EXPEVOLVE: STRUCTURING AND REUSING EXPERIENCE FOR EVOLUTIONARY OPEN-ENDED DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Evolutionary coding agents such as AlphaEvolve are one of the notable successes in using Large Language Models (LLMs) to build AI Scientists. These agents tackle open-ended scientific problems by iteratively improving and evolving programs, leveraging the prior knowledge and reasoning capabilities of LLMs. Despite their success, mutations over programs are mostly low-level upon surface heuristics, without a persistent unit of control. In contrast, science compounds through ideas. In this work, we propose ExpEvolve, which injects idea-level extraction and control into evolution: it discovers and leverages the high-level factors from programs that explain scoring differences among different programs. We characterize theoretical conditions on when discovery with idea factors improves sample efficiency. We also build a practical framework to show that the extracted high-level idea factors are helpful for guiding exploitation from experience and exploration when discovery comes to a plateau. Across 11 open-ended mathematical tasks, `ExpEvolve` significantly improves the final performance and sample efficiency up to $5.0\times$. The improvements also transfer to 4 other LLMs and to 6 machine learning tasks from MLS-Bench. Analysis shows that `ExpEvolve` can discover idea-level factors to provide better exploration and exploitation for evolutionary discovery.

1 INTRODUCTION

Large language models (LLMs) are now widely used to build agents that operate on increasingly long-horizon tasks (Plaat et al., 2025). One of the most ambitious applications is AI scientist agents (Zheng et al., 2025a), in which LLMs automate the scientific discovery process, including literature surveys (Asai et al., 2026), hypothesis generation (Wang et al., 2024; Yang et al., 2024), data-driven analysis (Majumder et al., 2025), experiment design (Li et al., 2025b), or even the whole discovery workflow (Lu et al., 2024; Yamada et al., 2025; Gottweis et al., 2026; Mitchener et al., 2025). LLM-driven discovery has shown strong results in a variety of domains, such as mathematics (OpenAI, 2026c;b), de novo protein design (Anthropic, 2026), algorithm discovery (Novikov et al., 2025; Lange et al., 2026), and the research that develops LLMs themselves, an early form of recursive self-improvement (RSI) (OpenAI, 2026a; Favaro & Clark, 2026; Lyu et al., 2026a; Karpathy, 2026). At the heart of the discovery loop is the *iteration over scientific hypotheses*, where scientific ideas compound throughout the evaluation of promising hypotheses (Wallace et al., 2026). Therefore, it is essential to *reuse the discovery experience* to generate high-quality hypotheses, e.g., avoiding getting stuck in repetitive mistakes or suboptimal solutions and encoding promising exploratory proposals.

In this work, we study experience reuse in open-ended discovery with evolutionary coding agents, where hypotheses are represented as *solution programs* to the underlying scientific problem, and experiences are accumulated as *evaluated programs and their scores* given by an explicit evaluator (Novikov et al., 2025). Typically, existing evolutionary coding agents such as `ShinkaEvolve` directly incorporate successful programs or rely on LLMs to summarize reusable insights and incorporate them as memory for future mutations with LLMs (Lange et al., 2026; Gideoni et al., 2026; Karpathy, 2026). However, program mutation contains an open-ended search space including low-level details such as hyperparameters and high-level insights such as optimization strategies. LLM-driven guidance is prone to focusing on low-level details (Karpathy, 2026; Pelleriti et al., 2026), which may not bring genuine new scientific discoveries. In contrast, human scientists tend to recognize the high-level contrasts among programs, accumulate and exploit scientific ideas from experience (Kuhn & Hawkins, 1963; Douven, 2025). The gap raises a challenging research question:

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

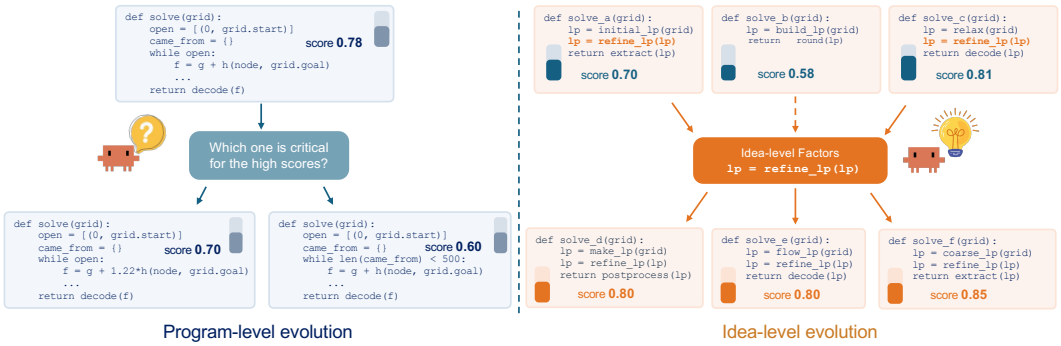


Figure 1: Comparison between experience reuse at program-level and idea-level. All programs solve the same grid task. **Left:** Experience reuse at program-level is prone to low-level edits. Program-level evolution derives new programs from a high-scoring parent, such as changing the heuristic weight or bounding the search loop. Both edited programs score lower. **Right:** Experience reuse at idea-level compares structurally different programs through a shared procedure-level factor, `lp = refine_lp(lp)`. Programs that contain the factor score higher than the program that omits it. This contrast singles out the factor, which is then inserted into new programs of different structures.

How can we reuse experience for evolutionary discovery like human scientists?

To tackle the question, we formulate evolutionary discovery as a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998) (Sec. 3). In each turn, the agent needs to propose meaningful solutions to uncover the underlying causal mechanism for solving the scientific problem, i.e., maximizing the evaluation score (Wallace, 1981; Glymour). Under the formulation, we introduce the notion of *ideas* as high-level mechanistic properties of the programs that determine the evaluation scores, e.g., the construction family or the optimization strategy (Def. 3.2). Without uncovering the ideas, the agent has to treat programs as unrelated and cannot tell whether a revision changes an idea or only a low-level detail, so discovery degenerates into exhaustive search (Thm. 3.1). Meanwhile, the identification of ideas also requires *sufficient exploration* to contrast programs with and without each idea (Thm. 3.3). Hence, an effective discovery agent needs to extract ideas from experience, and keep exploring programs where the ideas vary.

Motivated by the analysis, we propose `ExpEvolve`, which recognizes, structures, and leverages the high-level idea factors from discovery experience (Fig. 1). Specifically, `ExpEvolve` realizes the discovery of ideas via identifying high-level factors that help explain the program scoring differences. Built upon `COAT` (Liu et al., 2025), the discovered idea factors are expected to predict which program performs better given any pair of programs in the experience. Then, the admitted idea factors will be incorporated as an action of a multi-armed bandit that controls the exploration and exploitation of different ideas during discovery. To enable sufficient exploration, `ExpEvolve` also allocates slots for random exploration (Gideoni et al., 2026), and idea-guided exploration when the evolution comes to a plateau. The idea-guided exploration enforces LLMs to explore diverse ideas other than those widely used in top-scoring programs. The idea-based control in `ExpEvolve` allows efficient exploitation of promising ideas while continuing to explore programs with varied ideas.

We evaluate `ExpEvolve` on 11 open-ended mathematical discovery tasks with a budget of 200 evaluated programs, using `SCS` as the backbone and the main baseline (Sec. 5). Under two configurations that split the budget into 20 generations of 10 programs and 10 generations of 20 programs, `ExpEvolve` improves the final performance over `SCS` and `ShinkaEvolve` in most settings with smaller budgets up to 5.0×. Further analyses show that the discovered idea factors order the archived programs better than their code length on all 11 tasks. We further integrate `ExpEvolve` into `OpenRSI` (Yang et al., 2026b) on six tasks from MLS-Bench (Lyu et al., 2026a), where it improves the mean final best-so-far score of `OpenRSI` on all six tasks.

2 Related Work

In this section, we discuss the closest related works and defer a more detailed one in Appendix B.

AI Scientist agents. With the significant advancement in LLM capacity and the development of agentic systems, building LLM-based AI scientists has gained increasing attention (Zhou et al.,

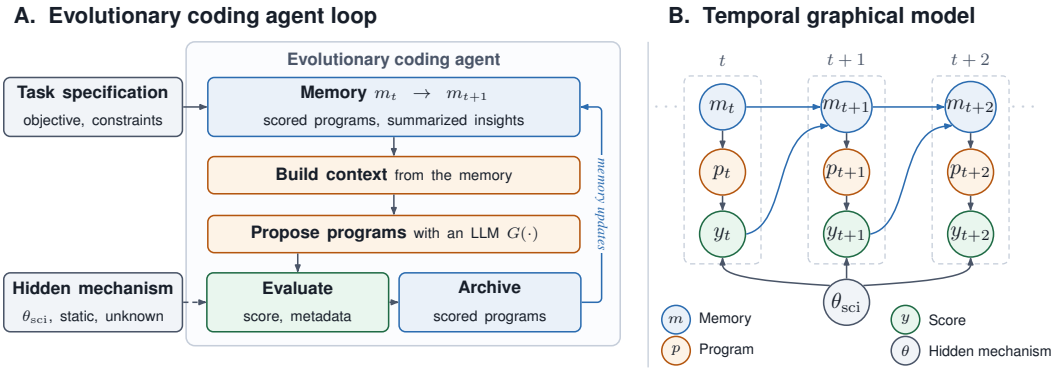


Figure 2: Evolutionary discovery as a POMDP. **(A)** In each iteration, an evolutionary coding agent builds the proposal context from its memory, e.g., parent and inspiration programs or summarized insights, and generates candidate programs with an LLM G. The evaluator scores every program, whose outcome depends on the hidden mechanism θ_{sci}, and the scored programs enter the archive and update the memory. **(B)** The corresponding temporal graphical model. The memory m_t guides the next program p_t, and the score y_t depends on both p_t and the static but unknown mechanism θ_{sci}, which in turn updates m_{t+1}.

2025). One research line automates part or the whole of the pipelines in scientific activities through agentic workflows (Lu et al., 2026; Yamada et al., 2025; Gottweis et al., 2026; Ghareeb et al., 2026; Mitchener et al., 2025; Feng et al., 2026b; Lyu et al., 2026b). Evolutionary coding agents are one of the most representative AI scientist agents that integrate LLMs into evolutionary search over programs scored by an explicit evaluator (Romera-Paredes et al., 2024; Novikov et al., 2025). AI scientist agents, including evolutionary coding agents, have made impact in a variety of domains, such as mathematics (OpenAI, 2026b), materials science (Ghafarollahi & Buehler, 2025), biology (Anthropic, 2026), and the development of LLMs themselves, e.g., accelerating LLM training and research toward recursive self-improvement (Karpathy, 2026; Lyu et al., 2026a; OpenAI, 2026a; Favaro & Clark, 2026; Zhang et al., 2026). As generation scales up, validating the generated hypotheses becomes the main cost (Wallace et al., 2026), and ideas that look promising often degrade once executed (Si et al., 2026). The expanding paradigm of LLM-driven scientific discovery makes the reuse of experience *increasingly more essential*, which is central to our work.

Experience reuse in evolutionary coding agents. Existing evolutionary coding agents typically reuse experience via conditioning on evaluated programs (Novikov et al., 2025; Gideoni et al., 2026) or summarizing textual insights from programs into memory for future mutations (Lange et al., 2026; Liu et al., 2024; Ye et al., 2024; Volkov et al., 2026). The useful experience can also be exploited for future search control, e.g., selection inspirations across diverse strategy clusters (Lange et al., 2026; Luo et al., 2026). Different from existing works that concentrate on the unit of experience as texts or programs, `ExpEvolve` discovers and exploits high-level idea factors to guide program mutations.

3 Theoretical Formulation of Evolutionary Discovery

In this section, we formalize evolutionary discovery and introduce the notion of ideas.

