COMPEVOLVE: COMPETITIVE EVOLUTION OF LLM AGENT HARNESS REGULARIZED BY BEHAVIORAL DIVERSITY AND GENERALIZATION JUDGING

Anonymous authors
Paper under double-blind review

ABSTRACT

Evolutionary search is a well-established paradigm for automatic design, and has recently been applied to evolve the *harness* around a large language model (LLM): its prompt, control flow, and multi-agent structure. Because it modifies the harness rather than the model weights, this paradigm extends even to closed-source models, at lower cost and with greater interpretability than fine-tuning. Within it, competition, keeping only the fittest of competing harnesses, is a natural mechanism for directed improvement. We find, however, that naive competition among LLM agents does not reliably surpass independent, non-competitive evolution: the population rapidly converges to a few near-identical architectures, and selection by training accuracy alone rewards designs that overfit the training problems. We present CompEvolve, which turns competitive evolution into a reliable optimizer through three components: information isolation prevents a capable LLM architect from exploiting the training signal, compelling it to improve genuinely transferable ability; selection regularized by behavioral diversity retains agents that are initially weak but complementary to the rest of the population, which are more likely to be recombined into stronger agents in later generations; and an LLM generalization judge evaluates the transferability of an agent's design at the architecture level, complementing training accuracy. CompEvolve consistently improves out-of-distribution generalization over independent evolution across multiple reasoning benchmarks and LLMs of varying capability.

1 INTRODUCTION

Whether a large language model (LLM) succeeds as an agent on complex tasks depends not only on the base model itself, but to a large extent on the *harness* built around it: its prompt, control flow, tool use, and multi-agent coordination structure. With the base model fixed, changing the harness alone can substantially improve reasoning performance, as shown by chain-of-thought prompting Wei et al. (2022), self-consistency Wang et al. (2023), and multi-agent debate and collaboration Du et al. (2024); Hong et al. (2024). This makes the harness itself a target worth optimizing.

Compared with fine-tuning, automatically optimizing the harness leaves model weights untouched, so it is far cheaper than training and applies even to closed-source models, produces human-readable traces and code that are easy to inspect and refine, and needs fewer training samples. Following this idea, a range of automatic harness-design methods have emerged: at the prompt level, OPRO Yang et al. (2024), DSPy Khattab et al. (2024), PromptBreeder Fernando et al. (2024), and EvoPrompt Guo et al. (2024); at the code and workflow level, ADAS Hu et al. (2025), AFlow Zhang et al. (2025b), AgentSquare Shang et al. (2025), and GPTSwarm Zhuge et al. (2024). More broadly, coupling LLMs with evolutionary search has produced expert-level programs and scientific discoveries in adjacent domains Romera-Paredes et al. (2024); Novikov et al. (2025).

Competition, which retains only the fittest among candidates according to specific rules, is a classical mechanism for directed improvement: evolutionary algorithms select individuals through tournaments Holland (1992); Goldberg & Deb (1991), competitive self-play has driven agents past human

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.

experts Silver et al. (2016; 2018); Vinyals et al. (2019), and population-based methods improve through survival of the fittest among individuals Jaderberg et al. (2017). Bringing competition into harness evolution is therefore a natural direction to explore.

We build a competitive evolutionary framework for agent harnesses, which yields a surprising observation: *naive competition does not reliably surpass independent, non-competitive evolution*. Survival of the fittest does not automatically translate into better out-of-distribution (OOD) generalization. One cause is the *premature loss of population diversity*: the population quickly homogenizes, and candidates that score modestly now but are behaviorally complementary are eliminated before later generations can improve them, depriving selection of the raw material it relies on. The other is *overfitting to the training set*: when the training data available for evolution is limited, selecting purely by training accuracy favors designs that fit the training problems but transfer poorly.

To address both, we present **CompEvolve**, which turns competitive evolution into a reliable optimizer. An architect LLM evolves the harness, which is executed by another (possibly closed-source) solver LLM. CompEvolve comprises three components. First, *information isolation*: the architect sees only a sanitized architectural brief, with no access to source code, original problems, or answers, so it cannot game the training signal (e.g., hard-coding answers to seen problems), a failure mode known as specification gaming or reward hacking Amodei et al. (2016); Skalse et al. (2022), and must instead improve genuinely transferable ability. Second, *selection-level behavioral-diversity regularization*: we select via soft sampling rather than a hard top-K, incorporating behavioral diversity into the fitness so that candidates with modest training accuracy but complementary behavior are retained into later generations. Third, a *generalization-oriented judge*: it assesses the transferability of a design at the architecture level, providing a signal complementary to training accuracy, screening out designs that overfit the training set and safeguarding OOD generalization.

