted on a single profiled training step after 30 warm-up steps. Step time is then measured over 150 steps split into three windows, and in each window the slowest rank determines the step time. The measurement is end-to-end: it includes data loading and all training-loop overhead. We accept a measurement only if the coefficient of variation of MFU across windows is at most 0.05. Table 16 reports the resulting values for the RankMixer baseline on QK-Video.

Table 16: Hardware measurements of the RankMixer baseline on QK-Video (6 GPUs, BF16, batch size 8,192 per GPU).

Quantity	Value
FLOPs per sample	58.67 M
Training MFU	1.43%
Step time	107.5 ms
Peak memory per GPU	8.12 GB

The baseline MFU is low, which is common for embedding-heavy ranking models whose dense part is small. This is the main reason we optimize MFU rather than FLOPs alone.

D.5 Implementation of the Automated Search Baselines

This section describes the NASRec weight-sharing search baseline and the OpenEvolve code-evolution baseline evaluated on the six CTR, MTL, and MDL benchmarks in Table 1. Our implementations adapt these methods to the recommendation tasks considered here. The task-specific data processing, model interfaces, search settings, and evaluation protocols used in these experiments are detailed below. Both use standalone PyTorch models and a shared candidate trainer, with the same preprocessing, data splits, random seed, optimizer, and early-stopping settings. Their dataset-specific training schedules match those in Table 11; method-specific search procedures and budgets are detailed below. These settings apply to the six accuracy benchmarks, separately from the QK-Video co-evolution experiment. Each baseline candidate uses one GPU and random seed 2022; independent workers can run across the eight-GPU node described in Section D.4.

Datasets, labels, and input features. Table 17 records the baseline input and label configurations. Both baselines use temporal 70%/10%/20% splits for CTR, the prepared Census-Income train/validation/test files, and seeded random 80%/10%/10% splits for MDL, following the dataset protocols in Section D.1. Amazon CTR uses the provided binary `label` column. For Amazon MDL, Beauty, Clothing, and Health form three domains, and ratings greater than 3 are positive. MovieLens CTR uses ratings of at least 4 as positives; MovieLens MDL uses ratings of at least 3. The latter groups age codes into $\{1, 18\}$, $\{25\}$, and $\{35, 45, 50, 56\}$, corresponding to the age groups in Section C.2. Categorical features are label-encoded with zero reserved for missing or padding entries. Continuous features use a separate dense-feature path; MovieLens MDL normalizes the age feature. Embedding dimensions are 16 for CTR and MDL and 8 for MTL.

Table 17: Input configurations of the two automated baselines. CTR splits are temporal, Census-Income uses prepared files, and MDL splits are seeded random splits. Sparse and dense refer to categorical and continuous input features, respectively.

Dataset / task	Samples	Target	Input features
MovieLens / CTR	1,000,209	rating ≥ 4	7 sparse
Books / CTR	19,651,355	provided binary label	2 sparse
Beauty / CTR	1,712,558	provided binary label	2 sparse
Census / MTL	299,285	income and marital status	32 sparse + 7 dense
Amazon / MDL	823,534	rating > 3	2 sparse; 3 domains
MovieLens / MDL	1,000,209	rating ≥ 3	6 sparse + 1 dense; 3 domains

Candidate training and validation-based selection. Each candidate implements a PyTorch `nn.Module` and is trained with Adam and binary cross-entropy with logits. Table 18 lists the shared baseline hyperparameters. The selection metric is validation AUC for CTR, the arithmetic mean of the two validation AUCs for Census-Income, and global validation AUC across domains for MDL. The corresponding test metrics are evaluated after candidate selection and do not guide search. For Census-Income, both baseline trainers average the two task losses with equal weights; the EvoSkillRec training description in Section D.1 uses their sum. Thus, the matching schedules do not imply identical loss normalization. For MDL, global and per-domain AUC are reported separately.

Table 18: Candidate-training settings shared by the NASRec and OpenEvolve baselines. Epochs is the maximum number of epochs, and patience is measured in epochs. These hyperparameters match Table 11; the Census loss reduction is described in the text.

Dataset / task	Epochs	Batch	Learning rate	Weight decay	Patience
MovieLens / CTR	5	1,024	10^{-3}	3×10^{-4}	3
Books / CTR	2	8,192	10^{-3}	3×10^{-4}	2
Beauty / CTR	2	8,192	10^{-3}	3×10^{-4}	2
Census / MTL	5	2,048	10^{-3}	10^{-4}	3
Amazon / MDL	1	4,096	10^{-3}	10^{-5}	3
MovieLens / MDL	1	4,096	10^{-3}	10^{-5}	3

Configured candidate budgets. Table 19 gives the configured budgets. Each baseline requests $B = 250$ standalone candidate-training slots on MovieLens CTR and MovieLens MDL, and $B = 200$ on the other four benchmarks. For OpenEvolve search, these correspond to 25 rounds with at most $n_b = 10$ or 8 proposals per round, respectively. EVOSKILLREC uses 25 rounds with eight candidate slots per round on all six benchmarks (Table 14). These are configured budgets, not counts of successfully trained models; invalid or failed proposals are not reported as successful evaluations. NASRec search additionally trains a weight-sharing supernet before standalone candidate training. Accordingly, candidate budgets do not measure total search computation.

Table 19: Configured candidate budgets on the six accuracy benchmarks. B is the standalone training budget for each automated baseline. $n_b = B/25$ is the OpenEvolve per-round proposal limit. EVOSKILLREC uses $25 \times 8 = 200$ candidate slots on each benchmark. Supernet training is additional to the NASRec budget B.

Dataset / task	n_b	Each baseline: B	EVOSKILLREC: slots
MovieLens / CTR	10	250	200
Books / CTR	8	200	200
Beauty / CTR	8	200	200
Census / MTL	8	200	200
Amazon / MDL	8	200	200
MovieLens / MDL	10	250	200

NASRec architecture search. The baseline samples a pool of $2B$ architectures from a fixed, task-specific search space and trains a weight-sharing supernet for three epochs. Architectures are sampled in a balanced manner during supernet training and ranked by inherited validation AUC. The highest-ranked distinct architectures are then instantiated as standalone models and trained independently under Table 18, up to budget B. This separates inherited-weight screening from the standalone evaluation used to select the reported model. The search spaces are:

- **CTR:** linear, FM, deep MLP, cross-network, AutoInt, and SENET-deep branches, with sum, gated, or concatenation-MLP fusion. The default configuration has embedding dimension 16, hidden widths $(256, 128, 64)$, dropout 0.2, three cross layers, and two attention heads.
- **MTL:** Shared-Bottom, MMoE, PLE, and task-specific towers, with variants using 4–8 experts and representation widths of 32–128.
- **MDL:** Shared-Bottom, STAR, MMoE, PLE, SAR-Net, and domain-specific towers, including 4–8-expert and wider STAR variants.