3.1 Evolutionary Discovery as a POMDP

Scientific discovery aims to uncover the causal mechanisms underlying a problem through interactions with the world (Kuhn & Hawkins, 1963). In evolutionary discovery, the agent interacts with the problem through an evaluator. We denote by $\theta_{\text{sci}} \in \Theta$ the hidden mechanism of the problem, which is fixed during the search and unknown to the agent. At turn t, the agent proposes a program $p_t \in \mathcal{P}$ and observes a score y_t with $\mathbb{E}[y_t \mid p_t] = F(p_t; \theta_{\text{sci}})$, where $F : \mathcal{P} \times \Theta \to \mathbb{R}$ is the objective, e.g., the value of a mathematical construction, the efficiency of a kernel, or the performance of a trained model (Novikov et al., 2025). The goal is to uncover θ_{sci} by optimizing for the evaluation score.

POMDP. As shown in Fig. 2B, the interaction forms a Partially Observable Markov Decision Process (POMDP) (Kaelbling et al., 1998) with a static hidden state $s_t = \theta_{\text{sci}}$, actions $a_t = p_t$, observations

$o_t = y_t$, an identity transition kernel, and the observation kernel $O(y_t \mid \theta_{\text{sci}}, p_t)$. Given a budget of T evaluations, the agent chooses $p_0, \ldots, p_{T-1}$ to find a program $\hat{p}$ with high $F(\hat{p}; \theta_{\text{sci}})$.

Evaluation as intervention. θ_{sci} can be parameterized as a structural causal model over variables of interest (Spirtes et al., 2000), in which a program specifies a configuration x_p of design variables X. Each evaluation can be considered as the expected outcome of the intervention:

$$F(p; \theta_{\text{sci}}) = \mathbb{E}\big[\, Y \,\big|\, \text{do}(X = x_p),\ \theta_{\text{sci}}\big]. \tag{1}$$

In this work, we leverage the causal view to provide a more in-depth understanding, while neither the analysis nor the method requires recovering the causal graph. A full discussion on the relations with causality can be found in Appendix C.

Experience-driven discovery. We define b_t as the agent's Bayesian belief after accumulating *experience* $h_t = \{(p_0, y_0), \ldots, (p_{t-1}, y_{t-1})\}$, which is essentially a probability distribution on Θ, $b_t(B) = \Pr(\Theta_{\text{sci}} \in B \mid h_t)$ for $B \subseteq \Theta$. In the ideal Bayesian formalism, the belief b_t is a sufficient statistic for decision-making (Kaelbling et al., 1998). In practice, the AI Scientist maintains an internal belief within LLMs, or an external one realized as memory $m_t = \Phi(h_t)$ for some (possibly learnable) summarization function Φ (Lange et al., 2026), to represent the *approximate representation* of its knowledge about θ_{sci} and the landscape of $F(\cdot; \theta_{\text{sci}})$. Each evaluation step (p_t, y_t) thus updates m_t, which in turn updates the agent's belief about θ_{sci} and determines the next action p_{t+1}.

3.2 Reusing Experience with Ideas

If the evaluator fully reveals θ_{sci} and the evaluation budget is unlimited, then enumerating every possible program can directly lead to discovery. However, in reality, the evaluation score only reveals partial information about θ_{sci} and evaluations can be expensive, e.g., training a model or running a physical experiment, so the agent can afford only a limited number of them (Wallace et al., 2026). Therefore, it is *essential* to reuse experience from past evaluations.

Without properly reusing experience, a single evaluation cannot reveal further insights about other unseen programs, and the agent then has to evaluate every candidate program to find the best one. Reusing experience requires some structure shared across programs, so that the evaluation of one program also informs the others. We formalize the contrast as follows.

Theorem 3.1 (Structured experience reuse, informal)**.** *Let $\mathcal{P} = \{p_1, \ldots, p_K\}$, and let each evaluation return $y_t = F(p_t; \theta_{\text{sci}}) + \varepsilon_t$ with $\varepsilon_t \sim \mathcal{N}(0, \sigma^2)$ i.i.d.*

(a) With reuse. *If each program has a known representation $\phi(p) \in \mathbb{R}^d$ with $\|\phi(p)\|_2 \le 1$ and $F(p; \theta_{\text{sci}}) = \langle \phi(p), w^\star \rangle$ for an unknown $w^\star \in \mathbb{R}^d$, then under a coverage condition on the representation, there exists a policy that, with probability at least $1 - \delta$, outputs a program within ϵ of the optimal value after $O\big(d\, \frac{\sigma^2}{\epsilon^2} \log \frac{K}{\delta}\big)$ evaluations.*

(b) Without reuse. *If programs are unrelated, i.e., $F(p_i; \theta_{\text{sci}}) = \mu_i$ for an arbitrary $\mu \in \mathbb{R}^K$, then any policy with the same guarantee for every μ needs $\Omega\big(K\, \frac{\sigma^2}{\epsilon^2} \log \frac{1}{\delta}\big)$ evaluations.*

The formal statement and proof are given in Appendix E. Theorem 3.1 implies that reusing experience at the high-level design choices can significantly improve sample efficiency from $\Omega(K)$ to $O(d \log K)$, where $K \gg d$. In evolutionary discovery, K counts the candidate program space, which grow combinatorially with the high-level design choices and further with the ways of implementing each choice, while d is determined by the design choices that affect the score. Most existing agents retain experience at the level of programs (Novikov et al., 2025; Lange et al., 2026), which ignores the high-level design choices. In contrast, human scientists recognize and record what they learn from experiments in terms of reusable concepts, e.g., a construction or an optimization technique, which carry over to later experiments. We are hence inspired to introduce the notion of *ideas*.

Definition 3.2 (Idea)**.** *An idea is an interpretable mapping $z : \mathcal{P} \to \mathcal{V}$ from programs to a binary or categorical value. It describes a high-level mechanistic property that can be shared by syntactically different programs, such as the construction family, initialization, or optimization procedure. A set of ideas $\{z_1, \ldots, z_r\}$ defines the idea representation $Z(p) = (z_1(p), \ldots, z_r(p))$.*

A program realizes a set of ideas together with low-level details. We call an idea representation Z *faithful* if the score depends on a program only through its ideas, i.e., $F(p; \theta_{\text{sci}}) = g(Z(p))$ for some

function g determined by θ_{sci}. Under a faithful idea representation, the hidden mechanism acts on a program only through its ideas, and Theorem 3.1 applies with $\phi(p)$ being the one-hot encoding of $Z(p)$, whose dimension is determined by the number of ideas.

Identifiability of ideas. However, the agent does not know in advance which representation is faithful, and the ideas have to be identified from the evaluated programs, while which programs are evaluated is decided by the search itself. When the search stops varying the candidate ideas, the exploration of ideas degenerates, as formalized below.

Theorem 3.3 (Degenerated exploration of ideas)**.** *Consider two candidate ideas z_0 and z_1, and two mechanisms θ_0 and θ_1 under which the score depends on z_0 and z_1, respectively, i.e., $F(p;\theta_i) = g(Z(p), z_i(p))$ for a common representation Z and function g, with the same observation noise. Suppose the optimal programs under θ_0 and θ_1 are different and unique with a margin $\Delta > 0$. Then for any policy that only evaluates programs on which z_0 and z_1 agree, there exists $i \in \{0, 1\}$ such that for every budget T, the output $\hat{p}$ satisfies $\mathbb{E}\big[\max_{p\in\mathcal{P}} F(p;\theta_i) - F(\hat{p};\theta_i)\big] \geq \Delta/2$.*

The formal statement and proof are given in Appendix F. Theorem 3.3 can be considered as a consequence of the partial observability of θ_{sci}, since the scores reveal θ_{sci} only through the evaluated programs. If two candidate ideas agree on all explored programs, the two mechanisms produce the same observations, and one cannot distinguish which idea determines the score. This situation is common in evolutionary search: as the search concentrates around its current best programs, the ideas shared by these programs stop varying. Identifying ideas thus requires sufficient exploration to contrast programs with and without each idea, to isolate contributions of ideas in the program space.

4 ExpEvolve: Discovering and Exploiting Ideas from Experience

Theoretical analysis in Sec. 3 suggests that a discovery agent should extract ideas from its experience, and keep exploring programs where the ideas vary. We realize the theoretical principles in `ExpEvolve`. The full procedure and implementation details are given in Alg. 1 and Appendix H.

4.1 Discovering Ideas from Experience

We leverage the discovery loop of the `COAT` framework (Liu et al., 2025) to find idea factors according to their theoretical properties.

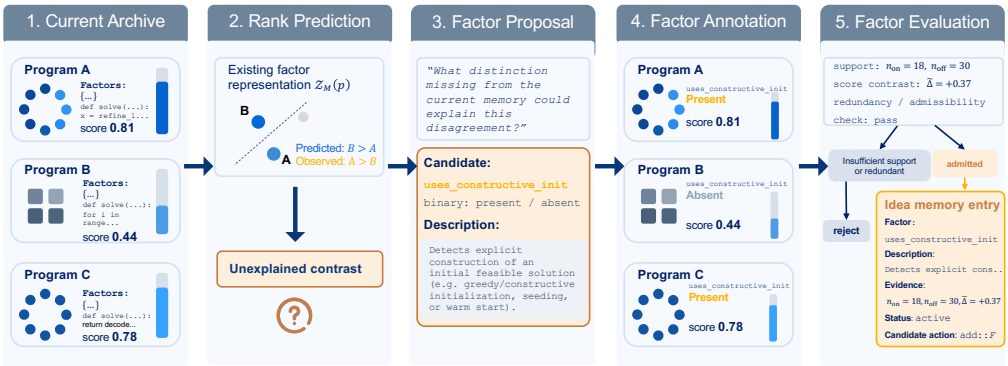


Figure 3: Illustration of idea factor discovery.

Idea memory. During the discovery, `ExpEvolve` maintains an idea memory $\mathcal{M}_t$ which is a set of idea factors. Each idea factor is stored with a name, a description of its code pattern, a type, i.e., binary or categorical, and a value set $\mathcal{V}$. For each program in the experience archive, the values of idea factors are annotated by an annotator LLM, which instantiates the idea representation $Z(p)$.

Explainability of idea factors. A faithful idea representation is expected to be able to predict the score of a program, i.e., $F(p;\theta_{\text{sci}}) = g(Z(p))$ (Sec. 3.2). However, predicting the exact scores from ideas is difficult, since the scores of different tasks lie on different scales, the unknown function g can be highly nonlinear, and the archive only contains a limited number of noisy evaluations. Therefore, we define the explainability of an idea representation by *its pairwise predictability*, i.e., whether $Z(p)$

and $Z(p')$ predict which of two programs p and p' is better. The ranking is already informative for optimizing the objective, invariant to the scale of the scores, and more robust to noise than the scores.

Discovering idea factors from unexplained pairs. The idea discovery process in `ExpEvolve` is an iterative loop: each time the discovery is triggered, `ExpEvolve` fits a pairwise ranker on the annotated idea factors, and selects the pairs of archived programs with a large score gap that the current idea factors fail to explain. The unexplained pairs suggest the *existence of missing idea factors* that separate the programs in each pair. For the selected pairs, a separate proposer LLM reads the programs to propose candidate idea factors that distinguish them. Each candidate is then annotated on the whole archive, and is admitted into the idea memory if each of its values is supported by enough programs and its values separate the scores.

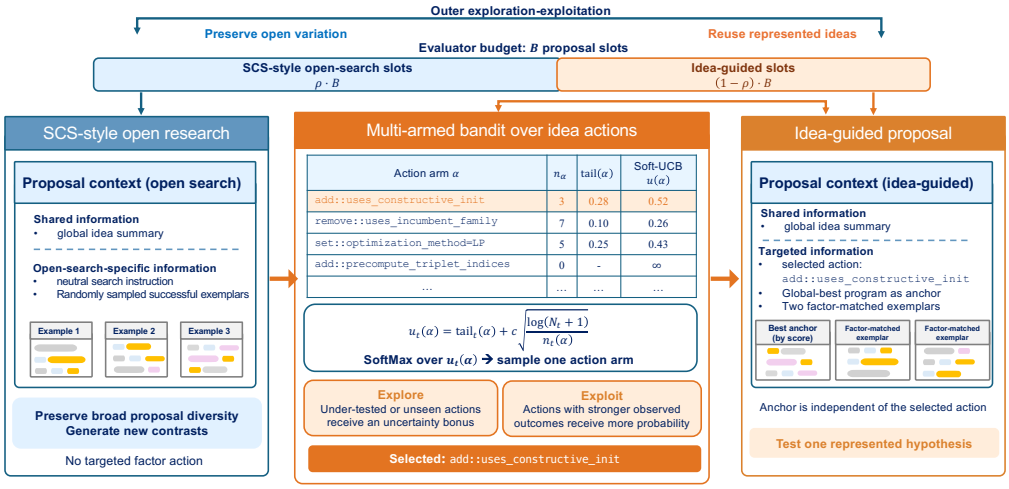


Figure 4: Illustration of two-level exploration-exploitation with discovered idea factors.