Our main contributions are as follows:

- **Bringing competition into harness evolution and diagnosing its failure.** We evolve LLM agent harnesses through explicit competitive tournaments and find that naive competition does not reliably surpass independent evolution, which we trace to premature loss of population diversity and overfitting to the training set.
- **The CompEvolve framework.** We combine information isolation, selection-level behavioral diversity, and a generalization-oriented judge to respectively mitigate answer gaming (spec-gaming), architectural convergence, and overfitting in competitive evolution.
- **Systematic validation and analysis.** On three reasoning tasks (mathematics, code, and vision reasoning) and across solver LLMs of varying capability, CompEvolve consistently outperforms independent evolution and other baselines; a further analysis reveals how each component improves the evolutionary process.

2 Related Work

Automatic design of agent harnesses. A growing body of work automatically designs the prompts, workflows, or multi-agent structures around an LLM. At the prompt level, OPRO Yang et al. (2024) uses an LLM as an optimizer that proposes better prompts from past attempts; EvoPrompt Guo et al. (2024) couples LLMs with evolutionary algorithms; PromptBreeder Fernando et al. (2024) evolves task prompts together with self-referential mutation prompts; and DSPy Khattab et al. (2024) compiles declarative LM pipelines and optimizes their prompts. At the code and workflow level, ADAS Hu et al. (2025) lets a meta agent iteratively program new agents into a growing archive; AFlow Zhang et al. (2025b) searches code-represented workflows with Monte Carlo tree search; AgentSquare Shang et al. (2025) searches a modular design space guided by a performance predictor; GPTSwarm Zhuge et al. (2024) represents agent systems as optimizable graphs; and EvoFlow Zhang et al. (2025a), closest in spirit, evolves a population of workflows with a niching evolutionary algorithm that maintains diversity. A related line uses LLM-guided evolution to discover novel programs and algorithms Romera-Paredes et al. (2024); Novikov et al. (2025). Across these methods, however, the selection signal is an absolute fitness, a heuristic surrogate, a tree-search return, or niching pressure: harnesses are never in explicit head-to-head competition, diversity (where present) comes from niching rather than a failure-mode regularizer in selection, and OOD generalization is not an explicit objective. CompEvolve departs from all three.

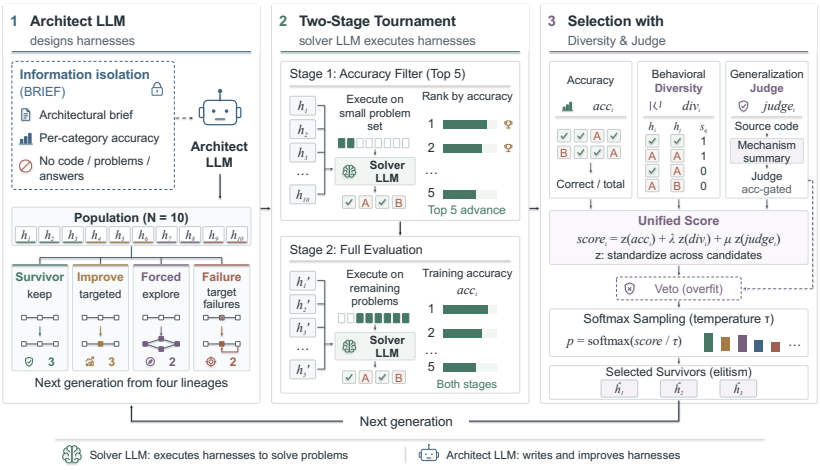

Figure 1: The architecture of CompEvolve. **(1)** The architect LLM, given a sanitized BRIEF of previous harnesses, writes the next population from four lineages (survivor, improve, forced, failure). **(2)** A two-stage tournament, executed by the solver LLM, filters the ten harnesses to five finalists by accuracy (Stage 1) and then evaluates them fully (Stage 2). **(3)** Survivors are selected by softmax sampling over a unified score that combines accuracy with behavioral diversity and a generalization judge, the latter also vetoing overfit designs.

Competition and diversity in evolutionary search. Competition is a foundational mechanism in evolutionary computation: tournament selection and related schemes retain the fittest among competing individuals Holland (1992); Goldberg & Deb (1991), and competitive coevolution pits individuals against evolving opponents Rosin & Belew (1997). Maintaining diversity is equally classical: MAP-Elites Mouret & Clune (2015) and the broader quality-diversity family Pugh et al. (2016) deliberately preserve behaviorally distinct solutions instead of optimizing a single objective. Beyond evolutionary algorithms, competitive self-play underlies systems that surpass human experts Silver et al. (2016; 2018); Vinyals et al. (2019), and population-based training improves a population through periodic survival-of-the-fittest replacement Jaderberg et al. (2017). Closer to our setting, the AI Co-Scientist Gottweis et al. (2026) improves scientific hypotheses with a tournament-and-Elo loop, though it evolves hypothesis content rather than reusable agent harnesses.