OpenEvolve code evolution. The proposal provider receives the current population, archive, recent failures, and island-specific parents, and generates ordinary PyTorch model modules. The entry point is `tools/ordinary_model_llm_provider.py`. Each proposal must define `build_model(context)`, use the feature dimensions provided by `context`, and return logits with the required task-specific shape. Candidate code cannot import EVOSKILLREC genome classes or access the filesystem or network. Before training, proposals undergo static safety, import/instantiation, forward-shape, and output-contract checks. Valid proposals use the same standalone trainer as the NASRec baseline. The search maintains a population of 32, an archive of 64, four islands, and two parents per proposal. One provider call per round returns at most n_b proposals. Invalid proposals and training failures are recorded; they are not assigned test results.

Shared settings and differences between methods. The two baselines share their data and candidate-training implementation, and the optimizer settings, epoch limits, batch sizes, and validation selection metrics align with those reported for EVOSKILLREC. The Census loss reduction differs as specified above. Search spaces, candidate representations, and proposal procedures remain method-specific; the comparison therefore evaluates the configured search systems under the reported protocols. Candidate records retain architecture and metric information, elapsed time, and the failure stage where applicable, to support inspection of individual evaluations.

Table 20: Method-specific search settings. B and n_b are defined in Table 19; they do not denote completed evaluation counts.

Method	Search and training budget	Search state
NASRec	3 supernet epochs; $2B$ screened architectures; B standalone training slots	Balanced sampling and inherited-AUC ranking
OpenEvolve	25 provider calls; at most n_b proposals per round; B configured training slots	Population 32; archive 64; 4 islands; 2 parents per proposal

E Supplementary Experimental Results

The supplementary results follow RQ1–RQ5 in the main text: predictive performance, evolution trajectories, contributions of the two search spaces, accuracy–efficiency trade-offs, and LLM backbones.

E.1 RQ1: Predictive Performance Across Recommendation Tasks

This appendix gives the full results behind Table 1 on all six datasets. Table 21 covers CTR prediction, Table 22 covers MTL on Census, and Tables 23 and 24 cover MDL on Amazon and MovieLens. Each table lists all manual models together with EvoSkillRec.

EvoSkillRec has the highest reported test AUC on all six datasets among the evaluated manual models and automated baselines. Relative to the strongest manual model, the observed AUC differences are 0.0035, 0.0121, and 0.0179 on the three CTR datasets, 0.0022 in mean AUC on Census, and 0.0021 and 0.0008 on the two MDL datasets. These are point estimates under the reported settings; the comparisons alone do not establish the reliability of small differences or isolate the contribution of each framework component.

Baseline behaviour per dataset (RQ1). The evaluated NASRec implementation does not surpass the strongest manual baseline on any task family. OpenEvolve exceeds the strongest manual baseline on the three CTR benchmarks but falls below it on MTL and MDL, with a difference of -0.0194 AUC on MDL-MovieLens. These outcomes describe the evaluated implementations and search settings. They do not by themselves distinguish the effects of search-space restrictions, initialization, architectural priors, or search efficiency.

Table 21: CTR prediction results on three datasets. Higher AUC (↑) and lower Logloss (↓) indicate better performance. The best manual-model result in each column is in **bold** and the second best is underlined.

	MovieLens		**Amazon Books**		**Amazon Beauty**	
Model	AUC ↑	Logloss ↓	AUC ↑	Logloss ↓	AUC ↑	Logloss ↓
Manual models						
Wide & Deep	**0.7487**	0.5921	**0.6587**	0.5073	0.6565	0.5052
DeepFM	0.7482	0.5921	0.6575	0.5024	0.6566	0.5107
DCN	0.7484	0.5895	0.6573	0.5048	0.6570	0.4839
DCN-V2	0.7479	0.5904	0.6571	**0.4953**	0.6612	0.5404
EDCN	<u>0.7487</u>	<u>0.5880</u>	0.6567	0.4960	<u>0.6639</u>	**0.4585**
AFM	0.7474	**0.5869**	0.6562	<u>0.4958</u>	0.6254	<u>0.4714</u>
AutoInt	0.7455	0.5932	<u>0.6579</u>	0.5029	**0.6666**	0.5629
EvoSkillRec	0.7522	0.5857	0.6708	0.4952	0.6845	0.4441

Table 22: Multi-task results on Census, reported as test AUC (↑). Income and Marital prediction are regarded as two tasks; Mean AUC is averaged over the two shared tasks. The best manual-model result in each column is in **bold** and the second best is underlined; tied entries are marked together.

Model	Income	Marital	Mean AUC
Manual models			
AITM	**0.9507**	0.9944	**0.9726**
PLE	0.9502	**0.9946**	0.9724
Shared-Bottom	0.9492	**0.9946**	0.9719
MMoE	0.9485	0.9945	0.9715
ESMM	0.8463	0.9912	0.9188
EvoSkillRec	0.9550	0.9946	0.9748

Table 23: Multi-domain results on Amazon. We report overall test AUC and Logloss (LL), as well as per-domain test AUC. Manual models are sorted by validation AUC, which is used for model selection. Among the manual models, the best result in each column is in **bold** and the second best is underlined.

	Overall		Per-domain AUC ↑		
Model	AUC ↑	LL ↓	D_0	D_1	D_2
Manual models					
SAR-Net	**0.7012**	**0.4632**	**0.6970**	**0.6768**	**0.7204**
M3oE	0.6905	0.4678	0.6859	0.6752	0.7026
AdaSparse	0.6876	0.4681	0.6800	0.6636	0.7080
MMoE	0.6756	0.4765	0.6669	0.6520	0.6973
PPNet	0.6771	0.4720	0.6765	0.6458	0.6991
HAMUR	0.6748	0.4778	0.6718	0.6484	0.6971
Shared-Bottom	0.6748	0.4732	0.6626	0.6498	0.6986
EPNet	0.6751	0.4772	0.6690	0.6559	0.6919
PLE	0.6719	0.4749	0.6647	0.6502	0.6969
STAR	0.6690	0.4749	0.6680	0.6397	0.6883
AdaptDHM	0.6305	0.5052	0.6331	0.6079	0.6504
EvoSkillRec	0.7033	0.4624	0.6963	0.6780	0.7240

Table 24: Multi-domain results on MovieLens. We report overall test AUC and Logloss (LL), as well as per-domain test AUC. Manual models are sorted by validation AUC, which is used for model selection. Among the manual models, the best result in each column is in **bold** and the second best is underlined.