4.2 Exploiting Idea Factors for Discovery

We build a two-level exploration and exploitation framework with a multi-armed bandit (MAB) to fully exploit discovered idea factors while exploring varying design spaces.

Exploration and MAB exploitation. In each generation, `ExpEvolve` keeps at least a fraction ρ of the slots for random exploration that follows `SCS` (Gideoni et al., 2026) with randomly sampled example programs, which explores programs beyond the current ideas and provides new material for discovery. The remaining slots are allocated to idea-guided actions by MAB.

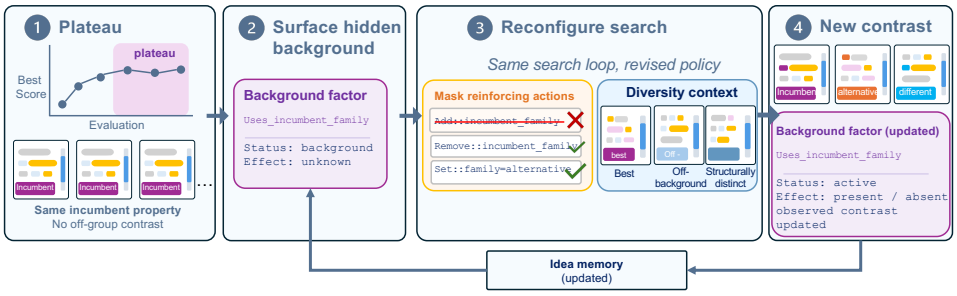


Figure 5: Illustration of idea-guided exploration at plateaus.

Idea factors as MAB actions. To exploit an idea factor z, `ExpEvolve` compiles it into edit actions, e.g., ADD(z) and REMOVE(z) for a binary idea factor. An action adds to the context an instruction that describes z, together with demonstration programs in the target state, to ask the mutator LLM to realize the action in the new program. The MAB selects the action for each idea-guided slot. The reward of an action is the score of the program generated under the action relative to the random exploration in the same generation, i.e.,$r = \frac{y - \bar{y}_{\text{rand}}}{\sigma_{\text{rand}}}$, where y is the score of the generated program, and $\bar{y}_{\text{rand}}$ and σ_{rand} are the mean and the standard deviation of the scores of random exploration in the same generation. The normalization removes the drift of scores across generations and the

Table 1: Final performance at the budget of 200 evaluated programs. Mean reports the seed mean ± seed standard deviation over 3 seeds, and Best reports the best seed.

Task	20 generations × 10 programs				10 generations × 20 programs				ShinkaEvolve	
	ExpEvolve		SCS		ExpEvolve		SCS			
	Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best
Hadamard ↑	**0.9211 ± 0.000**	**0.9211**	0.6641 ± 0.169	0.8596	**0.9259 ± 0.008**	**0.9357**	0.8349 ± 0.101	0.9211	0.6467 ± 0.139	0.8066
Pack ↑	**2.63542 ± 0.001**	**2.63598**	2.61697 ± 0.028	2.63572	**2.63499 ± 0.002**	**2.63598**	2.62305 ± 0.011	2.63104	2.58201 ± 0.037	2.61408
Linprog ↑	**2.63542 ± 0.001**	**2.63598**	**2.63542 ± 0.001**	**2.63598**	**2.63542 ± 0.001**	**2.63598**	2.55161 ± 0.006	2.55608	2.51434 ± 0.026	2.54142
SumDiff ↑	**1.11609 ± 0.006**	**1.11937**	1.11087 ± 0.001	1.11143	**1.12224 ± 0.001**	**1.12281**	1.11608 ± 0.005	1.12006	1.10954 ± 0.000	1.10954
Heilbronn ↑	**0.0353 ± 0.001**	**0.0359**	0.0313 ± 0.003	0.0340	**0.0358 ± 0.001**	**0.0365**	0.0336 ± 0.001	0.0343	0.0303 ± 0.002	0.0324
Min overlap ↓	**0.3810 ± 0.000**	**0.3810**	0.3816 ± 0.001	**0.3810**	**0.3810 ± 0.000**	**0.3810**	**0.3810 ± 0.000**	**0.3810**	0.3824 ± 0.000	0.3824
Kissing ↑	**526 ± 97**	**582**	413 ± 23	438	**582 ± 0**	**582**	**582 ± 0**	**582**	462 ± 90	566
Second autocorr. ↑	**0.8825 ± 0.000**	**0.8825**	0.8398 ± 0.016	0.8499	**0.8825 ± 0.000**	**0.8825**	**0.8825 ± 0.000**	**0.8825**	0.8745 ± 0.008	0.8825
First autocorr. ↓	**1.5113 ± 0.002**	**1.5098**	1.5235 ± 0.011	1.5151	**1.5195 ± 0.006**	**1.5151**	1.5508 ± 0.011	1.5378	1.5540 ± 0.029	1.5203
Max–min ratio ↓	**12.9262 ± 0.064**	**12.8892**	12.9338 ± 0.077	**12.8892**	**12.8892 ± 0.000**	**12.8892**	12.9808 ± 0.033	12.9425	13.0000 ± 0.000	13.0000
Hermite ↓	**0.35473 ± 0.003**	**0.35263**	0.39431 ± 0.059	0.35945	**0.35089 ± 0.002**	**0.34820**	0.35956 ± 0.005	0.35394	4.72779 ± 3.788	0.35429

differences of score scales across tasks. The MAB selects the actions with an upper confidence bound (UCB) rule on the maximum reward of each action (Appendix H.3).

Exploration at plateaus. Oftentimes, evolutionary discovery can concentrate around its best programs and a single strategy. Consequently, the ideas widely used in top-scoring programs stop varying, and their roles cannot be identified from the archive (Thm. 3.3). When the best score does not improve for κ generations, ExpEvolve switches to idea-guided exploration, which enforces the mutator LLM to explore diverse ideas other than those widely used in top-scoring programs. Specifically, a separate LLM probes the properties shared by the top-scoring programs as background idea factors, and an LLM judge determines which idea factors in the memory depend on the background. ExpEvolve then masks the actions of the dependent idea factors, instructs the mutator LLM to avoid them, and assigns more slots to random exploration with diversified demonstrations.

5 Experiments

We evaluate ExpEvolve on 11 open-ended mathematical discovery tasks and further integrate it into an evolutionary agent for Machine Learning Engineering (MLE) tasks.

5.1 Experimental Setup

Tasks. We evaluate on 11 open-ended mathematical discovery tasks. Nine tasks are the problems studied by SCS (Gideoni et al., 2026), which are taken from AlphaEvolve (Novikov et al., 2025), including the first and second autocorrelation inequalities, the uncertainty inequality, the Erdős minimum overlap problem, sums and differences of sets, the max–min distance ratio, the Heilbronn problem for triangles, the kissing number in 11 dimensions, and circle packing. The other two tasks are the Hadamard matrix problem (Wang et al., 2026) and a variant of circle packing that recovers the radii by linear programming. We report the best score at the end of the budget, as the mean and standard deviation over seeds, and the best seed. Full details of the tasks are given in Appendix I.1.

Budget. Each run evaluates 200 programs. We consider two configurations with the same budget, i.e., 10 generations of 20 programs as in SCS (Gideoni et al., 2026) and 20 generations of 10 programs, which presents more challenges to accumulating reusable knowledge from experience.

Implementation. We compare ExpEvolve with SCS (Gideoni et al., 2026), which is the backbone of ExpEvolve and shows competitive results against more elaborate evolutionary coding agents. We further compare with ShinkaEvolve (Lange et al., 2026), which maintains an island-based archive and summarizes the insights of recent programs with a meta-summary. Details are given in Appendix I.2. We use DeepSeek-V4-Flash by default, and each setting is run with 3 seeds. ExpEvolve uses a random-exploration fraction $\rho = 0.60$ for 20-gen. and $\rho = 0.40$ for 10-gen. to keep a similar number of random-exploration slots, i.e., 6 and 8. We donot tune other hyperparameters.

5.2 Main Results

ExpEvolve consistently improves the final performance. Table 1 shows the final performance at the budget of 200 programs. ExpEvolve improves the mean and best final performances across

most settings over SCS and ShinkaEvolve. Moreover, the improvements are robust to how the budget is parallelized. From 10 generations of 20 programs to 20 generations of 10 programs, the mean final score of SCS can drop dramatically because of limited capability in experience reuse, while ExpEvolve still maintains competitive performance. More details are given in Appendix J.

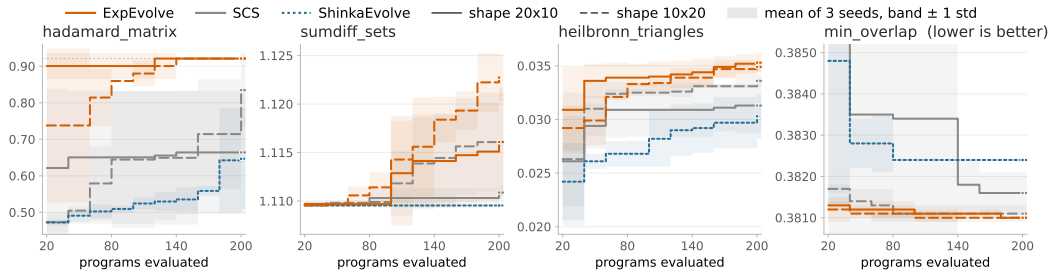


Figure 6: Best-so-far scores along the budget of 200 evaluated programs.

ExpEvolve improves sample efficiency. Fig. 6 shows the best-so-far scores along the budget on Hadamard, Sums and differences, Heilbronn, and Minimum overlap. On these four tasks, ExpEvolve needs 1.7× to 26.2× fewer evaluations than SCS under both configurations.

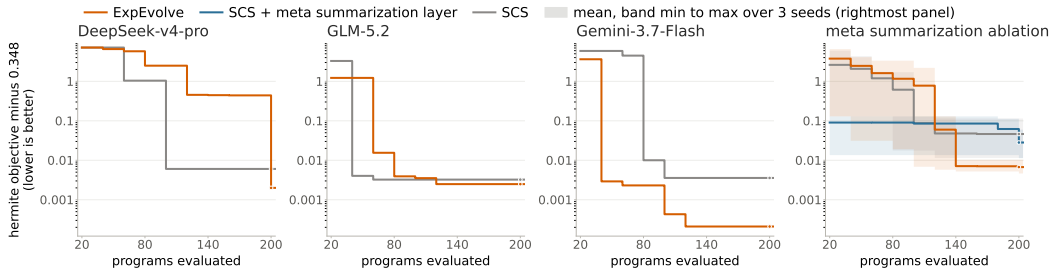


Figure 7: (a) Best-so-far scores of SCS and ExpEvolve with other LLMs. (b) Best-so-far scores of SCS, SCS+meta, and ExpEvolve.

The improvements transfer to other LLMs. Fig. 7 (a) shows that the advantages of ExpEvolve over SCS transfers to other LLMs such as DeepSeek-v4-pro, GLM-5.2 and Gemini-3.7-Flash.

Summarizing experience as memory helps the early search but saturates. ShinkaEvolve and SCS+meta retain the experience as a textual summary of recent programs, which provides additional context to the mutator LLM without controlling which ideas are evaluated. Fig. 7 (b) shows that SCS+meta improves over SCS in the early stage. However, its improvement slows down afterwards while ExpEvolve keep improving. It again verifies the advantages of idea-guided evolution.

The discovered idea factors explain the order of archived programs. Following the definition of explainability in Sec. 4.1, we measure the fraction of archived program pairs that discovered factors can order correctly. As shown in Fig. 8(a), the discovered idea factors reach an accuracy of 0.79 to 0.99. Fig. 8(b) shows the idea factor with the largest weight on each task.