LLMs as evaluators. Using an LLM to judge another model's behavior is common: an LLM judge rates free-form responses in MT-Bench Zheng et al. (2023), and G-Eval Liu et al. (2023b) scores generation quality through structured prompts. Ours differs in both target and use: rather than only scoring an output, it also reads a harness's code and reasons counterfactually about how the design would fail on unseen problems, estimating transferability instead of correctness.

3 The CompEvolve Framework

3.1 Problem Setup and Evolutionary Skeleton

An agent harness is a program h implementing `h(x)->y`: given a problem x, it calls a solver LLM (at most B calls per problem) and returns an answer y. The search is driven by an architect LLM that reads the previous generation and edits harness code. What is evolved is the harness, not any model's weights; the solver may be different, possibly closed-source. CompEvolve evolves a population of such harnesses across generations, each cycling through the three stages of Figure 1: population generation from four lineages, a two-stage tournament, and survivor selection.

Population and lineages. We maintain a population of $N = 10$ harnesses and evolve it for G generations. Each generation's ten candidates come from four lineages: 3 **survivor** (the top harnesses from the previous generation, kept verbatim as elitism), 3 **improve** (a targeted refinement of a survivor), 2 **forced** (a harness deliberately different from the current population, for exploration), and

2 **failure** (a harness targeting failure modes exposed by the previous population). Of these, the survivors are copied verbatim; the improve, forced, and failure candidates are written by the architect. The mix balances exploitation (survivor, improve) with exploration (forced, failure).

Two-stage tournament. To compare candidates efficiently under a fixed budget, evaluation proceeds in two stages: in *Stage 1*, all $N = 10$ candidates answer a small problem set, are ranked *by accuracy alone*, and the top five advance; in *Stage 2*, the five finalists answer the remaining problems. The training accuracy acc_i is the overall correctness across both stages.

Unified selection score. Let acc_i, div_i, and judge_i denote the accuracy, behavioral diversity, and generalization-judge score of candidate i, and let $z(\cdot)$ standardize each term across the candidate set (z-score). The selection score is

$$\mathrm{score}_i = z(\mathrm{acc}_i) + \lambda\, z(\mathrm{div}_i) + \mu\, z(\mathrm{judge}_i). \tag{1}$$

We compute score_i for the Stage 2 finalists and draw survivors by *softmax sampling* with temperature τ, rather than a hard top-K:

$$p_i = \frac{\exp(\mathrm{score}_i/\tau)}{\sum_j \exp(\mathrm{score}_j/\tau)}. \tag{2}$$

Behavioral diversity and the judge act only at survivor selection, not in the Stage 1 filter. The three components map onto this loop: information isolation constrains the architect's input (Sec. 3.2), behavioral-diversity regularization supplies the div term and the soft sampling (Sec. 3.3), and the generalization judge supplies the judge term and vetoes clearly overfit designs (Sec. 3.4).

3.2 Information Isolation (BRIEF)

Competitive evolution asks the architect to improve harnesses from feedback. Shown source code, training problems, and answers, a capable LLM finds shortcuts, such as hard-coding answers to seen problems or branching on problem identifiers, that inflate training accuracy while collapsing OOD. We block this with *BRIEF information isolation*: the architect sees only a sanitized brief containing a natural-language description of each harness's architecture and its per-category accuracy, but no source code, problem text, or answers. Unable to locate specific training examples, the architect can only propose mechanism-level changes and is pushed toward genuinely transferable ability.

3.3 Selection-Level Behavioral Diversity

Under naive competition the population quickly homogenizes toward a few high-scoring paradigms, eliminating modestly scoring but behaviorally complementary designs before later generations can improve them. We mitigate this with a behavioral-diversity term in the selection score.

Failure-mode diversity. For two candidates i, j we define their similarity from their *correctness patterns* rather than from code or architecture. For a problem q,

$$s_q(i,j) = \begin{cases} 1, & \text{both correct;} \\ 1, & \text{both wrong with equivalent answers;} \\ 0, & \text{one correct, or both wrong differently,} \end{cases} \tag{3}$$

where answer equivalence uses the same normalization as majority voting. The mean similarity of i to the rest is $\bar{s}_i = \frac{1}{N-1}\sum_{j\neq i}\frac{1}{|Q|}\sum_{q\in Q} s_q(i,j)$, and its diversity score is $\mathrm{div}_i = 1 - \bar{s}_i$. Compared with architecture fingerprints, which collapse distinct designs into one class, and with raw answer agreement, failure-mode similarity captures whether the *errors* are complementary, which is what preserving candidates requires. This echoes the long-studied link between member-error diversity and ensemble accuracy Kuncheva & Whitaker (2003).