	Overall		Per-domain AUC ↑		
Model	AUC ↑	LL ↓	D_0	D_1	D_2
Manual models					
AdaSparse	0.7920	0.5393	0.7974	**0.7961**	0.7856
SAR-Net	**0.7924**	**0.5384**	0.7970	0.7953	**0.7869**
EPNet	0.7917	0.5390	0.7972	0.7960	0.7866
M3oE	0.7912	0.5419	0.7945	0.7950	0.7852
Shared-Bottom	0.7918	0.5399	0.7964	0.7952	0.7841
MMoE	0.7905	0.5432	**0.7975**	0.7948	0.7835
HAMUR	0.7900	0.5423	0.7965	0.7953	0.7849
PLE	0.7890	0.5420	0.7961	0.7949	0.7851
PPNet	0.7889	0.5422	0.7967	0.7955	0.7791
STAR	0.7846	0.5470	0.7866	0.7891	0.7784
AdaptDHM	0.7805	0.5553	0.7802	0.7851	0.7744
EvoSkillRec	0.7932	0.5371	0.7971	0.7970	0.7871

E.2 RQ2: Architecture Evolution Across Generations

This appendix shows how EvoSkillRec evolves the genome, generation by generation, on all six datasets. Table 12 lists the seed model of each run. Tables 25–27 cover CTR prediction on MovieLens, Amazon-Beauty, and Amazon-Books, Table 28 covers MTL on Census, and Tables 29 and 30 cover MDL on Amazon and MovieLens-1M. Each table lists the best surviving candidate of every generation, the space it comes from, the mutation applied, and its validation and test scores. A generation is omitted from a table when none of its mutations produced a valid candidate, so the generation indices in these tables are not always consecutive.

The mutation operators are as follows. In the skill space, *specialize* adapts an existing Tier-1 path, *add* inserts a library skill as a new node, *replace* substitutes one module of the genome with an interface-compatible Tier-1 module while leaving the rest of the genome unchanged. For example, `sarnet_expert_mixer` denotes only the expert-mixing component derived from SAR-Net. The round-12 edit replaces the current expert mixer with this component, not the parent with the complete SAR-Net baseline. Architecture-name shorthand elsewhere in the tables also refers to a module replacement. The *hybridize* operator recombines two Tier-1 branches, and *reuse* draws a promoted Tier-2 skill. In the code space, *invent* writes a new module from scratch. Selection uses validation AUC only; test scores are reported but never used in the search. A ✓ marks a generation whose validation AUC beats the best so far, seed included.

The tables record validation improvements from both proposal spaces. Examples include code-space candidates at generations 3 and 9 on Amazon-Beauty and generations 0 and 4 on Amazon-Books, and the skill-space module replacement at generation 12 on Amazon MDL. Later skill-space rows also include reuse operations. These records describe candidate outcomes within each run; they do not isolate the marginal contribution of an individual module. Validation and test rankings need not coincide, and selection is based only on validation performance.

Table 25: Evolution trace for CTR prediction on MovieLens, seeded with DeepFM. Each row is the best surviving candidate of one generation. Operators and markers are defined in Appendix E.2.

			Val	Test		
Gen.	**Space**	**Mutation**	AUC	AUC	LL	**Imp.**
0	Skill	*specialize* `senet_fm`	0.7570	0.7513	0.5843	✓
1	Skill	*specialize* `senet_fm`	0.7570	0.7515	0.5845	
2	Skill	*add* `crossnet_v2`	0.7568	0.7519	0.5838	
3	Code	*invent* `autoint_senet_bridge_branch_r3`	0.7566	0.7518	0.5836	
4	Code	*invent* `contextual_subspace_moe_branch_r4`	0.7570	0.7520	0.5825	✓
5	Code	*invent* `bilinear_spline_interaction_branch_r5`	0.7571	0.7519	0.5828	✓
6	Skill	*reuse* `autoint_senet_bridge_branch_r3`	0.7574	0.7529	0.5817	✓
7	Skill	*reuse* `autoint_senet_bridge_branch_r3`	0.7566	0.7520	0.5831	
8	Code	*invent* `cin_cross_tower_replace_r9`	0.7574	0.7515	0.5862	
9	Code	*invent* `field_cumulant_moment_branch_r10`	0.7568	0.7516	0.5849	
10	Skill	*reuse* `field_relation_capsule_branch_r3`	0.7577	0.7512	0.5863	✓
11	Skill	*reuse* `autoint_senet_bridge_branch_r3`	0.7572	0.7513	0.5864	
12	Skill	*reuse* `field_relation_capsule_branch_r3`	0.7574	0.7523	0.5853	
13	Skill	*reuse* `oblivious_cross_tree_branch_r6`	0.7570	0.7514	0.5855	
14	Skill	*reuse* `autoint_senet_bridge_branch_r3`	0.7571	0.7515	0.5849	
15	Skill	*reuse* `autoint_senet_bridge_branch_r3`	0.7572	0.7515	0.5850	
16	Skill	*reuse* `oblivious_cross_tree_branch_r6`	0.7572	0.7519	0.5858	
17	Skill	*reuse* `oblivious_cross_tree_branch_r6`	0.7576	0.7517	0.5859	
18	Code	*invent* `contrastive_hyperedge_bank_replace_r20`	0.7576	0.7517	0.5859	
19	Skill	*reuse* `oblivious_cross_tree_branch_r6`	0.7569	0.7516	0.5856	
20	Skill	*reuse* `oblivious_cross_tree_branch_r6`	0.7575	0.7519	0.5857	
21	Code	*invent* `field_tuple_contrastive_memory_branch_r22`	0.7576	0.7517	0.5859	
22	Code	*invent* `neural_bdd_clause_lattice_replace_r23`	0.7579	0.7522	0.5857	✓
23	Skill	*reuse* `field_product_kernel_branch_r11`	0.7572	0.7511	0.5855	

Table 26: Evolution trace for CTR prediction on Amazon-Beauty, seeded with DeepFM. Operators and markers are defined in Appendix E.2.