In an example run, idea-guided actions produce about half of the new best programs. Fig. 8(c) shows a run on Sums and differences with 10 generations of 20 programs. 6 of the 11 new best programs come from idea-guided slots. The 92 idea-guided slots of the run select 7 distinct actions, and all the new best programs from idea-guided slots come from 2 of them.

5.3 Additional Experiments on Machine Learning Engineering Tasks

We further study whether the effectiveness of ExpEvolve transfers to MLE by integrating it into OpenRSI (Yang et al., 2026b) and evaluating it on 6 tasks from MLS-Bench (Lyu et al., 2026a): *Online bandit*, *Anomaly detection*, *Clustering*, *Multi-objective optimization*, *Subgroup calibration shift*, and *Evolution strategy*. Search leverages the scores and execution feedback from each task.

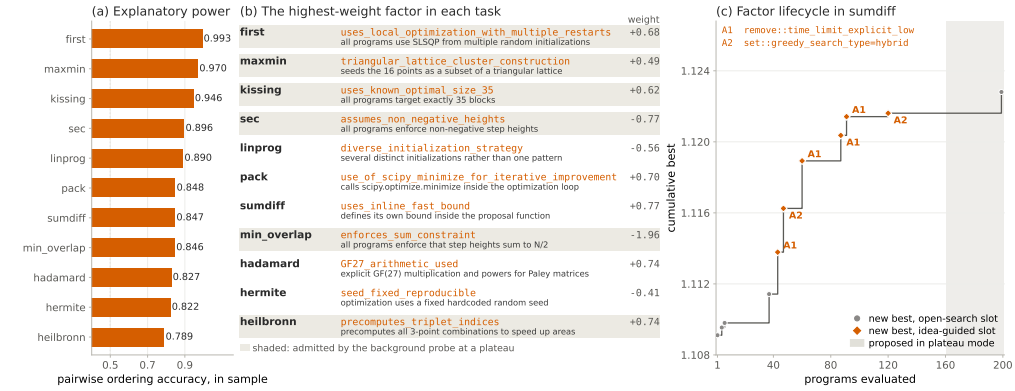
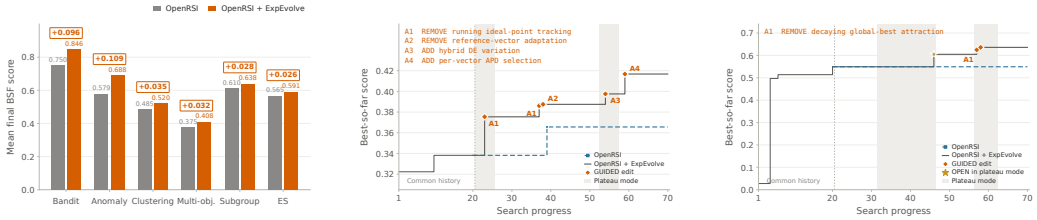


Figure 8: Quality of discovered idea factors. (a) Pairwise accuracy of the discovered idea factors. (b) The idea factor with the largest weight. (c) Illustration of usage of factors in sumdiff task.



(a) Final BSF performance (b) Multi-objective optimization (c) Evolution strategy

Figure 9: Results on MLS-Bench. (a) Mean final scores. (b,c) Trajectories on two example tasks.

Experimental setup. We use the evolutionary harness from OpenRSI, which organizes search through DRAFT, IMPROVE, DEBUG, and CROSSOVER operators. For each task, we retain a fixed history containing 20 completed search slots from an earlier OpenRSI run. The OpenRSI baseline and OpenRSI equipped with `ExpEvolve` continue from this same history for **50** additional evaluator calls, with **3** seeds. Both methods use **5** search slots per generation. All program generation uses Qwen3.6-35B-A3B. The same checkpoint performs factor discovery and annotation in `ExpEvolve`. For each task and method, we average the final best-so-far (BSF) scores over three seeds.

Idea-guided evolution transfers to MLE tasks. Fig. 9(a) compares OpenRSI with OpenRSI+`ExpEvolve` across the six tasks, where `ExpEvolve` shows superior performance on all six tasks, with the largest gains on Online bandit and Anomaly detection.

Idea-guided actions produce better exploration and exploitation on MLE tasks. Fig. 9(b) and Fig. 9(c) show two concrete trajectories in the tasks of multi-objective optimization and evolution strategy, respectively. We can find that `ExpEvolve` can effectively discover useful ideas such as removing running ideal-point tracking (A1), removing reference-vector adaptation (A2), and adding hybrid differential-evolution variation (A3) to effectively boost the performance. When the evolution gets stuck at plateaus, exploiting useful strategies proposed by `ExpEvolve` in multi-objective optimization, or exploration guided by idea factors in `ExpEvolve` can effectively break the plateaus and lead to improvements. More details about the implementation can be found in Appendix K.1.

6 CONCLUSIONS

We introduced `ExpEvolve`, which structures the experience of evolutionary discovery into idea factors and reuses them to guide the search. `ExpEvolve` discovers idea factors from the pairs of programs that the current idea factors fail to explain, exploits them as the actions of a multi-armed bandit, and keeps exploring programs where the ideas vary, through random exploration and idea-guided exploration at plateaus. Our analysis shows that reusing a faithful idea representation reduces the number of evaluations from $\Omega(K)$ to $O(d \log K)$, and that identifying ideas requires exploring programs where the ideas vary. On 11 mathematical discovery tasks and 6 tasks from MLS-Bench, `ExpEvolve` improves the final performance and the sample efficiency of evolutionary discovery under the same evaluation budget.

Ethics Statement

This work studies evolutionary coding agents on open-ended mathematical discovery tasks and on machine learning tasks from MLS-Bench. It does not involve human subjects or personally identifiable information, and all tasks come from public benchmarks or prior work.

Reproducibility Statement

The full procedure of `ExpEvolve` is given in Alg. 1 and Alg. 2, and its implementation details and hyperparameters are given in Appendix H and Table 3. The tasks, evaluators, and reference values are described in Appendix I.1, the configurations of the baselines in Appendix I.2, the budget and resources in Appendix I.3, the setting of MLS-Bench in Appendix K.1, and all prompts in Appendix K. The per-seed results are given in Appendix J. The formal statements and proofs of the theoretical results are given in Appendix E and Appendix F. All experiments on mathematical discovery use DeepSeek-V4-Flash unless stated otherwise, and each setting is run with 3 seeds. We will provide an anonymized link to our code upon the agreement of the chairs during the discussion period.

AI use statement

We used LLM assistance to revise the manuscript's organization and wording, identify and check related references, and examine the interpretation of reported experiments. The proposed framework itself uses LLMs to generate programs, propose idea factors, and assess program properties.

References

Kartik Ahuja, Divyat Mahajan, Yixin Wang, and Yoshua Bengio. Interventional causal representation learning. In *International Conference on Machine Learning (ICML)*, 2023. (Cited on page 21)

Anthropic. Claude accelerates protein design and analytical chemistry. https://www.anthropic.com/research/Claude-accelerates-protein-design, August 2026. 354 binders against 14 of 15 targets from 1,320 designs, produced and tested independently by Adaptyv Bio and Twist Bioscience. (Cited on pages 1, 3 and 19)

Akari Asai, Jacqueline He, Rulin Shao, Weijia Shi, Amanpreet Singh, Joseph Chee Chang, Kyle Lo, Luca Soldaini, Sergey Feldman, et al. Synthesizing scientific literature with retrieval-augmented language models. *Nature*, 2026. doi: 10.1038/s41586-025-10072-4. (Cited on pages 1 and 19)

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: an open source evolutionary coding agent for algorithm discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025. (Cited on page 19)

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. (Cited on page 19)

Philippe Brouillard, Sébastien Lachapelle, Alexandre Lacoste, Simon Lacoste-Julien, and Alexandre Drouin. Differentiable causal discovery from interventional data. *Advances in Neural Information Processing Systems*, 33:21865–21877, 2020. (Cited on page 20)

Sébastien Bubeck, Christian Coester, Ronen Eldan, Timothy Gowers, Yin Tat Lee, Alexandru Lupsasca, Mehtaab Sawhney, Robert Scherrer, Mark Sellke, Brian K. Spears, Derya Unutmaz, Kevin Weil, Steven Yin, and Nikita Zhivotovskiy. Early science acceleration experiments with gpt-5. *ArXiv*, abs/2511.16072, 2025. (Cited on page 19)

Mert Cemri et al. AdaEvolve: Adaptive LLM driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026. (Cited on page 19)

Krzysztof Chalupka, Pietro Perona, and Frederick Eberhardt. Visual causal feature learning. In *Proceedings of the Thirty-First Conference on Uncertainty in Artificial Intelligence*, 2015. (Cited on page 21)

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. MLE-bench: Evaluating machine learning agents on machine learning engineering. In *ICLR*, 2025. arXiv:2410.07095. (Cited on page 19)

Yongqiang Chen, Wei Huang, Kaiwen Zhou, Yatao Bian, Bo Han, and James Cheng. Understanding and improving feature learning for out-of-distribution generalization. In *Advances in Neural Information Processing Systems*, 2023. (Cited on page 22)

Audrey Cheng, Shu Liu, Melissa Z. Pan, Zhifei Li, Bowen Wang, Alexander Krentsel, Tian Xia, Mert Cemri, Jongseok Park, Shuo Yang, Jeff Chen, Lakshya A Agrawal, Aditya Desai, Jiarong Xing, Koushik Sen, Matei Zaharia, and Ion Stoica. Barbarians at the gate: How ai is upending systems research. *ArXiv*, abs/2510.06189, 2025. (Cited on page 19)

Luis Da Costa, Álvaro Fialho, Marc Schoenauer, and Michèle Sebag. Adaptive operator selection with dynamic multi-armed bandits. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pp. 913–920, 2008. (Cited on page 19)

Nikita Dhawan, Leonardo Cotta, Karen Ullrich, Rahul G Krishnan, and Chris J Maddison. End-to-end causal effect estimation from unstructured natural language data. *Advances in Neural Information Processing Systems*, 37:77165–77199, 2024. (Cited on page 20)

Igor Douven. Abduction. In Edward N. Zalta and Uri Nodelman (eds.), *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2025 edition, 2025. (Cited on page 1)

Marina Favaro and Jack Clark. When AI builds itself: Our progress toward recursive self-improvement and its implications. Anthropic Institute, https://www.anthropic.com/institute/recursive-self-improvement, June 2026. (Cited on pages 1, 3 and 19)

Peilin Feng, Zhutao Lv, Junyan Ye, Xiaolei Wang, Xinjie Huo, Jinhua Yu, Wanghan Xu, Wenlong Zhang, Lei Bai, Conghui He, and Weijia Li. Earth-Agent: Unlocking the full landscape of earth observation with agents. In *ICLR*, 2026a. arXiv:2509.23141. (Cited on page 19)

Shiyang Feng, Runmin Ma, Xiangchao Yan, et al. InternAgent-1.5: A unified agentic framework for long-horizon autonomous scientific discovery. *arXiv preprint arXiv:2602.08990*, 2026b. (Cited on pages 3 and 19)

Bogdan Georgiev, Javier Gómez-Serrano, Terence Tao, and Adam Zsolt Wagner. Mathematical exploration and discovery at scale. *ArXiv*, abs/2511.02864, 2025. (Cited on page 19)

Alireza Ghafarollahi and Markus J Buehler. Sciagents: automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025. (Cited on pages 3 and 19)

Ali Essam Ghareeb et al. A multi-agent system for automating scientific discovery. *Nature*, 655 (8122):497–505, 2026. doi: 10.1038/s41586-026-10652-y. (Cited on pages 3 and 19)

Yonatan Gideoni, Sebastian Risi, and Yarin Gal. Simple baselines are competitive with code evolution. In *ICLR 2026 Workshop on AI with Recursive Self-Improvement*, 2026. arXiv:2602.16805. (Cited on pages 1, 2, 3, 6, 7, 19, 29, 33, 34 and 35)

Clark Glymour. An outline of the history of methods of discovering causality. URL https://www.cmu.edu/dietrich/philosophy/docs/glymour/an-outline-of-the-history-of-methods-of-discovering-causality.pdf. Accessed: 2026-01-29. (Cited on page 2)