Diversity-regularized soft selection. We add div to the selection score with weight λ (Eq. 1) and sample survivors with softmax. A candidate below the hard accuracy cutoff but behaviorally complementary can thus still be selected and improved in the next generation.

Selection level, not architecture level. This diversity acts on *which* candidates are selected, that is, on the composition of the population, and does not change each candidate's own objective of maximizing accuracy. Forcing a single agent to use many distinct internal strategies would instead make a weaker solver spread its capacity and harm its OOD performance.

3.4 GENERALIZATION JUDGE

Selecting by training accuracy alone favors designs that do well on training problems but transfer poorly (overfitting). We add an *LLM generalization judge* that estimates a harness's transferability from its *design*: accuracy measures current ability, while the judge measures the soundness of the design's generalization basis. Among candidates of similar accuracy it biases selection toward stronger generalization bases and pulls the population away from designs that overfit.

Two-step scoring. To reduce the noise of scoring long code directly, the judge works in two steps. First, a *summarizer* LLM reads the harness source and produces a structured description of its mechanism (solving strategy, verification, aggregation, and so on). Second, a *judge* LLM reasons counterfactually over this description, asking where the design would fail on problems from a different distribution, and outputs a generalization score $\mathrm{judge}_i \in [0, 1]$. Candidates are scored jointly, providing a common frame of reference.

Rubric. The rubric separates a *generalization basis* (general reasoning such as chain-of-thought, answer verification, voting, and retry) from *domain specialization* (techniques tuned to specific problem types). Specialization earns credit when it does not harm generalization and is never penalized; only genuine overfitting, such as hard-coded answers or identifier-keyed logic, is.

Trust gate. To keep the judge from rewarding a degenerate, very low-accuracy design, we gate its influence by accuracy: the judge score is ignored when $\mathrm{acc}_i < 0.3$, fully trusted when $\mathrm{acc}_i \geq 0.6$, and linearly interpolated between.

Use: selection signal and overfit veto. The judge enters evolution in two complementary ways: as the $\mu\, z(\mathrm{judge})$ term of the selection score (after the trust gate), preferring a sounder generalization basis among candidates of comparable accuracy; and as a *veto floor* that removes candidates judged clearly overfit (hard-coded answers or identifier-keyed logic) from the next generation. It is computed in parallel with Stage 2 evaluation and adds no wall-clock time.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Tasks and data. We evaluate CompEvolve on three tasks. *Mathematics*: competition problems pooled from AIME and OlympiadBench He et al. (2024), 100 for training and the rest as an in-distribution test set (COMP); GSM8K Cobbe et al. (2021) and levels 3 and 5 of MATH Hendrycks et al. (2021) serve as OOD test sets. *Code*: 100 problems sampled from MBPP+ for training, the rest in-distribution, and HumanEval+ as OOD Austin et al. (2021); Chen et al. (2021); Liu et al. (2023a). *Visual reasoning*: 100 problems sampled from MathVerse Zhang et al. (2024) for training, the rest in-distribution, with MathVista Lu et al. (2024) and AI2D Kembhavi et al. (2016) as OOD.

Models and hyperparameters. Each task uses two solvers of different capability (math: qwen-flash / qwen-plus; code: qwen3-coder-flash / qwen3-coder-plus; visual reasoning: qwen3-vl-flash / qwen3-vl-plus). The architect is qwen3.7-max. We use $N{=}10$, $G{=}5$ generations, 5 candidates kept after Stage 1, at most $B{=}6$ solver calls per problem, $\lambda{=}0.5$, $\mu{=}0.5$, and $\tau{=}1.0$.

Metric. After evolution we take the top-3 surviving agents of the final generation (selected by training-side signals only) and report the median of their accuracies on each test set (median-of-top3). This aggregates over the small survivor set rather than reporting a single best agent, reducing sensitivity to the stochastic selection of any one individual.

Table 1: Mathematics: accuracy (%) as flash / plus. COMP is the in-distribution competition test set; GSM8K / MATH-L3 / MATH-L5 are OOD. Mean averages the four sets.