			Val	Test		
Gen.	**Space**	**Mutation**	AUC	AUC	LL	**Imp.**
0	Skill	`relation_capsule_router_branch`	0.6943	0.6755	0.3893	✓
1	Skill	*reuse* `fourier_pair_kernel_branch`	0.6945	0.6747	0.3869	✓
2	Code	*invent* `tensor_train_field_interaction`	0.6966	0.6761	0.3915	✓
3	Code	*invent* `lowrank_cross_attention_tower`	0.7015	0.6809	0.4256	✓
4	Code	*invent* `fourier_volterra_kernel_branch`	0.7012	0.6807	0.4264	
5	Code	*invent* `contextual_branch_token_transformer_fusion`	0.7013	0.6803	0.4255	
6	Skill	*reuse* `contextual_branch_token_transformer_fusion`	0.7014	0.6806	0.4260	
7	Skill	*hybridize* `crossnet_afm`	0.7014	0.6806	0.4260	
9	Code	*invent* `contextual_field_reparameterizer`	0.7082	0.6843	0.4451	✓
10	Skill	*reuse* `contextual_branch_token_transformer_fusion`	0.7084	0.6847	0.4465	✓
11	Skill	*hybridize* `crossnet_afm`	0.7082	0.6843	0.4451	
14	Code	*invent* `sobol_tensor_mixture_cross_branch`	0.7087	0.6844	0.4423	✓
15	Skill	*reuse* `symmetric_bilinear_crossnet_branch`	0.7088	0.6842	0.4420	✓
16	Skill	*hybridize* `crossnet_afm`	0.7088	0.6842	0.4420	
18	Skill	*reuse* `adaptive_crossnet_expert_branch_1`	0.7090	0.6844	0.4463	✓
19	Skill	*hybridize* `crossnet_afm`	0.7090	0.6844	0.4463	
24	Skill	*reuse* `adaptive_crossnet_expert_branch_1`	0.7092	0.6845	0.4442	✓

Table 27: Evolution trace for CTR prediction on Amazon-Books, seeded with DeepFM. Operators and markers are defined in Appendix E.2.

			Val	Test		
Gen.	**Space**	**Mutation**	AUC	AUC	LL	**Imp.**
0	Code	*invent* `contextual_mixture_fusion`	0.6849	0.6634	0.5088	✓
1	Skill	*hybridize* `crossnet_afm`	0.6866	0.6579	0.5777	✓
2	Skill	*hybridize* `crossnet_afm`	0.6862	0.6578	0.5756	
3	Skill	*hybridize* `crossnet_afm`	0.6873	0.6593	0.5849	✓
4	Code	*invent* `contrastive_branch_consensus_fusion`	0.6940	0.6682	0.5599	✓
5	Skill	*reuse* `field_wise_monomial_lattice_branch`	0.6939	0.6663	0.5601	
6	Code	*invent* `branch_field_coattention_fusion`	0.6946	0.6699	0.5482	✓
7	Skill	*hybridize* `crossnet_afm`	0.6946	0.6699	0.5482	
10	Skill	*reuse* `complex_phase_correlation_branch`	0.6947	0.6696	0.5488	✓
11	Skill	*hybridize* `crossnet_afm`	0.6947	0.6696	0.5488	
12	Skill	*reuse* `hyperbolic_user_item_metric_branch`	0.6950	0.6693	0.5490	✓
13	Skill	*reuse* `hyperbolic_user_item_metric_branch`	0.6950	0.6693	0.5490	
14	Skill	*reuse* `kernel_memory_pair_compatibility_branch`	0.6958	0.6708	0.4952	✓
15	Skill	*hybridize* `crossnet_afm`	0.6958	0.6705	0.4955	
16	Skill	*hybridize* `crossnet_afm`	0.6955	0.6702	0.5428	
19	Skill	*reuse* `neural_outer_product_sketch_branch`	0.6958	0.6701	0.4967	
20	Skill	*reuse* `relation_capsule_router_branch`	0.6958	0.6698	0.5009	
21	Skill	*hybridize* `crossnet_afm`	0.6958	0.6700	0.5003	

Table 28: Evolution trace for multi-task learning on Census, seeded with Shared-Bottom. Selection uses validation mean AUC over the two tasks; test scores are reported only. Per-task validation scores are omitted for space, and values are given to five decimals because differences are small. Operators and markers are defined in Appendix E.2.

			Val	Test			
Gen.	**Space**	**Mutation**	Mean AUC	Mean AUC	Income AUC	Marital AUC	**Imp.**
0	Skill	`field_covariance_shrinkage`	0.97240	0.97473	0.95460	0.99486	✓
1	Skill	*specialize* `wider_shared_bottom`	0.97228	0.97425	0.95367	0.99483	
2	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97243	0.97477	0.95473	0.99482	✓
3	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97224	0.97455	0.95427	0.99483	
4	Code	*invent* `macro_flat_subspace_expert_router_v1`	0.97223	0.97438	0.95392	0.99485	
5	Code	*invent* `macro_flat_cross_stage_lattice_v1`	0.97231	0.97416	0.95357	0.99475	
6	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97225	0.97449	0.95410	0.99489	
7	Code	*invent* `macro_flat_multiscale_dilated_mixer_v1`	0.97244	0.97427	0.95366	0.99488	✓
8	Code	*invent* `macro_flat_block_krylov_resolvent_v1`	0.97252	0.97463	0.95437	0.99489	✓
9	Code	*invent* `macro_flat_virtual_task_view_fusion_v1`	0.97230	0.97452	0.95418	0.99486	
10	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97237	0.97446	0.95408	0.99485	
11	Code	*invent* `macro_flat_butterfly_expert_lattice_v1`	0.97237	0.97442	0.95402	0.99482	
12	Code	*invent* `macro_flat_hypercomplex_factor_mixer_v1`	0.97238	0.97439	0.95389	0.99488	
13	Code	*invent* `macro_flat_sparse_coordinate_graph_v1`	0.97231	0.97460	0.95433	0.99486	
14	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97244	0.97464	0.95441	0.99488	
15	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97239	0.97449	0.95412	0.99486	
16	Code	*invent* `macro_flat_wavelet_packet_cross_router_v1`	0.97241	0.97454	0.95422	0.99487	
17	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97265	0.97482	0.95500	0.99464	✓
18	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97252	0.97465	0.95448	0.99481	
19	Code	*invent* `macro_field_cumulant_conditioned_channel_mixer_v1`	0.97236	0.97398	0.95333	0.99463	
20	Code	*invent* `macro_flat_semiseparable_scan_bank_v1`	0.97250	0.97453	0.95415	0.99490	
21	Code	*invent* `macro_flat_oblique_forest_recomposer_v1`	0.97246	0.97447	0.95410	0.99484	
22	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97243	0.97464	0.95446	0.99482	
23	Skill	*reuse* `macro_field_covariance_shrinkage_v1`	0.97250	0.97418	0.95351	0.99486	
24	Code	*invent* `macro_flat_butterfly_permutation_fabric_v1`	0.97263	0.97448	0.95413	0.99483	