Juraj Gottweis et al. Accelerating scientific discovery with co-scientist. *Nature*, 655(8122):487–496, 2026. doi: 10.1038/s41586-026-10644-y. (Cited on pages 1, 3 and 19)

Sander Greenland, Judea Pearl, and James M Robins. Causal diagrams for epidemiologic research. *Epidemiology*, 10(1):37–48, 1999. (Cited on page 20)

Patrik Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. *Advances in neural information processing systems*, 21, 2008. (Cited on page 20)

Biwei Huang, Kun Zhang, Jiji Zhang, Joseph Ramsey, Ruben Sanchez-Romero, Clark Glymour, and Bernhard Schölkopf. Causal discovery from heterogeneous/nonstationary data. *Journal of Machine Learning Research*, 21(89):1–53, 2020. (Cited on page 20)

Kexin Huang, Ying Jin, Ryan Li, Michael Y Li, Emmanuel Candes, and Jure Leskovec. Automated hypothesis validation with agentic sequential falsifications. In *ICML*, 2025a. (Cited on page 19)

Yuxuan Huang, Yihang Chen, Haozheng Zhang, Kang Li, Huichi Zhou, Meng Fang, Linyi Yang, Xiaoguang Li, Lifeng Shang, Songcen Xu, et al. Deep research agents: A systematic examination and roadmap. *arXiv preprint arXiv:2506.18096*, 2025b. (Cited on page 19)

Antti Hyttinen, Frederick Eberhardt, and Patrik O Hoyer. Experiment selection for causal discovery. *The Journal of Machine Learning Research*, 14(1):3041–3071, 2013. (Cited on page 20)

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. DSBench: How far are data science agents from becoming data science experts? In *ICLR*, 2025. arXiv:2409.07703. (Cited on page 19)

Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998. (Cited on pages 2, 3, 4 and 20)

Andrej Karpathy. autoresearch: session reports of an autonomous single-GPU training-loop agent. GitHub discussions #32 and #43, https://github.com/karpathy/autoresearch/discussions/32, https://github.com/karpathy/autoresearch/discussions/43, 2026. Discussion #43 is the 126-experiment log classified in the paper; discussion #32 (8 March 2026) is an 89-experiment log with the same format. (Cited on pages 1, 3 and 19)

Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best-arm identification in multi-armed bandit models. *Journal of Machine Learning Research*, 17(1):1–42, 2016. (Cited on page 27)

Valentin Khrulkov, Andrey V. Galichin, et al. GigaEvo: An open source optimization framework powered by LLMs and evolution algorithms. *arXiv preprint arXiv:2511.17592*, 2025. (Cited on page 19)

Thomas S. Kuhn and David Hawkins. The structure of scientific revolutions. *American Journal of Physics*, 31:554–555, 1963. (Cited on pages 1, 3 and 20)

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. ShinkaEvolve: Towards open-ended and sample-efficient program evolution. In *ICLR*, 2026. arXiv:2509.19349. (Cited on pages 1, 3, 4, 7, 19 and 35)

Jon M Laurent, Joseph D Janizek, Michael Ruzo, Michaela M Hinks, Michael J Hammerling, Siddharth Narayanan, Manvitha Ponnapati, Andrew D White, and Samuel G Rodriques. Lab-bench: Measuring capabilities of language models for biology research. *arXiv preprint arXiv:2407.10362*, 2024. (Cited on page 19)

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. (Cited on page 19)

Jin Li, Shoujin Wang, Qi Zhang, Feng Liu, Tongliang Liu, Longbing Cao, Shui Yu, and Fang Chen. Revealing multimodal causality with large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. (Cited on pages 20 and 21)

Junyi Li, Yongqiang Chen, Chenxi Liu, Qianyi Cai, Tongliang Liu, Bo Han, Kun Zhang, and Hui Xiong. Can large language models help experimental design for causal discovery? *ArXiv*, abs/2503.01139, 2025b. URL https://arxiv.org/abs/2503.01139. (Cited on pages 1 and 20)

Long Li, Weiwen Xu, Jiayan Guo, Ruochen Zhao, Xingxuan Li, Yuqian Yuan, Boqiang Zhang, Yuming Jiang, Yifei Xin, Ronghao Dang, Yu Rong, Deli Zhao, Tian Feng, and Lidong Bing. Chain of ideas: Revolutionizing research via novel idea development with LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 8971–9004, 2025c. doi: 10.18653/v1/2025.findings-emnlp.477. (Cited on page 19)

Michael Y Li, Vivek Vajipey, Noah D Goodman, and Emily B Fox. Critical: Critic automation with language models. *arXiv preprint arXiv:2411.06590*, 2024. (Cited on page 19)

Xiu-Chuan Li and Tongliang Liu. Efficient and trustworthy causal discovery with latent variables and complex relations. In *The Thirteenth International Conference on Learning Representations*, 2025. (Cited on page 20)

Xiu-Chuan Li, Jun Wang, and Tongliang Liu. Recovery of causal graph involving latent variables via homologous surrogates. In *The Thirteenth International Conference on Learning Representations*, 2025d. (Cited on page 20)

Weixin Liang, Yuhui Zhang, Hancheng Cao, Binglu Wang, Daisy Yi Ding, Xinyu Yang, Kailas Vodrahalli, Siyu He, Daniel Scott Smith, Yian Yin, et al. Can large language models provide useful feedback on research papers? a large-scale empirical analysis. *NEJM AI*, 1(8):AIoa2400196, 2024. (Cited on page 19)

Chenxi Liu, Yongqiang Chen, Tongliang Liu, Mingming Gong, James Cheng, Bo Han, and Kun Zhang. Discovering and reasoning of causality in the hidden world with large language models, 2025. URL https://arxiv.org/abs/2402.03941. (Cited on pages 2, 5, 20 and 21)

Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In *International Conference on Machine Learning (ICML)*, 2024. (Cited on page 3)

Yujie Liu, Zonglin Yang, Tong Xie, Jinjie Ni, Ben Gao, Yuqiang Li, Shixiang Tang, Wanli Ouyang, Erik Cambria, and Dongzhan Zhou. ResearchBench: Benchmarking LLMs in scientific discovery via inspiration-based task decomposition. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 13187–13207, 2026. (Cited on page 19)

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. (Cited on page 1)

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. Towards end-to-end automation of AI research. *Nature*, 651(8107):914–919, 2026. doi: 10.1038/s41586-026-10265-5. (Cited on pages 3 and 19)

Sichun Luo, Yi Huang, Haochen Luo, Fengyuan Liu, Guanzhi Deng, Lei Li, Qinghua Yao, Zefa Hu, Junlan Feng, and Qi Liu. SeaEvo: Advancing algorithm discovery with strategy space evolution. *arXiv preprint arXiv:2604.24372*, 2026. (Cited on page 3)

Haoze Lv, Ning Lu, Shengcai Liu, Shaofeng Zhang, and Ke Tang. MuEvo: LLM-driven evolution of multi-heuristic ensemble. *arXiv preprint arXiv:2608.03636*, 2026. (Cited on page 19)

Bohan Lyu, Yucheng Yang, Siqiao Huang, Jiaru Zhang, Qixin Xu, Xinghan Li, Xinyang Han, Yicheng Zhang, Huaqing Zhang, Runhan Huang, Kaicheng Yang, Zitao Chen, Wentao Guo, Junlin Yang, Xinyue Ai, Wenhao Chai, Yadi Cao, Ziran Yang, Kun Wang, Dapeng Jiang, Huan-ang Gao, Shange Tang, Chengshuai Shi, Simon S. Du, Max Simchowitz, Jiantao Jiao, Dawn Song, and Chi Jin. MLS-Bench: A holistic and rigorous assessment of AI systems on building better AI. *arXiv preprint arXiv:2605.08678*, 2026a. (Cited on pages 1, 2, 3, 8, 19 and 39)

Yougang Lyu, Xi Zhang, Xinhao Yi, Yuyue Zhao, Shuyu Guo, Wenxiang Hu, Jan Piotrowski, Jakub Kaliski, Jacopo Urbani, Zaiqiao Meng, Lun Zhou, and Xiaohui Yan. EvoScientist: Towards multi-agent evolving AI scientists for end-to-end scientific discovery. *arXiv preprint arXiv:2603.08127*, 2026b. (Cited on pages 3 and 19)

Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. DiscoveryBench: Towards data-driven discovery with large language models. In *International Conference on Learning Representations (ICLR)*, 2025. (Cited on pages 1 and 19)

Indrajeet Mandal, Jitendra Soni, Mohd Zaki, Morten M Smedskjaer, Katrin Wondraczek, Lothar Wondraczek, Nitya Nand Gosvami, and NM Anoop Krishnan. Evaluating large language model agents for automation of atomic force microscopy. *Nature Communications*, 16(1):9104, 2025. (Cited on page 19)

Ludovico Mitchener, Angela Yiu, Benjamin Chang, Mathieu Bourdenx, Tyler Nadolski, Arvis Sulovari, Eric C. Landsness, Dániel L. Barabási, Siddharth Narayanan, Nicky Evans, Shriya Reddy, Martha S. Foiani, Aizad Kamal, Leah P. Shriver, Fang Cao, Asmamaw T. Wassie, Jon M. Laurent, Edwin Melville-Green, Mayk Caldas Ramos, Albert Bou, Kaleigh F. Roberts, Sladjana Zagorac, Timothy C. Orr, Miranda E. Orr, Kevin J. Zwezdaryk, Ali E. Ghareeb, Laurie McCoy, Bruna Gomes, Euan A Ashley, Karen E. Duff, Tonio Buonassisi, Tom Rainforth, Randall J. Bateman, Michael Skarlinski, Samuel G. Rodriques, Michaela M. Hinks, and Andrew D. White. Kosmos: An ai scientist for autonomous discovery. *ArXiv*, abs/2511.02824, 2025. (Cited on pages 1, 3 and 19)

Joris M Mooij, Sara Magliacane, and Tom Claassen. Joint causal inference from multiple contexts. *Journal of machine learning research*, 21(99):1–108, 2020. (Cited on page 20)

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015. (Cited on page 19)

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. (Cited on pages 1, 3, 4, 7, 19, 20 and 33)

OpenAI. Research acceleration: The view inside OpenAI. OpenAI blog, 6 September 2026, September 2026a. Reports the automated research intern milestone and 3.1 agent-workdays per human workday in the research organization. (Cited on pages 1, 3 and 19)

OpenAI. Ten advances in mathematics and theoretical computer science. https://openai.com/index/ten-advances-in-mathematics/, August 2026b. Results on ten open problems with Lean 4 formal proofs; peer review pending at the time of writing. (Cited on pages 1, 3 and 19)

OpenAI. An OpenAI model has disproved a central conjecture in discrete geometry. https://openai.com/index/model-disproves-discrete-geometry-conjecture/, May 2026c. Counterexample to the Erdős unit distance conjecture, verified by external mathematicians with a companion paper. (Cited on pages 1 and 19)

Judea Pearl. *Causality*. Cambridge university press, 2009. (Cited on page 20)

Nico Pelleriti et al. What do evolutionary coding agents evolve? *arXiv preprint arXiv:2605.20086*, 2026. (Cited on page 1)

Ronan Perry, Julius Von Kügelgen, and Bernhard Schölkopf. Causal discovery in heterogeneous environments under the sparse mechanism shift hypothesis. *Advances in Neural Information Processing Systems*, 35:10904–10917, 2022. (Cited on page 20)

Aske Plaat, Max van Duijn, Niki van Stein, Mike Preuss, Peter van der Putten, and Kees Joost Batenburg. Agentic large language models, a survey. *arXiv preprint arXiv:2503.23037*, 2025. (Cited on page 1)

Joaquin Quinonero-Candela, Masashi Sugiyama, Anton Schwaighofer, and Neil D Lawrence. *Dataset shift in machine learning*. Mit Press, 2008. (Cited on page 22)

Loïc M. Roch et al. Chemos: An orchestration software to democratize autonomous discovery. *PLOS ONE*, 15(4):e0229862, 2020. (Cited on page 19)

Bernardino Romera-Paredes et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. (Cited on pages 3 and 19)

Paul K. Rubenstein, Sebastian Weichwald, Stephan Bongers, Joris M. Mooij, Dominik Janzing, Moritz Grosse-Wentrup, and Bernhard Schölkopf. Causal consistency of structural equation models. In *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*, 2017. (Cited on page 21)

Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021. (Cited on page 21)

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve. (Cited on page 19)

Ivaxi Sheth, Zhijing Jin, Bryan Wilder, Dominik Janzing, and Mario Fritz. Can llms propose instrumental variables for causal reasoning? In *NeurIPS 2025 Workshop on CauScien: Uncovering Causality in Science*, 2025. (Cited on page 20)

Shohei Shimizu, Patrik O Hoyer, Aapo Hyvärinen, Antti Kerminen, and Michael Jordan. A linear non-gaussian acyclic model for causal discovery. *Journal of Machine Learning Research*, 7(10), 2006. (Cited on page 20)

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K. Reddy. LLM-SR: Scientific equation discovery via programming with large language models. In *ICLR*, 2025a. (Cited on page 19)

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K Reddy. Llm-srbench: A new benchmark for scientific equation discovery with large language models. In *ICML*, 2025b. (Cited on page 19)

Chenglei Si, Tatsunori Hashimoto, and Diyi Yang. The ideation–execution gap: Execution outcomes of LLM-generated versus human research ideas. In *International Conference on Learning Representations (ICLR)*, 2026. (Cited on pages 3 and 19)

Marta Soare, Alessandro Lazaric, and Rémi Munos. Best-arm identification in linear bandits. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. (Cited on page 27)

Peter Spirtes, Christopher Meek, and Thomas Richardson. Causal inference in the presence of latent variables and selection bias. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pp. 499–506, 1995. (Cited on page 20)

Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*. MIT press, 2000. (Cited on pages 4 and 20)

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, et al. Paperbench: Evaluating ai's ability to replicate ai research. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pp. 56843–56873, 2025. (Cited on page 19)

Anja Šurina, Amin Mansouri, Lars C. P. M. Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with LLMs: Evolutionary search meets reinforcement learning. In *Conference on Language Modeling (COLM)*, 2025. arXiv:2504.05108. (Cited on page 20)

Kyle Swanson, Wesley Wu, Nash L Bulaong, John E Pak, and James Zou. The virtual lab of ai agents designs new sars-cov-2 nanobodies. *Nature*, 646(8085):716–723, 2025. (Cited on page 19)

Guiyao Tie, Pan Zhou, and Lichao Sun. A survey of AI scientists. *arXiv preprint arXiv:2510.23045*, 2025. (Cited on page 19)

Gary Tom, Stefan P Schmid, Sterling G Baird, Yang Cao, Kourosh Darvish, Han Hao, Stanley Lo, Sergio Pablo-García, Ella M Rajaonson, Marta Skreta, et al. Self-driving laboratories for chemistry and materials science. *Chemical Reviews*, 124(16):9633–9732, 2024. (Cited on page 19)

Daniel Truhn, Shekoofeh Azizi, James Zou, Leonor Cerda-Alberich, Faisal Mahmood, and Jakob Nikolas Kather. Artificial intelligence agents in cancer research and oncology. *Nature Reviews Cancer*, 26(4):256–269, 2026. (Cited on page 19)

Viktor Volkov, Valentin Khrulkov, Andrey V. Galichin, Danil Sivtsov, Nikita Glazkov, Olga Volkova, Konstantin Pchelin, Iaroslav Bespalov, Dmitry V. Dylov, Petr Anokhin, and Ivan Oseledets. EvoMem: Memory-augmented evolution for code optimization. *arXiv preprint arXiv:2608.10795*, 2026. (Cited on page 3)

Don Wallace, Conor Griffin, Sean O'Neill, Thang Luong, and Owen Larter. Conjecture machines: AI agents and the new validation bottleneck in science. Google DeepMind, https://deepmind.google/public-policy/conjecture-machines-ai-agents-and-the-new-validation-bottleneck-in-science/, July 2026. (Cited on pages 1, 3, 4 and 19)

W.A. Wallace. *Causality and Scientific Explanation*. Number v. 2 in Causality and Scientific Explanation. University Press of America, 1981. ISBN 9780819114815. (Cited on page 2)

Haiyuan Wan, Chen Yang, Junchi Yu, Meiqi Tu, Jiaxuan Lu, Di Yu, Jianbao Cao, Ben Gao, Jiaqing Xie, Aoran Wang, et al. Deepresearch arena: The first exam of llms' research abilities via seminar-grounded tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 33341–33349, 2026. (Cited on page 19)

Hanchen Wang, Yichun He, Paula P. Coelho, Matthew Bucci, Abbas Nazir, Bob Chen, Linh T. Trinh, Serena Zhang, Ziyu Lu, Kexin Huang, et al. SpatialAgent: An autonomous AI agent for spatial biology. *bioRxiv*, 2025a. doi: 10.1101/2025.04.03.646459. (Cited on page 19)

Qingyun Wang, Doug Downey, Heng Ji, and Tom Hope. Scimon: Scientific inspiration machines optimized for novelty. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 279–299, 2024. (Cited on pages 1 and 19)

Wenyi Wang, Piotr Piękos, Li Nanbo, Firas Laakom, Yimeng Chen, Mateusz Ostaszewski, Mingchen Zhuge, and Jürgen Schmidhuber. Huxley-Gödel Machine: Human-level coding agent development by an approximation of the optimal self-improving machine. *arXiv preprint arXiv:2510.21614*, 2025b. (Cited on page 20)

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Thetaevolve: Test-time learning on open problems. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2511.23473. (Cited on pages 7, 20 and 35)

Zifeng Wang, Benjamin Danek, and Jimeng Sun. Biodsa-1k: Benchmarking data science agents for biomedical research. *arXiv preprint arXiv:2505.16100*, 2025c. (Cited on page 19)

David P. Woodruff, Vincent Cohen-Addad, Lalit Jain, Jieming Mao, Song Zuo, MohammadHossein Bateni, Simina Brânzei, Michael P. Brenner, Lin Chen, Ying Feng, Lance Fortnow, Gang Fu, Ziyi Guan, Zahra Hadizadeh, Mohammad Taghi Hajiaghayi, Mahdi JafariRaviz, Adel Javanmard, Karthik C. S., Ken ichi Kawarabayashi, Ravi Kumar, Silvio Lattanzi, Euiwoong Lee, Yi Li, Ioannis Panageas, Dimitris Paparas, Benjamin Przybocki, Bernardo Subercaseaux, Ola Svensson, Shayan Taherijam, Xuan Wu, Eylon Yogev, Morteza Zadimoghaddam, Samson Zhou, Yossi Matias, James Manyika, and Vahab S. Mirrokni. Accelerating scientific research with gemini: Case studies and common techniques. *ArXiv*, abs/2602.03837, 2026. (Cited on page 19)

Yunze Wu, Dayuan Fu, Weiye Si, Zhen Huang, Mohan Jiang, Keyu Li, Shijie Xia, Jie Sun, Tianze Xu, Yang Xiao, Pengrui Lu, Xiaojie Cai, Lyumanshan Ye, Wenhong Zhu, Xiangkun Hu, and Pengfei Liu. InnovatorBench: Evaluating agents' ability to conduct innovative AI research. In *ICLR*, 2026. arXiv:2510.27598. (Cited on page 19)

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025. (Cited on pages 1, 3 and 19)

Cheng Yang, Jiaxuan Lu, Haiyuan Wan, Junchi Yu, and Feiwei Qin. From what to why: A multi-agent system for evidence-based chemical reaction condition reasoning. In *ICLR*, 2026a. (Cited on page 19)

Junlin Yang, Che Jiang, Yu Fu, Tianwei Luo, Can Ren, Weizhi Wang, Kaikai Zhao, Hongyi Liu, Yuxin Zuo, Yuru Wang, et al. Frontis-ma1: Training an ai4ai model towards recursive self-improvement in machine learning engineering. *arXiv preprint arXiv:2607.28568*, 2026b. (Cited on pages 2, 8 and 39)

Karren Yang, Abigail Katcoff, and Caroline Uhler. Characterizing and learning equivalence classes of causal dags under interventions. In *International Conference on Machine Learning*, pp. 5541–5550. PMLR, 2018. (Cited on page 20)

Zonglin Yang, Xinya Du, Junxian Li, Jie Zheng, Soujanya Poria, and Erik Cambria. Large language models for automated open-domain scientific hypotheses discovery. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13545–13565, 2024. (Cited on pages 1 and 19)

Zonglin Yang, Wanhao Liu, Ben Gao, Tong Xie, Yuqiang Li, Wanli Ouyang, Soujanya Poria, Erik Cambria, and Dongzhan Zhou. Moose-chem: Large language models for rediscovering unseen chemistry scientific hypotheses. In *ICLR*, 2025. (Cited on page 19)

Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. ReEvo: Large language models as hyper-heuristics with reflective evolution. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 43571–43608, 2024. (Cited on page 3)

Cunxi Yu, Rongjian Liang, Chia-Tung Ho, and Haoxing Ren. Autonomous code evolution meets NP-completeness. *arXiv preprint arXiv:2509.07367*, 2025a. (Cited on page 20)

Zhaojian Yu, Kaiyue Feng, Yilun Zhao, Shilin He, Xiao-Ping Zhang, and Arman Cohan. AlphaResearch: Accelerating new algorithm discovery with language models. *arXiv preprint arXiv:2511.08522*, 2025b. (Cited on page 20)

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time. In *International Conference on Machine Learning (ICML)*, 2026. arXiv:2601.16175. (Cited on page 20)

Jenny Zhang, Shengran Hu, Cong Lu, Robert Tjarko Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *International Conference on Learning Representations (ICLR)*, 2026. arXiv:2505.22954. (Cited on pages 3 and 20)

Sha Zhang, Suorong Yang, Tong Xie, Xiangyuan Xue, Zixuan Hu, Rui Li, Wenxi Qu, Zhenfei Yin, Tianfan Fu, Di Hu, et al. Position: Intelligent science laboratory requires the integration of cognitive and embodied ai. *arXiv preprint arXiv:2506.19613*, 2025. (Cited on page 19)

Tianshi Zheng, Zheye Deng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Zihao Wang, and Yangqiu Song. From automation to autonomy: A survey on large language models in scientific discovery. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17733–17750, 2025a. (Cited on pages 1 and 19)

Tianshi Zheng, Kelvin Kiu-Wai Tam, Newt Hue-Nam K. Nguyen, Baixuan Xu, Zhaowei Wang, Jiayang Cheng, Hong Ting Tsang, Weiqi Wang, Jiaxin Bai, Tianqing Fang, Yangqiu Song, Ginny Y. Wong, and Simon See. NewtonBench: Benchmarking generalizable scientific law discovery in LLM agents. In *ICLR*, 2026. arXiv:2510.07172. (Cited on page 19)

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *International Conference on Machine Learning (ICML)*, 2025b. arXiv:2501.08603. (Cited on page 19)

Lianhao Zhou, Hongyi Ling, Cong Fu, Yepeng Huang, Michael Sun, Wendi Yu, Xiaoxuan Wang, Xiner Li, Xingyu Su, Junkai Zhang, Xiusi Chen, Chenxing Liang, Xiaofeng Qian, Heng Ji, Wei Wang, Marinka Zitnik, and Shuiwang Ji. Autonomous agents for scientific discovery: Orchestrating scientists, language, code, and physics. *ArXiv*, abs/2510.09901, 2025. (Cited on pages 2 and 19)

Qing Zhu, Fei Zhang, Yan Huang, Hengyu Xiao, LuYuan Zhao, XuChun Zhang, Tao Song, XinSheng Tang, Xiang Li, Guo He, et al. An all-round ai-chemist with a scientific mind. *National Science Review*, 9(10):nwac190, 2022. (Cited on page 19)

A Limitations and Future directions

Due to the resource constraints, the tasks studied in this work remain limited to small-scale verification and are relatively simple. There remains a broader scope of scientific problems to be explored where verification signals can be expensive, such as biology, medicine, etc.. Meanwhile, the factor discovery can also be guided with additional background knowledge of the task.

B Related Work

In this section, we present more detailed discussions of related works in the literature.