Category	Method	GSM8K	MATH-L3	MATH-L5	COMP	Mean
No evolution	Direct CoT	90.1 / 95.2	94.9 / 96.8	81.1 / 86.7	57.4 / 61.7	80.9 / 85.1
No evolution	Self-Consistency (k=3)	92.1 / 94.3	95.6 / 97.8	83.1 / 90.1	59.1 / 64.3	82.5 / 86.6
Auto design	OPRO	88.9 / **96.1**	86.4 / 80.6	70.2 / 53.8	47.8 / 43.5	73.3 / 68.5
Auto design	AFlow	84.5 / 92.1	94.3 / 91.8	85.6 / 74.2	65.2 / 55.7	82.4 / 78.5
Auto design	ADAS	93.1 / 93.8	97.8 / 98.2	87.8 / 91.2	70.4 / 83.5	87.3 / 91.7
Auto design	EvoFlow	92.1 / 94.2	96.1 / 97.1	87.8 / 90.2	74.8 / 79.1	87.7 / 90.2
Evolution	Independent evolution	**95.1** / 94.7	96.2 / 98.1	86.5 / 92.2	82.6 / 85.2	90.1 / 92.6
Evolution	Naive competition	94.1 / 94.3	96.7 / **98.9**	84.1 / 94.8	80.9 / **86.1**	89.0 / 93.5
Evolution	**CompEvolve**	92.9 / 96.0	**99.0 / 98.9**	**95.2 / 96.8**	**83.5 / 86.1**	**92.7 / 94.5**

Baselines. We compare against three groups. (i) No evolution: **Direct CoT** Wei et al. (2022) answers with a single chain-of-thought sample; **Self-Consistency** (k=3) Wang et al. (2023) majority-votes over three sampled chains. (ii) Automatic design: **OPRO** Yang et al. (2024), **AFlow** Zhang et al. (2025b), **ADAS** Hu et al. (2025), and **EvoFlow** Zhang et al. (2025a), each delivering its single best discovered agent. (iii) Evolution controls: **independent evolution** maintains 10 non-communicating evolutionary lines, and **naive competition** is competitive evolution without our three components.

4.2 MAIN RESULTS

Mathematics. Table 1 reports accuracy on the four math test sets. GSM8K and MATH-L3 are near saturation for both solvers (apart from the search-unstable OPRO and AFlow, all methods fall in a narrow 90–99% band), so method differences concentrate on MATH-L5 and COMP, which retain ample headroom. Overall, CompEvolve attains the best four-set mean on both solvers (92.7 / 94.5).

First, *evolution clearly beats non-evolutionary baselines, and the gap concentrates on hard sets*. On the two most discriminative sets, CompEvolve improves over Self-Consistency by $+24.4$ / $+21.8$ on COMP and $+12.1$ / $+6.7$ on MATH-L5; on the four-set mean the same comparison shrinks to $+10.2$ / $+7.9$, purely dilution by the saturated sets.

Second, *CompEvolve's net gain over the two equal-budget evolution controls lies almost entirely in OOD transfer rather than in-distribution fit*: it is close on the in-distribution COMP (83.5 / 86.1 vs. 82.6 / 85.2 and 80.9 / 86.1) yet clearly ahead on the OOD MATH-L5 (95.2 / 96.8 vs. 86.5 / 92.2 and 84.1 / 94.8). Naive competition or independent evolution already suffices to fit the training distribution. The value of the three components lies in turning comparable training fit into gains that survive distribution shift, precisely where automated harness design may fail.

Third, *competition by itself does not reliably produce gains; the gains depend on what selection sees and what it keeps*. Naive competition and independent evolution are nearly tied on the four-set mean (89.0 vs. 90.1 with the flash solver; 93.5 vs. 92.6 with plus), and neither is consistently better on COMP. This counter-intuitive result motivates the diagnosis in Section 4.4: the value of competitive evolution lies not in competition itself but in how the selection stage uses information and diversity (Sections 4.3–4.6).

The advantage over the other automatic-design methods likewise tracks the quality of their selection signals. ADAS and EvoFlow maintain archives or niches that delay structural convergence, so they come closest to the two evolution controls, but their fitness is still unprotected training accuracy and they end up near naive competition; AFlow and OPRO are weaker in turn. Inspecting their traces confirms why: OPRO optimizes a single CoT prompt with no structural operators, and its training accuracy fluctuates between 0.28 and 0.52 without a clear trend, so its selected prompt largely reflects validation noise on the 50-problem set; AFlow tends to favor revise prompts that re-solve from scratch rather than patch the reviewed solution, which can overwrite already-correct solutions on hard problems. Relying on small-validation-set training accuracy as the sole signal thus exposes these methods to noise and transfers less reliably to harder OOD reasoning.

Table 2: Code: accuracy (%) as flash / plus. MBPP+ is in-distribution and HumanEval+ is OOD.