Table 29: Evolution trace for multi-domain learning on Amazon (5-core), seeded with SAR-Net. D_0–D_2 denote test AUC on each domain; per-domain validation scores are omitted for space. Scores are displayed to four decimals, and ✓ denotes an update of the best-so-far validation candidate. The round-12 operation replaces the current expert mixer with the SAR-Net expert-mixer component, preserving the rest of the evolved genome. Operators and markers are defined in Appendix E.2.

			Val	Test		Test AUC by domain			
Gen.	**Space**	**Mutation**	AUC	AUC	LL	D_0	D_1	D_2	**Imp.**
0	Code	*invent* `domain_aware_field_norm`	0.6786	0.6796	0.4713	0.6740	0.6528	0.7002	✓
1	Skill	*specialize* `ple_more_experts`	0.6794	0.6801	0.4718	0.6779	0.6464	0.7029	✓
2	Code	*invent* `domain_modulated_lowrank_field_rotation`	0.6794	0.6801	0.4718	0.6779	0.6464	0.7029	
3	Code	*invent* `domain_contrastive_field_residual`	0.6817	0.6823	0.4706	0.6825	0.6477	0.7043	✓
4	Code	*invent* `flat_domain_affine_norm`	0.6853	0.6846	0.4695	0.6820	0.6515	0.7074	✓
5	Code	*invent* `field_token_self_attention_adapter`	0.6856	0.6848	0.4692	0.6823	0.6520	0.7075	✓
6	Code	*invent* `flat_domain_prototype_residual`	0.6846	0.6839	0.4697	0.6812	0.6508	0.7068	
7	Code	*invent* `flat_sparse_mixture_adapter`	0.6869	0.6872	0.4680	0.6840	0.6545	0.7102	✓
8	Code	*invent* `rep_cross_domain_contrastive_residual`	0.6866	0.6881	0.4676	0.6859	0.6552	0.7106	
9	Skill	*replace* `m3oe`	0.6902	0.6894	0.4685	0.6874	0.6701	0.7041	✓
10	Code	*invent* `field_domain_token_affine_residual`	0.6901	0.6890	0.4687	0.6873	0.6693	0.7038	
11	Code	*invent* `flat_domain_moment_normalizer`	0.6927	0.6926	0.4672	0.6867	0.6749	0.7085	✓
12	Skill	*replace* `sarnet_expert_mixer`	0.7021	0.7012	0.4636	0.6946	0.6777	0.7233	✓
13	Code	*invent* `field_domain_channel_competition`	0.7022	0.7013	0.4635	0.6949	0.6773	0.7233	✓
14	Code	*invent* `flat_domain_hyper_adapter`	0.7026	0.7013	0.4630	0.6947	0.6775	0.7230	✓
15	Code	*invent* `field_domain_cross_stitch_channel_mixer`	0.7024	0.7020	0.4631	0.6945	0.6774	0.7237	
16	Code	*invent* `flat_domain_residual_mixture_calibrator`	0.7031	0.7018	0.4627	0.6945	0.6778	0.7231	✓
17	Code	*invent* `flat_domain_squeeze_excitation`	0.7033	0.7019	0.4627	0.6947	0.6779	0.7232	✓
18	Code	*invent* `flat_domain_sparse_delta_router`	0.7032	0.7027	0.4629	0.6956	0.6782	0.7240	
19	Code	*invent* `field_domain_cross_stitch_residual`	0.7032	0.7034	0.4619	0.6961	0.6783	0.7239	
20	Code	*invent* `field_domain_lowrank_hyper_token_adapter`	0.7031	0.7028	0.4624	0.6959	0.6780	0.7229	
21	Code	*invent* `field_domain_shared_private_splitter`	0.7034	0.7033	0.4624	0.6963	0.6780	0.7240	✓

Table 30: Evolution trace for multi-domain learning on MovieLens-1M, seeded with SAR-Net. D_0–D_2 denote test AUC on each domain; per-domain logloss is omitted for space. Operators and markers are defined in Appendix E.2.

			Val	Test		Test AUC by domain			
Gen.	**Space**	**Mutation**	AUC	AUC	LL	D_0	D_1	D_2	**Imp.**
0	Code	*invent* `flat_domain_feature_gate`	0.7947	0.7928	0.5379	0.7976	0.7953	0.7869	✓
1	Skill	*replace* `m3oe`	0.7930	0.7912	0.5419	0.7945	0.7950	0.7852	
2	Skill	*specialize* `ple_more_experts`	0.7941	0.7923	0.5384	0.7970	0.7957	0.7864	
3	Code	*invent* `domain_representation_slot_norm`	0.7939	0.7923	0.5387	0.7962	0.7958	0.7849	
4	Code	*invent* `domain_slot_contextual_lowrank_adapter`	0.7938	0.7919	0.5385	0.7961	0.7954	0.7860	
5	Code	*invent* `domain_representation_active_slot_cross_attention`	0.7942	0.7923	0.5395	0.7966	0.7957	0.7852	
7	Code	*invent* `flat_domain_basis_residual`	0.7942	0.7930	0.5384	0.7976	0.7961	0.7863	
8	Code	*invent* `flat_domain_cross_basis_adapter`	0.7951	0.7932	0.5375	0.7970	0.7953	0.7878	✓
9	Code	*invent* `flat_domain_blockwise_calibrator`	0.7949	0.7932	0.5378	0.7978	0.7959	0.7873	
10	Code	*invent* `field_domain_token_fourier_bias`	0.7946	0.7929	0.5374	0.7968	0.7958	0.7872	
11	Code	*invent* `field_domain_cross_token_squeeze`	0.7952	0.7927	0.5379	0.7969	0.7953	0.7870	✓
12	Code	*invent* `flat_domain_hadamard_expert_residual`	0.7950	0.7931	0.5374	0.7977	0.7959	0.7869	
14	Code	*invent* `flat_domain_residual_temperature_gate`	0.7950	0.7933	0.5372	0.7975	0.7961	0.7867	
15	Code	*invent* `flat_domain_covariance_residual_smoother`	0.7942	0.7922	0.5391	0.7981	0.7951	0.7857	
19	Skill	*replace* `adasparse`	0.7926	0.7910	0.5395	0.7972	0.7972	0.7853	
22	Skill	*specialize* `deeper_domain_tower`	0.7879	0.7881	0.5529	0.7956	0.7931	0.7810	
23	Skill	*replace* `mmoe`	0.7953	0.7932	0.5371	0.7971	0.7970	0.7871	✓
24	Skill	*replace* `shared_bottom`	0.7927	0.7917	0.5395	0.7971	0.7952	0.7840	