AI Scientist agents. With the significant advancement in LLM capacity and the development of agentic systems, a rising number of works develop agents to help scientific discovery (Zheng et al., 2025a; Zhou et al., 2025; Tie et al., 2025). Some systems automate the whole research workflow, from idea generation to experimentation and paper writing (Lu et al., 2026; Yamada et al., 2025; Gottweis et al., 2026; Ghareeb et al., 2026; Mitchener et al., 2025; Feng et al., 2026b; Lyu et al., 2026b). One research line automates specific scientific activities, including literature review and deep research (Asai et al., 2026; Huang et al., 2025b), hypothesis generation (Li et al., 2025c; Yang et al., 2024; Wang et al., 2024; Yang et al., 2025), hypothesis verification (Li et al., 2024; Huang et al., 2025a), and feedback on scientific papers (Liang et al., 2024). Another line targets data-driven discovery and machine learning engineering (Majumder et al., 2025; Wang et al., 2025c; Jing et al., 2025; Chan et al., 2025), as well as embodied agents in self-driving laboratories that conduct real-world experiments (Roch et al., 2020; Zhu et al., 2022; Tom et al., 2024; Zhang et al., 2025; Mandal et al., 2025). A growing set of benchmarks evaluates the research abilities of these agents, e.g., replicating and conducting AI research (Starace et al., 2025; Wu et al., 2026; Lyu et al., 2026a), generating hypotheses from inspirations (Liu et al., 2026), discovering scientific laws and equations (Zheng et al., 2026; Shojaee et al., 2025b), and conducting research tasks in biology (Laurent et al., 2024), together with broader evaluations of research abilities (Wan et al., 2026). These lines of work have made impact in mathematics (OpenAI, 2026c;b; Bubeck et al., 2025; Woodruff et al., 2026), chemistry (Yang et al., 2026a; Boiko et al., 2023), materials science (Ghafarollahi & Buehler, 2025), earth science (Feng et al., 2026a), biology (Swanson et al., 2025; Wang et al., 2025a; Truhn et al., 2026; Anthropic, 2026), and the development of LLMs themselves (Karpathy, 2026; OpenAI, 2026a; Favaro & Clark, 2026). As generation scales up, validating the generated hypotheses becomes the main cost (Wallace et al., 2026), and ideas that look promising often degrade once executed (Si et al., 2026), which motivates us to study how to reuse experience to reduce the evaluations needed in evolutionary discovery.

Evolutionary coding agents. FunSearch (Romera-Paredes et al., 2024) pairs an LLM with an evaluator and a program database to discover new constructions in extremal combinatorics, and `AlphaEvolve` (Novikov et al., 2025) extends the loop to full codebases with an ensemble of LLMs, with applications to a large set of mathematical problems (Georgiev et al., 2025). Follow-up frameworks improve the search process: OpenEvolve provides an open-source implementation (Sharma, 2025); `ShinkaEvolve` improves sample efficiency with parent sampling, novelty rejection, bandit-based LLM selection, and a meta-scratchpad that summarizes insights of recent programs (Lange et al., 2026); CodeEvolve and GigaEvo maintain diversity with island models and quality-diversity archives (Assumpção et al., 2025; Khrulkov et al., 2025); MCTS-AHD organizes heuristics in a Monte Carlo search tree (Zheng et al., 2025b); and AdaEvolve adapts search intensity and resource allocation from the history of improvements (Cemri et al., 2026). The paradigm has also been applied to equation discovery (Shojaee et al., 2025a) and systems research (Cheng et al., 2025). `SCS` shows that simply conditioning on previous solutions is competitive with more elaborate pipelines (Gideoni et al., 2026), which we adopt as the backbone and main baseline. Classical evolutionary computation also reuses experience through credit assignment over predefined units: adaptive operator selection allocates budget across operators with multi-armed bandits (Da Costa et al., 2008), quality-diversity methods maintain elites across behavior niches (Mouret & Clune, 2015) or search for novelty alone (Lehman & Stanley, 2011), and MuEvo revises the priorities of heuristic components with short-budget probing (Lv et al., 2026). In `ExpEvolve`, the units receiving credit are ideas discovered during the search, and plateau handling targets the ideas shared by top programs.

Learned mutators and self-improving agents. Another line trains the LLM mutator with reinforcement learning, either during evolution (Šurina et al., 2025) or at test time on a single problem (Wang et al., 2026; Yuksekgonul et al., 2026), where experience is stored in model weights. Self-improving agents such as the Darwin Gödel Machine and the Huxley-Gödel Machine evolve the code of the agent itself (Zhang et al., 2026; Wang et al., 2025b), SATLUTION evolves a solver repository together with its own rulebase (Yu et al., 2025a), and AlphaResearch alternates between idea proposal and program verification (Yu et al., 2025b). These directions are orthogonal to ours, which uses a fixed LLM and a fixed harness.

Causality for scientific discovery. There is a long history of discussions on how to understand the world through observations (Greenland et al., 1999; Spirtes et al., 2000; Pearl, 2009). One research line is causal discovery, which learns the causal structure among variables from observational data under assumptions such as conditional independence or constrained functional forms (Spirtes et al., 1995; Shimizu et al., 2006; Hoyer et al., 2008), from interventional data and data of multiple domains (Yang et al., 2018; Brouillard et al., 2020; Mooij et al., 2020; Huang et al., 2020; Perry et al., 2022), and in the presence of latent variables (Li et al., 2025d; Li & Liu, 2025), while experiment design selects the interventions that make the causal structure identifiable (Hyttinen et al., 2013; Li et al., 2025b). Recently, LLMs have been used to construct structured variables out of unstructured data (Liu et al., 2025; Li et al., 2025a), to estimate treatment effects from unstructured data (Dhawan et al., 2024), and to propose instrumental variables (Sheth et al., 2025). Compared with these works, the variables in `ExpEvolve`, i.e., ideas, are constructed from unstructured programs, and are evaluated through interventions that the search itself chooses adaptively; Sec. 3.2 characterizes what such an archive can identify.

C A Causal View of the Formulation

In this section, we present a more detailed discussion of the causal view of evolutionary discovery in Sec. 3.1.

C.1 Formulation of Scientific Discovery

Scientific discovery aims to uncover the underlying scientific knowledge or the causal mechanisms from interactions with the world (Kuhn & Hawkins, 1963), which can be formulated as a Partially Observed Markov Decision Process (POMDP) (Kaelbling et al., 1998) (Fig. 10).

Scientific knowledge. The primary objective of an AI Scientist is to uncover the *underlying scientific knowledge* about the task-world, represented by a latent variable $\Theta_{\text{sci}} \in \Theta$, where Θ may encode causal structure, mechanisms, inductive biases, constraints, etc. Specifically, $\Theta_{\text{sci}} = \theta_{\text{sci}}$ can be parameterized as a Structural Causal Model (SCM) $\theta_{\text{sci}} = (\mathcal{G}, \mathcal{F}, P_U)$ (Spirtes et al., 2000), where $\mathcal{G} = (V, E)$ is a directed graph whose nodes V represent variables of interest and whose edges E encode direct causal dependencies; $\mathcal{F} = \{f_v\}_{v \in V}$ is a collection of structural equations $v = f_v(\text{Pa}(v), u_v)$, where $\text{Pa}(v)$ denotes the parents of v in $\mathcal{G}$ and u_v is an exogenous noise variable; P_U is a distribution over the exogenous variables $U = \{u_v\}_{v \in V}$.

Evaluation as intervention on SCM. Given the SCM parametrization of θ_{sci}, we can consider that a program $p \in \mathcal{P}$ is encoded as a particular configuration of design variables $X = x_p$. Then, F can be implemented as

$$F(p; \theta_{\text{sci}}) := \mathbb{E}\big[\, Y \,\big|\, \text{do}(X = x_p),\, \theta_{\text{sci}} \big],$$

i.e., the expected outcome under the intervention $\text{do}(X = x_p)$ in the true causal model θ_{sci}. Typical implementations of F can be the objective value of a combinatorial optimization problem, the efficiency of a kernel program, or the performance of a machine learning model (Novikov et al., 2025).

Belief as revealed knowledge. As in Sec. 3.1, the AI Scientist maintains a memory $m_t = \Phi(h_t)$ as an approximate belief about θ_{sci}. Each evaluation step (p_t, y_t) thus updates m_t, which in turn updates the agent's effective belief about θ_{sci}. In this sense, each step *reveals part of the underlying scientific knowledge*, which in turn determines the next action p_{t+1}.

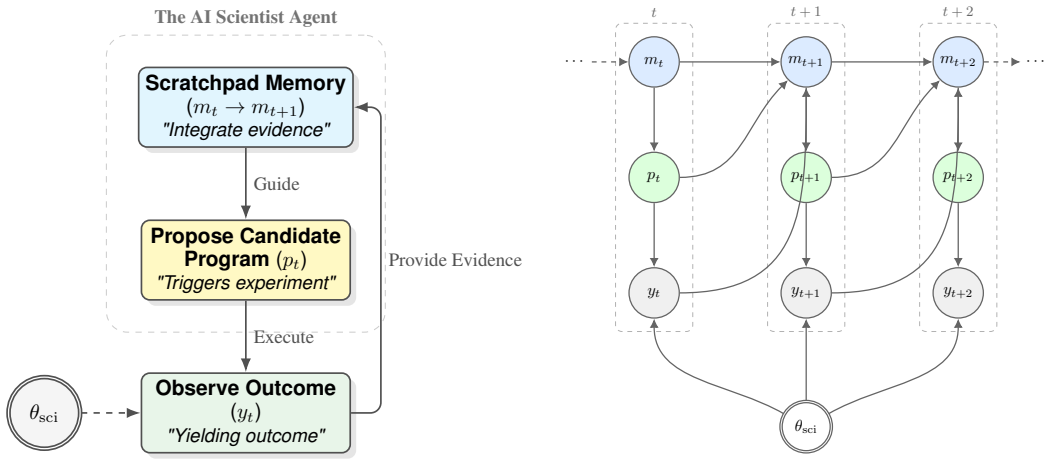


Figure 10: The iterative scientific discovery loop. **Left:** Conceptual flow of the agent. The agent maintains a scratchpad memory (m), proposes a program (p), and observes the outcome (y) which is constrained by the unknown world state (θ_{sci}). The outcome feeds back into the memory for the next step. **Right:** The diagram illustrates how the AI Scientist probes the unknown world state θ_{sci}. By proposing a candidate program p_t, the agent triggers an experiment yielding outcome y_t. This observation provides evidence about θ_{sci}, which is integrated into the agent's scratchpad memory m_{t+1}. Over time steps $t, t+1, \ldots$, this recurrent process allows the agent to navigate the performance landscape and converge towards optimal programs despite the static but unknown nature of θ_{sci}.

C.2 Ideas and Causal Representation Learning

Ideas as high-level causal variables. In the SCM view above, a program specifies a configuration x_p of design variables X, which are low-level and high-dimensional, e.g., every line of the code. An idea z (Definition 3.2) is a function of the program, and hence of x_p, that summarizes the design variables at a higher level. A faithful idea representation (Sec. 3.2) requires that $\mathbb{E}[Y \mid \text{do}(X = x_p), \theta_{\text{sci}}]$ depends on x_p only through $Z(p)$, i.e., interventions on the design variables that keep all ideas fixed leave the expected outcome unchanged. Hence, an intervention on the ideas is well defined at the level of expected outcomes. This condition is related to causal feature learning (Chalupka et al., 2015), which constructs macro-level causal variables by grouping the micro-level states that induce the same outcome distribution under intervention, and to causal abstraction (Rubenstein et al., 2017), which studies when a high-level causal model is consistent with a low-level one. Faithfulness is a mean-level version of these conditions, restricted to the score of interest.

Relation to causal representation learning. Causal representation learning (CRL) aims to recover high-level causal variables from low-level observations (Schölkopf et al., 2021), where identifiability usually relies on additional information such as interventions on the latent variables (Ahuja et al., 2023). Ideas can be viewed as the high-level causal variables of programs, and the two settings differ in three aspects. (i) In CRL, the representation is usually an encoder learned from the observations, while an idea is an interpretable function of the program that an LLM can annotate, similar to how LLMs construct high-level variables from unstructured data (Liu et al., 2025; Li et al., 2025a). (ii) CRL usually aims to recover all latent variables, while ideas only need to be faithful to the score. (iii) The agent decides which programs to evaluate, and an edit that changes one idea and keeps the others fixed acts as an intervention on that idea, although an LLM edit may also change other ideas. Theorem 3.3 shows that when the evaluated programs do not vary the candidate ideas, the ideas that matter cannot be identified from the scores.