Method	MBPP+	HumanEval+	Mean
Direct CoT	65.4/68.7	73.2/82.9	69.3/75.8
Self-Consistency	65.5/71.2	73.8/82.9	69.7/77.1
OPRO	64.9/68.5	71.3/79.9	68.1/74.2
AFlow	65.8/70.1	76.2/82.3	71.0/76.2
ADAS	66.5/70.9	78.7/83.5	72.6/77.2
EvoFlow	66.9/71.1	79.9/84.1	73.4/77.6
Independent evo.	67.6/69.8	81.1/84.8	74.4/77.3
Naive competition	66.6/**73.7**	76.8/82.9	71.7/78.3
+ isolation	67.3/68.0	**82.9**/86.0	75.1/77.0
CompEvolve	**67.9**/70.4	**82.9/87.8**	**75.4/79.1**

Table 3: Visual reasoning: accuracy (%) as flash/plus. MathVerse is in-distribution; MathVista and AI2D are OOD. Mean averages the three sets. GEN0 is the un-evolved initial multi-agent population.

Method	MathVerse	MathVista	AI2D	Mean
Direct CoT	64.8/72.0	48.2/54.4	68.2/68.8	60.4/65.1
Self-Consistency	66.0/73.0	50.5/56.0	69.5/70.0	62.0/66.3
OPRO	66.2/72.5	49.8/55.6	69.3/69.9	61.8/66.0
AFlow	78.5/80.5	57.5/59.0	79.0/81.0	71.7/73.5
ADAS	80.0/82.0	59.5/61.0	80.5/82.5	73.3/75.2
EvoFlow	81.0/82.8	60.5/62.0	81.5/83.0	74.3/75.9
GEN0 (initial pop.)	82.0/83.5	62.0/63.1	82.5/83.8	75.5/76.8
Independent evo.	83.2/84.8	62.6/63.8	83.6/85.6	76.5/78.1
Naive competition	84.5/85.4	65.0/65.7	84.8/86.2	78.1/79.1
CompEvolve	**87.6/88.0**	**68.5/69.0**	**87.4/88.6**	**81.2/81.9**

Code. Table 2 shows that code has a far larger cheating surface than math: without information isolation, competition tempts the architect to hard-code or overfit training cases, reaching inflated training accuracy while collapsing OOD (worst real case: 82% training vs. 12.2% on HumanEval+). The failure is masked in-distribution and exposed only OOD: with the plus solver, naive competition posts the highest MBPP+ score (73.7) but fails on HumanEval+. As in math, the automatic-design methods that can organize multi-agent structure (AFlow, ADAS, EvoFlow) beat the no-agent baseline. Adding information isolation immediately restores OOD performance ($+6.1$/$+3.1$), the largest single contribution of isolation across the three tasks, and the full method is best on both solvers.

Visual reasoning. Table 3 decomposes the gains into two additive sources. The first is *multi-agent structure itself*: from no agent to the un-evolved initial population GEN0, MathVista flash rises from 48.2 to 62.0 ($+13.8$), so wrapping the model in a multi-agent harness already provides a substantial gain. Consistently, automatic-design methods that can organize multi-agent workflows (AFlow, ADAS, EvoFlow) capture a large part of this structural benefit, whereas OPRO, which only rewrites a single prompt, stays near the no-agent level on all three tasks. The second is *evolution*: on top of GEN0, CompEvolve pushes MathVista flash further to 68.5 (another $+6.5$) and is best on both OOD sets, consistent with math and code.

4.3 ABLATIONS

We ablate the three components cumulatively, starting from each task's naive competition and adding information isolation, behavioral diversity, and the generalization judge in turn (Table 4). The division of labor mirrors each task's failure mode. (1) *Information isolation* adds little in math (flash $+0.4$, plus $+0.2$) but is the largest single step in code (HumanEval+ $+6.1$/$+3.1$), exactly where spec-gaming occurs. (2) *Behavioral diversity* is the largest step in math and visual reasoning (math flash $+2.7$ on MATH-L5 $84.7 \rightarrow 93.2$; vision $+1.4$/$+1.1$) but yields no stable gain in code, where correctness is nearly binary. (3) The *generalization judge* adds $+0.6$/$+0.3$ in math on the general-

Table 4: Cumulative ablation of the three components (flash / plus, %), adding information isolation, behavioral diversity, and the generalization judge in turn from each task's naive competition. Only discriminative sets are shown; Mean is the four-set average for math (as in Table 1) and the per-task average for code and vision.