E.3 RQ3: Contributions of the Search Spaces and Skill Reuse

Table 2 includes a dual-space variant that retains both skill-space and code-space proposals but disables retrieval and recomposition of generated Tier-2 skills in later skill-space proposals. The variant can still compose curated skills and evaluate candidates containing newly synthesized modules. The full model has higher reported test AUC on all six benchmarks, with absolute gains ranging from 0.0004 to 0.0221 over this variant. The largest differences occur on MDL-Amazon (0.7033 vs. 0.6812, a gain of 0.0221) and CTR-Beauty (0.6845 vs. 0.6650, a gain of 0.0195). The variant without reuse also scores below the skill-space-only variant on both MDL benchmarks and below the code-space-only variant on all three CTR benchmarks. These results support the contribution of reusing generated skills under the reported settings; enabling both proposal spaces alone does not reproduce the full model's gains.

Table 3 reports descriptive statistics for the two proposal spaces. The trajectory tables record improvements after both code-space proposals and skill-space transitions, with their mix varying by dataset and stage. Positive-gain rates use evaluated candidates as the denominator, whereas survivor yields use all proposed candidates. Skill space has the larger observed $\Delta_{\max}$ on five of six benchmarks (by a factor of 2.4 on Beauty). Its survivor yield is higher on every CTR and MTL benchmark, but is 3.1% and 6.2% on the two MDL benchmarks, compared with 62.5% and 50.6% for code space. These aggregate rates do not isolate the effects of proposal quotas, duplicates, compatibility failures, or parent quality. Tier-2 promotions exceed 40 on CTR-MovieLens and both MDL benchmarks, but remain below 20 on Beauty, Books and Census. Promotion makes accepted code-space implementations available to subsequent skill-space proposals, so the proposal space of a later improvement need not identify the original source of every module it uses. These aggregate statistics describe proposal behavior within full-model runs; Table 2 provides the direct comparison with a dual-space variant without reuse.

E.4 RQ4: Accuracy–Efficiency Trade-offs on QK-Video

We evolve the RankMixer backbone to improve predictive quality (AUC) and hardware efficiency (MFU) jointly. The source architectures and skill library are described in Sections C.3 and A.4; the training and selection protocols are specified in Sections D.1 and D.2. Figure 5 compares the baseline with the Pareto-optimal models.

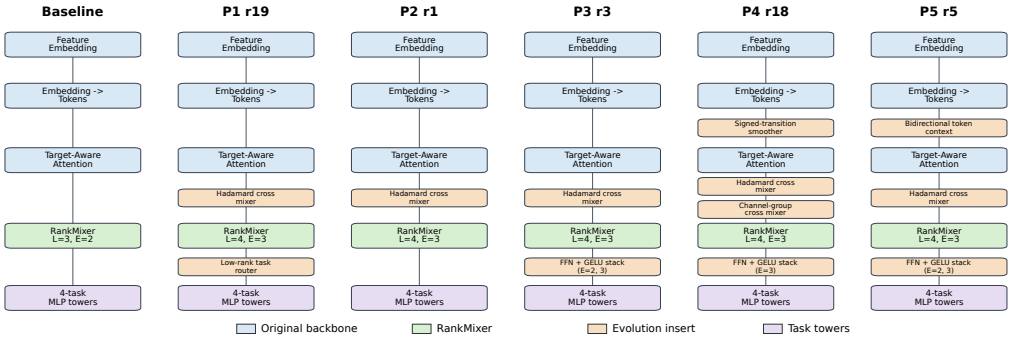

Figure 5: Architectural differences between the baseline (RankMixer) and the Pareto-optimal models on QK-Video.

AUC–MFU score for comparison after search. To summarise the trade-off between ranking quality and hardware utilisation, we give each evaluated architecture c the score

$$\mathrm{Score}(c) = \frac{1}{2}\,\frac{A(c) - A_{\min}}{\max(A_{\max} - A_{\min}, \epsilon)} + \frac{1}{2}\,\frac{M(c) - M_{\min}}{\max(M_{\max} - M_{\min}, \epsilon)}, \tag{6}$$

where $A(c)$ is the mean validation AUC across tasks and $M(c)$ is the measured training MFU. For $X \in \{A, M\}$, $X_{\min}$ and $X_{\max}$ are the minimum and maximum of X over $\mathcal{C}$, the set of architectures with valid measurements in the completed run, and $\epsilon = 10^{-12}$ avoids division by zero. All candidates share the same training budget and evaluation protocol. The bounds are fixed for the whole run and are not recomputed for each generation, and test metrics are not used. The score is only used for comparison after the run. Survivors are selected during evolution by the constrained Pareto criterion.

Accuracy–efficiency trade-offs and search progress. Pushing either objective alone costs the other. The best-AUC candidates in Table 31 are all retokenizers. The signed-bucket retokenizer (generation 14) gains $+6.7 \times 10^{-4}$ validation AUC but loses 8% MFU, because bucketing and hashing are memory-bound. The highest-MFU candidate (generation 7, $+12\%$ MFU) falls below the baseline in AUC. Both are discarded, and every survivor lies between 1.47% and 1.57% MFU. The Hadamard cross mixer appears in all five points of the Pareto front over all eligible candidates (Table 32). Paired with the low-rank task-basis router (P1), it is the only configuration that beats the baseline on validation AUC, test AUC and MFU. At the other end, P5 raises MFU by 9.8% over the baseline but gives up 7×10^{-4} validation AUC relative to P1.

The per-generation log (Table 33) shows three phases. Generations 0–5 make structural edits that raise MFU, such as per-token FFNs and Hadamard mixing. Generations 7–16 try sequence modules placed before DIN, and retokenizers. They reach the best AUC of the run (generation 14), but the retokenizers are discarded for their MFU cost. From generation 17 onward the search moves toward task routing, and the low-rank router (generation 19) improves both objectives. Of the 180 candidates, 26% survive and 23 skills are promoted to Tier-2. After generation 2 the budget shifts from skill space (4/2) to code space (mostly 2/4). This observed allocation does not establish that useful library recombinations have been exhausted. Failures concentrate late in the run (15 of 33 in generations 23–29). Rising failures and fewer promotions could serve as a stopping signal.

Table 31: Top candidates on the RankMixer backbone, sorted by validation AUC. **Gen.**: generation; **St.**: outcome (S = survivor, D = discarded); **Score**: equation 6. Validation gAUC and logloss are omitted. Best candidate value per column in **bold**.

			Val	Test				
Candidate	**Gen.**	**St.**	AUC	AUC	gAUC	Logloss	**MFU (%)**	**Score**
Baseline (RankMixer)	—	—	0.9295	0.9358	0.7833	0.0744	1.433	0.793
Signed-bucket retokenizer	14	D	**0.9301**	**0.9365**	0.7822	**0.0740**	1.317	0.868
Field-hash retokenizer	13	D	**0.9301**	0.9364	0.7815	0.0741	1.360	0.882
Markov bucket exchange	11	D	0.9300	0.9362	0.7833	0.0745	1.465	0.912
Field-covariance retokenizer	13	D	0.9300	0.9361	0.7831	0.0743	1.364	0.861
Low-rank task-basis router	19	S	0.9300	0.9360	0.7844	0.0744	1.477	0.911
Hadamard cross mixer	1	S	0.9300	0.9357	0.7825	0.0745	1.513	**0.924**
Directional-decay exchange	12	S	0.9299	0.9360	0.7834	0.0744	1.494	0.906
Lag-delta interest filter	7	S	0.9299	0.9359	**0.7848**	0.0744	1.472	0.888
Field-moment retokenizer	26	D	0.9299	0.9360	0.7813	0.0741	1.206	0.763
Orthogonal task-subspace fusion	22	S	0.9299	0.9360	0.7845	0.0744	1.488	0.891
Per-token FFN/GELU hybrid	25	D	0.9298	0.9358	0.7828	0.0743	1.535	0.907
Wider RankMixer block	21	S	0.9298	0.9362	0.7838	0.0743	1.506	0.893
Bidirectional context exchange	5	S	0.9293	0.9356	0.7821	0.0743	1.574	0.819
Per-token FFN/GELU hybrid	7	D	0.9290	0.9353	0.7828	0.0746	**1.605**	0.776

Table 32: Pareto front over all eligible candidates on validation AUC and MFU (RankMixer backbone), using the eligibility criteria in Section D.2. Δ: difference from the baseline in 10^{-4} AUC. Rows run from highest AUC to highest MFU. Best value per column in **bold**.

			Val		Test		
Model	**Gen.**	**Architectural change**	AUC	Δ	AUC	Δ	**MFU (%)**
Baseline (RankMixer)	—	Original backbone	0.929477	—	0.935804	—	1.433
P1	19	Hadamard cross mixer + low-rank task-basis router	**0.929986**	$\mathbf{+5.08}$	**0.935978**	$\mathbf{+1.74}$	1.477
P2	1	Hadamard cross mixer	0.929963	$+4.86$	0.935729	-0.75	1.513
P3	3	Hadamard cross mixer + per-token FFN/GELU	0.929632	$+1.54$	0.935701	-1.03	1.565
P4	18	Signed-transition smoother + Hadamard + channel-group mixer + FFN/GELU	0.929462	-0.16	0.935401	-4.04	1.567
P5	5	Bidirectional token context + Hadamard + FFN/GELU	0.929273	-2.04	0.935614	-1.90	**1.574**

Table 33: Per-generation log on the RankMixer backbone. **Change**: the generation's best-AUC candidate, with its outcome in **St. MFU**: the generation's highest MFU, with that candidate's outcome as superscript. **K/C**: skill-space/code-space split of the six candidates (Section 4.2.3). **S/D/F/U**: survived, discarded, failed, duplicate. **P**: skills promoted to Tier-2. Best values in **bold**.