(a) Mathematics

Configuration	MATH-L5	COMP	Mean	ΔMean
Naive competition	84.1/94.8	80.9/86.1	89.0/93.5	—
+ isolation	84.7/95.1	81.5/86.0	89.4/93.7	+0.4/+0.2
+ diversity	93.2/95.8	82.8/86.1	92.1/94.2	+2.7/+0.5
+ judge (full)	95.2/96.8	83.5/86.1	92.7/94.5	+0.6/+0.3

(b) Code

Configuration	MBPP+	HumanEval+	Mean	ΔMean
Naive comp.	66.6/73.7	76.8/82.9	71.7/78.3	—
+ isolation	67.3/68.0	82.9/86.0	75.1/77.0	+3.4/−1.3
+ diversity	67.4/68.2	82.9/86.0	75.2/77.1	+0.1/+0.1
+ judge (full)	67.9/70.4	82.9/87.8	75.4/79.1	+0.2/+2.0

(c) Visual reasoning

Configuration	MathVerse	MathVista	AI2D	Mean	ΔMean
Naive comp.	84.5/85.4	65.0/65.7	84.8/86.2	78.1/79.1	—
+ isolation	85.3/86.2	65.8/66.5	85.6/87.0	78.9/79.9	+0.8/+0.8
+ diversity	86.8/87.4	67.5/68.1	86.6/87.6	80.3/81.0	+1.4/+1.1
+ judge (full)	87.6/88.0	68.5/69.0	87.4/88.6	81.2/81.9	+0.9/+0.9

reasoning axis (MATH-L5 $93.2 \rightarrow 95.2$) and in code serves mainly as a cheat-veto. Each component contributes most where its target failure mode is most pronounced, improving OOD generalization rather than training fit.

4.4 WHY NAIVE COMPETITION FAILS

Naive competition does not convert competitive pressure into a consistent advantage: it ties independent evolution on math, is worse OOD on code (spec-gaming), and leads only slightly on vision while staying far below CompEvolve. We trace this to two quantifiable mechanisms.

Architectural collapse removes discriminability among candidates. Under competition the population homogenizes rapidly. For the math runs on both solvers, we recorded the per-generation architectural paradigm of the top-3 survivors (architect-reported structure labels, manually verified into four classes: parallel-arbitration, decompose-and-solve, generate-verify, self-critique). By the final generation, the survivors of naive competition on both solvers had almost entirely collapsed into the single "parallel solving + majority arbitration" paradigm.

Low complementarity caps the benefit of selection. Although each round holds multiple agents, their answers are highly correlated: an oracle that picks the best single agent per problem beats the best individual agent by only 1.26 percentage points on the math task, while majority voting is about 11–15 percentage *lower*, because when errors are highly correlated voting amplifies shared failures instead of cancelling them. When candidates are near-identical, selection has nothing to gain.

The two mechanisms compound: collapse leaves nothing to choose among, low complementarity nothing to gain. This is exactly what selection-level behavioral diversity targets.

4.5 WHY BEHAVIORAL DIVERSITY WORKS

Behavioral diversity helps not by raising current-generation scores but by retaining weak yet complementary candidates as raw material for later strong ones. Figure 2 tracks a representative qwen-flash math lineage: Candidate A ranks 8th of 10 in generation 2 (below the median 81.5), so hard top-5

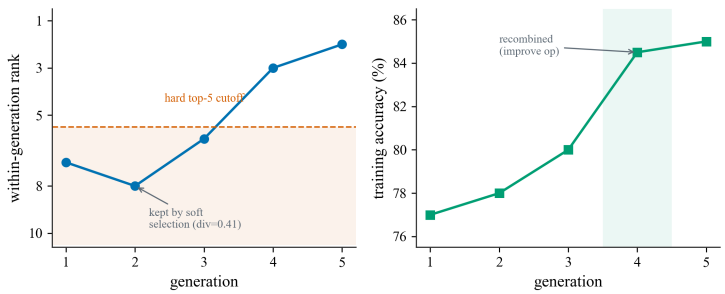

Figure 2: A lineage soft-retained by behavioral diversity. Left: within-generation training-accuracy rank; the shaded region marks elimination under hard top-5, which Candidate A enters at generation 2 (rank 8), but its generation-highest diversity score keeps it alive under soft selection. Right: the lineage's training accuracy; an *improve* recombination at generation 4 sends its descendant into the top-3.

would eliminate it, but it is the only design that decomposes and independently verifies each subproblem and its generation-highest diversity score ($\mathrm{div} = 0.41$) keeps it alive under soft selection; an *improve* recombination at generation 4 then lifts its descendant to the top-3 and the best MATH-L5 score (95.2). Under naive competition such candidates are hard-eliminated by generation 2–3 and MATH-L5 stalls at 84.1 (Table 1).