				Val	Test			
Gen.	**Change**	**St.**	**K/C**	AUC	AUC	**MFU (%)**	**S/D/F/U**	**P**
0	token-mix + per-token FFN	S	4/2	0.929776	0.935908	1.536^{S}	2/2/2/0	0
1	Hadamard cross post-attention	S	4/2	0.929963	0.935729	1.519^{S}	2/4/0/0	1
2	token-mix + per-token FFN	S	4/2	0.929473	0.935687	1.534^{S}	2/1/2/1	0
3	per-token FFN + GELU	S	2/4	0.929631	0.935701	1.565^{S}	2/3/1/0	0
4	add FFN after RankMixer	D	2/4	0.929623	0.935608	1.564^{D}	1/4/1/0	1
5	triangular token context before DIN	S	2/4	0.929625	0.935573	1.574^{S}	2/4/0/0	2
6	consensus-disagreement fusion bridge	D	3/3	0.929662	0.935807	1.467^{D}	0/6/0/0	0
7	lag-delta interest filter before DIN	S	2/4	0.929876	0.935922	$\mathbf{1.605}^{\mathrm{D}}$	2/3/1/0	1
8	multi-scale delta exchange before DIN	S	3/3	0.929747	0.935558	1.510^{D}	2/4/0/0	2
9	spectral token filter before DIN	S	2/4	0.929713	0.935676	1.553^{D}	2/3/1/0	1
10	field-moment retokenizer	D	2/4	0.929736	0.936431	1.542^{D}	0/4/2/0	0
11	Markov bucket exchange before DIN	D	2/4	0.930017	0.936176	1.541^{D}	2/4/0/0	1
12	directional-decay exchange	S	2/4	0.929915	0.936034	1.545^{D}	2/3/1/0	2
13	field-hash retokenizer	D	2/4	0.930114	0.936423	1.585^{D}	2/3/1/0	0
14	signed-bucket retokenizer	D	2/4	**0.930143**	**0.936516**	1.483^{S}	2/3/1/0	1
15	compact anchor reservoir retokenizer	D	2/4	0.929599	0.935658	1.528^{D}	0/3/2/1	0
16	reuse multi-scale delta exchange	S	2/4	0.929785	0.936061	1.521^{D}	2/4/0/0	1
17	linear EMA contrast	S	2/4	0.929725	0.936000	1.540^{S}	2/2/1/1	1
18	token-transition Laplacian	D	2/4	0.929845	0.935729	1.567^{S}	2/4/0/0	2
19	low-rank task-basis router	S	2/4	0.929986	0.935977	1.577^{D}	2/3/1/0	2
20	reuse directional-decay exchange	D	2/4	0.929838	0.936072	1.484^{S}	1/5/0/0	1
21	wider RankMixer block	S	3/3	0.929823	0.936163	1.545^{S}	2/4/0/0	0
22	orthogonal task-subspace fusion	S	2/4	0.929855	0.936027	1.533^{D}	2/3/1/0	1
23	per-token FFN + GELU	S	2/4	0.929397	0.935656	1.531^{D}	1/1/3/1	0
24	rank-1 task-affinity fusion	S	2/4	0.929540	0.935561	1.504^{D}	1/2/3/0	1
25	per-token FFN + GELU	D	3/3	0.929823	0.935760	1.535^{D}	1/2/3/0	0
26	field-moment retokenizer	D	2/4	0.929863	0.935986	1.530^{D}	2/3/1/0	1
27	causal token spline	D	2/4	0.929593	0.935939	1.514^{D}	2/3/0/1	1
28	reuse task-axis contextual router	S	2/4	0.929606	0.936007	1.471^{S}	1/3/2/0	0
29	affine-free task-token router	D	2/4	0.929377	0.935614	1.533^{S}	1/1/3/1	0

E.5 RQ5: Effect of the LLM Backbone

We examine the effect of the LLM that proposes mutations by re-running the search in all three settings with four alternative backbones. The main experiments in RQ1–RQ4 use GPT-5.5; the four backbones in Table 34 are separate runs. Within each setting, the four runs start from the same architecture and use the same operators and selection rule.

Table 34: Best candidates from four LLM backbones on Amazon-Books (CTR), Census-Income (MTL), and MovieLens (MDL), initialized from DeepFM, Shared-Bottom, and SAR-Net, respectively. MTL AUC is averaged over the two tasks. Best Round marks the selected candidate's first appearance. Operation records the architectural change and its type: Code (code-space generation), Skill (skill-space reuse), or Template (template-level specialization).

Setting	Backbone	Val AUC	Test AUC	Best Round	Operation
CTR	GLM-5.2	0.6937	0.6684	24	Skill: add a reciprocal field-subspace router
	DeepSeek-V4-Pro	0.6952	0.6686	18	Code: replace DCN-V2 cross fusion with sample-adaptive bilinear gated fusion (23 branches)
	GPT-5.6 (SOL)	0.6968	0.6722	6	Code: add a dual-anchor cross-lattice interaction module
	Claude Opus 4.8	**0.6972**	**0.6738**	23	Skill: add a complex Volterra resonance factor branch
MTL	GLM-5.2	**0.9718**	**0.9739**	11	Code: add hyperbolic (Poincaré) projection on field embeddings
	DeepSeek-V4-Pro	0.9714	0.9734	24	Code: add dimension-wise 1D-conv gating on field embeddings (residual)
	GPT-5.6 (SOL)	0.9714	0.9736	24[†]	Code: add Low-rank Neural Flow
	Claude Opus 4.8	0.9714	0.9734	18	Skill: add a bidirectional orthogonal task-exchange module (shared/private subspaces)
MDL	GLM-5.2	**0.7957**	**0.7935**	19	Code: add a low-rank relation mixer
	DeepSeek-V4-Pro	0.7954	0.7924	2	Template: replace the domain module with PLE; shared experts $1 \to 2$, specific experts $2 \to 3$
	GPT-5.6 (SOL)	0.7954	0.7931	21	Code: add a domain slot memory over fields
	Claude Opus 4.8	0.7955	0.7934	5	Code: add a domain-conditioned soft field selection

[†]The round-1 candidate already reaches 0.97136 validation AUC; round 24 adds only 6.6×10^{-6}.

In these runs, all four backbones have higher test AUC than the strongest manual model on CTR (Amazon-Books; Wide & Deep, 0.6587) and MTL (Census-Income; AITM, 0.9726 mean AUC). On MDL (MovieLens), three have higher test AUC than the strongest manual model (SAR-Net, 0.7924), and DeepSeek-V4-Pro matches its reported value. Among the four alternatives, Claude Opus 4.8 has the highest observed CTR test AUC, while GLM-5.2 has the highest observed MTL and MDL test AUC. The selected MDL candidates use a low-rank relation mixer (GLM-5.2), a domain slot memory (GPT-5.6), and soft field selection (Claude Opus 4.8). These comparisons describe the selected candidates; they do not establish a stable ranking of backbones or attribute the differences to those modules alone.

The selected candidates appear at different rounds. GPT-5.6 finds its selected CTR candidate at round 6, and its round-1 MTL candidate is within 6.6×10^{-6} validation AUC of the selected round-24 candidate. The selected CTR candidates for GLM-5.2 and Claude Opus 4.8 appear at rounds 24 and 23, respectively. DeepSeek-V4-Pro's selected CTR candidate fuses 23 branches, whereas its selected MDL candidate specializes PLE by increasing the expert counts. These are observations from the reported runs and do not isolate the contribution of code generation versus reuse. On CTR, the selected candidates from these four alternative backbones have higher test LogLoss (0.533–0.554) than the manual models on Amazon-Books (0.495–0.507); this range does not include the separate GPT-5.5 main-experiment result of 0.4952. All results come from a single search per backbone and setting. Without independent repeats, the reliability of between-backbone differences is undetermined; we neither assign a statistical tie threshold nor infer a stable ranking.