4.6 CAN AN LLM JUDGE GENERALIZATION?

Yes, on three counts. (i) Its score tracks general-reasoning transfer: on math the rank correlation with MATH-L5 is $\rho = +0.73$ (n=37, $p < 10^{-4}$) but with in-distribution COMP only $\rho = -0.16$ ($p = 0.34$), so an LLM can predict a design's transferability. (ii) As a fitness term it lifts general reasoning without hurting specialization: math $+0.6$ / $+0.3$ almost entirely from MATH-L5 (93.2→95.2), vision $+0.8$ to $+1.0$ on all three sets, code mainly through the veto. (iii) As a cheat veto, against labels defined by a train–OOD accuracy gap above 40 points, it fires on 68.3% of spec-gaming designs at a 2.4% false-alarm rate, catching for instance an MBPP "cheat sheet" design (82% train vs. 12.2% HumanEval+) that feeds training answers to the solver.

5 CONCLUSION

We brought explicit competition into the evolution of LLM agent harnesses and found, counter-intuitively, that naive competition does not reliably surpass independent evolution. Two mechanisms drive this: the population homogenizes in structure (architectural collapse), and selecting by training accuracy alone rewards overfitting. The bottleneck thus lies not in competitive pressure but in what information selection receives and which candidates it keeps.

CompEvolve addresses this at the selection level with three components: information isolation cuts the channel through which the architect could hack the training signal; behavioral diversity keeps weak but complementary candidates alive until recombination realizes their value; and a generalization judge supplies a design-level signal orthogonal to training accuracy and vetoes clearly overfit designs.

Across mathematics, code, and visual reasoning, and across solvers of different capability, CompEvolve matches naive competition in distribution but is decisively ahead out of distribution, showing that in competition-based harness evolution the design of the selection signal is what matters.

The main limitation is that all three components act on evolution's selection side: they change what the population keeps, not what the architect can propose, so population innovation is ultimately capped by the architect's generative capability, and our evaluation is confined to automatically verifiable tasks. Extending optimization to the generation side and validating on open-ended tasks are important future directions.

AI USE STATEMENT

Generative AI tools were used to assist with code implementation and to improve the language and readability of the manuscript. Other than this implementation and writing assistance, generative AI tools were not used to generate synthetic data; develop theoretical or conceptual frameworks, hypotheses, mathematical statements, or proofs; design or critique methods or experiments; translate text; clean or restructure data; perform qualitative or thematic analysis; or interpret the results. The authors reviewed and revised all AI-assisted code and text and take full responsibility for the final content of this work, including all code, text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety, 2016.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, et al. Evaluating large language models trained on code, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems, 2021.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 11733–11763. PMLR, 2024.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13481–13544. PMLR, 2024.

David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, volume 1, pp. 69–93. Morgan Kaufmann, 1991.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with co-scientist. *Nature*, 655:487–496, 2026. doi: 10.1038/s41586-026-10644-y.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun.

Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (ACL 2024)*, pp. 3828–3850. Association for Computational Linguistics, 2024.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems 34 (NeurIPS 2021) Datasets and Benchmarks Track*, 2021.

John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press, 1992. Originally published 1975, University of Michigan Press.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks, 2017.

Aniruddha Kembhavi, Mike Salvato, Eric Kolve, Minjoon Seo, Hannaneh Hajishirzi, and Ali Farhadi. A diagram is worth a dozen images. In *Computer Vision – ECCV 2016*. Springer, 2016.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Ludmila I. Kuncheva and Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003. doi: 10.1023/A:1022859003006.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*, 2023a.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using GPT-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP 2023)*, pp. 2511–2522. Association for Computational Linguistics, 2023b.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites, 2015.

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

Justin K. Pugh, Lisa B. Soros, and Kenneth O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016. doi: 10.3389/frobt.2016.00040.

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(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Christopher D. Rosin and Richard K. Belew. New methods for competitive coevolution. *Evolutionary Computation*, 5(1):1–29, 1997. doi: 10.1162/evco.1997.5.1.1.

Yu Shang, Yu Li, Keyu Zhao, Likai Ma, Jiahe Liu, Fengli Xu, and Yong Li. Agentsquare: Automatic LLM agent search in modular design space. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward gaming. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, pp. 9460–9471, 2022.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35 (NeurIPS 2022)*, 2022.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024.

Guibin Zhang, Kaijie Chen, Guancheng Wan, Heng Chang, Hong Cheng, Kun Wang, Shuyue Hu, and Lei Bai. Evoflow: Evolving diverse agentic workflows on the fly, 2025a.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. Aflow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025b.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, Peng Gao, and Hongsheng Li. Mathverse: Does your multimodal LLM truly see the diagrams in visual math problems? In *Computer Vision – ECCV 2024*. Springer, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023) Datasets and Benchmarks Track*, 2023.

Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62743–62767. PMLR, 2024.