C.3 Essentiality of Causal Knowledge for AI Scientists

If the evaluation revealed the scientific knowledge completely, then with more experiment turns, the agent's revealed scientific knowledge would converge to θ_{sci}. However, the observation from the evaluation is usually only a proxy of the scientific knowledge Θ_{sci}: each evaluation returns a

scalar score of the program that the agent chooses to evaluate. Different hypotheses about θ_{sci}, e.g., ones under which different ideas determine the score, can produce the same scores on all evaluated programs. For example, if all evaluated programs in circle packing share the same initialization and the same refinement, the scores cannot tell which of the two accounts for the performance. Therefore, the causal knowledge often cannot be identified from the scores of the programs explored so far, and it is essential for an AI Scientist to explore sufficiently, i.e., to evaluate programs that distinguish the competing hypotheses. We denote by e_{src} the environment in which the evaluation is conducted.

Definition C.1 (Causal AI Scientist)**.** *A* Causal AI Scientist *is an agent specified by: (i) a policy* $\pi_t(\cdot \mid \theta_t, e_{\text{src}})$ *selecting* p_t*, (ii) a counterfactual / explanatory operator* CF*, that answer interventional queries* (e, p) *via* $\text{CF}(\theta_t; e, p)$ *as an "explanation" of predicted performance, where* θ_t *is the knowledge revealed at turn* t*.*

Without the revealing of the causal knowledge, the discovery process suffers from significant inefficiency and suboptimality issues. We discuss the two issues more concretely below.

Evolutionary efficiency of Causal AI Scientist. The efficiency issue is characterized by Theorem 3.1. In the linear setting of Theorem 3.1, a Causal AI Scientist can be implemented via estimating the $w^\star$ and optimizing for $\hat{p}$ from the history, while the black-box baseline does not consider the interactions between the historical observations. Theorem 3.1 shows that, when $K \gg d$, which is usually the case as the space for all programs is significantly larger than the underlying SCM, encoding (correct) causal structure, e.g., a faithful idea representation, yields an exponential (or at least multiplicative) gain in sample efficiency under finite budgets. For example, if K grows exponentially with d, e.g., $K = 2^d$, the upper bound in Theorem 3.1 grows polynomially in d, as $O\big(d\,(d + \log \frac{1}{\delta})\frac{\sigma^2}{\epsilon^2}\big)$, while the lower bound for the black-box baseline grows exponentially in d, as $\Omega\big(2^d \log \frac{1}{\delta}\frac{\sigma^2}{\epsilon^2}\big)$.

Sufficient exploration of Causal AI Scientist. The suboptimality issue is characterized by Theorem 3.3. Let $S \subseteq \mathcal{P}$ be the set of programs evaluated by the agent. Two hypotheses θ_0 and θ_1 can be distinguished from the history only if their observation models differ on some program in S; otherwise, the history has the same distribution under both hypotheses. Under the idea-level mechanisms in Theorem 3.3, i.e., $F(p; \theta_i) = g(Z(p), z_i(p))$ for $i \in \{0, 1\}$, the two hypotheses agree on every program where z_0 and z_1 agree, so only the programs where the two ideas differ can distinguish them. Intuitively, Theorem 3.3 implies that if the evaluated programs do not distinguish the corresponding θ_{sci} among $\{\theta_0, \theta_1\}$, then the solution $\hat{p}$ incurs an expected regret of at least $\Delta/2$ under one of them, regardless of the budget. This situation is common in evolutionary search: as the search concentrates around its current best programs, the ideas shared by these programs stop varying among the evaluated programs, and the scores cannot tell whether these ideas are the ones that matter. Hence, a Causal AI Scientist needs to explore the programs in which the competing ideas differ, so that the scores can distinguish the hypotheses. This barrier can be removed by exploration within the same evaluator, which differs from the transfer across environments discussed below.

C.4 Transfer of Scientific Knowledge across Environments

Beyond a single evaluator, the revealed scientific knowledge is also expected to transfer to other environments. If the objective function F is static universally, then with more experiment turns, the optimized solution p_t and the agent's revealed scientific knowledge can also be applied universally. However, the evaluation is usually conducted at some specific environment $e \in \mathcal{E}$, which only gives a proxy knowledge θ_e about the scientific knowledge Θ_{sci}. For example, the performance of a machine learning model is usually assessed on finite samples from the test distribution, and there also exist distribution shifts from the test distribution when deploying the model in the real world (Quinonero-Candela et al., 2008). Different from Θ_{sci} that characterizes the complete causal structure about the scientific problem, optimization under environment θ_e may introduce some spurious correlations that maximize the objective value F_e (Chen et al., 2023). Therefore, to retain the optimality of $\hat{p}$ beyond the source environment e_{src} to some target e_{tgt}, it is essential to reveal the causal knowledge and answer causal questions for an AI Scientist, e.g., through the operator CF in Definition C.1. Let $P_e(\cdot \mid p, \theta)$ denote the observation model of the evaluator in environment e, with mean $F_e(p; \theta)$. We have the following:

Theorem C.2. *Consider* $e_{\text{src}}, e_{\text{tgt}} \in \mathcal{E}$ *and* $\theta_0, \theta_1 \in \Theta$ *such that* $P_{e_{\text{src}}}(\cdot \mid p, \theta_0) = P_{e_{\text{src}}}(\cdot \mid p, \theta_1)$ *for all* $p \in \mathcal{P}$*, and the optimal programs in* e_{tgt} *under* θ_0 *and* θ_1 *are different and unique, each better*

than all other programs by at least $\Delta > 0$. Then for any policy π that can interact only with e_{src}, there exists $i \in \{0, 1\}$ such that for every budget T, $\mathbb{E}\big[\max_{p \in \mathcal{P}} F_{e_{\text{tgt}}}(p; \theta_i) - F_{e_{\text{tgt}}}(\hat{p}; \theta_i)\big] \geq \Delta/2$.

The formal description and the proof are given in Appendix G. Intuitively, Theorem C.2 implies that if the source environment does not distinguish the corresponding θ_{sci} among $\{\theta_0, \theta_1\}$, then the solution $\hat{p}$ solved given source environment is suboptimal in the target environment under one of them, no matter how many evaluations are made in the source environment. In the real world, it is usually the case that two machine learning models will have similar performances under the public test benchmarks, but exhibit significantly different behaviors when generalizing to distributions from other environments. Different from Theorem 3.3, this barrier cannot be removed by more exploration in the source environment, since the two hypotheses agree on every program there. We leave a full study of the transfer of scientific knowledge across environments to future work.

D Additional Technical Details

D.1 Notation

Table 2: Notation used in the formulation and theorems.

Symbol	Meaning
$\mathcal{P}$	Program / pipeline / model space (candidate designs)
K	Number of candidate programs, $K := \lvert\mathcal{P}\rvert$ (finite in Theorem 3.1)
$p \in \mathcal{P}$	A program to evaluate (action)
$\mathcal{X}$	Design-variable space (encoding of programs)
$x_p \in \mathcal{X}$	Encoding of program p (e.g., design variables $X = x_p$)
z, $Z(p)$	An idea and the idea representation of program p (Definition 3.2)
r	Number of ideas in an idea representation
$\phi(p) \in \mathbb{R}^d$	Known representation of program p in Theorem 3.1, e.g., the one-hot encoding of $Z(p)$
d	Dimension of the representation, e.g., the number of value combinations of the ideas
Θ	Hypothesis space of scientific knowledge (e.g., SCMs / mechanisms)
Θ_{sci}	Latent RV taking values in Θ (Bayesian view)
$\theta^\star \in \Theta$	True (fixed but unknown) scientific knowledge instance (realization)
μ_0	Prior over Θ (i.e., $\Theta_{\text{sci}} \sim \mu_0$)
$\mathcal{E}$	Environment / protocol index set (evaluation regimes, deployments)
$e \in \mathcal{E}$	Environment index; e_{src} source, e_{tgt} target
$F_e(p; \theta)$	True performance in env e (scalar objective)
$P_e(\cdot \mid p, \theta)$	Observation model (likelihood) for evaluator output in env e
y_t	Observed evaluator outcome at round t
h_t	History $\{(p_0, y_0), \ldots, (p_{t-1}, y_{t-1})\}$
b_t	Bayesian belief/posterior over θ: $b_t(\cdot) = \Pr(\Theta_{\text{sci}} \in \cdot \mid h_t, e_{\text{src}})$
T	Evaluation budget / horizon (number of program evaluations)

D.2 Random variable, space, and realization (to avoid notation confusion)

We use the following (standard) convention.

(i) Hypothesis space. Θ is a set that contains all candidate scientific-knowledge hypotheses.

(ii) True but unknown instance. The real world is governed by a fixed but unknown $\theta^\star \in \Theta$.

(iii) Bayesian view (optional but convenient). A Bayesian agent models uncertainty by treating $\theta^\star$ as a realization of a latent random variable Θ_{sci} with prior μ_0, i.e. $\Theta_{\text{sci}} \sim \mu_0$ and $\theta^\star$ is one draw from it. The belief b_t is simply the posterior distribution of Θ_{sci} after seeing history h_t.

(iv) Does scientific knowledge change across environments? In our formulation, the *underlying* scientific knowledge $\theta^\star$ is static across rounds. Different environments $e \in \mathcal{E}$ represent different evaluation/deployment protocols (distribution shifts, constraint changes, measurement noise, private vs public tests, etc.). Formally, environments affect either the true performance map $F_e(\cdot; \theta)$ and/or the observation kernel $P_e(\cdot \mid p, \theta)$, while $\theta^\star$ itself remains the same hidden instance.

D.3 Evaluator as an observation model (covers deterministic and stochastic evaluators)

Fix an environment $e \in \mathcal{E}$. When the agent evaluates program p, it receives an observation $y \in \mathcal{Y}$ drawn from

$$y \sim P_e(\cdot \mid p, \theta^\star),$$

where $P_e(\cdot \mid p, \theta)$ is a conditional distribution on $\mathcal{Y}$.

Deterministic evaluator. A deterministic evaluator is the special case where there exists a function g_e such that

$$P_e(\cdot \mid p, \theta) = \delta_{g_e(p;\theta)}(\cdot), \quad \text{i.e.,} \quad y = g_e(p; \theta^\star) \text{ a.s.}$$

In many program-evolution settings, the evaluator is designed to deterministically check validity and compute an objective score (e.g., via a verifier and a scoring routine).

Stochastic/noisy evaluator. A common instantiation is additive noise:

$$y = F_e(p; \theta^\star) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

but only the proof of Theorem 3.1 relies on the Gaussian form; the other results hold for any noise distribution shared by the hypotheses.

D.4 Belief and Bayes update: kernel form and undergraduate-friendly special cases

Let $h_t = \{(p_0, y_0), \ldots, (p_{t-1}, y_{t-1})\}$ be the history. The Bayesian belief (posterior) is

$$b_t(B) = \Pr(\Theta_{\text{sci}} \in B \mid h_t, e_{\text{src}}), \qquad B \subseteq \Theta.$$

General Bayes update (kernel form). After choosing p_t and observing y_t in e_{src}, the posterior is

$$b_{t+1}(B) = \frac{\int_B P_{e_{\text{src}}}(dy_t \mid p_t, \theta)\, b_t(d\theta)}{\int_\Theta P_{e_{\text{src}}}(dy_t \mid p_t, \theta)\, b_t(d\theta)}. \tag{2}$$

Finite hypothesis space (sum form). If $\Theta = \{\theta_1, \ldots, \theta_N\}$ is finite and the likelihood has a pmf $P_{e_{\text{src}}}(y_t \mid p_t, \theta_i)$, then

$$b_{t+1}(\theta_i) = \frac{b_t(\theta_i)\, P_{e_{\text{src}}}(y_t \mid p_t, \theta_i)}{\sum_{j=1}^N b_t(\theta_j)\, P_{e_{\text{src}}}(y_t \mid p_t, \theta_j)}.$$

Continuous hypothesis space (density form). If $P_{e_{\text{src}}}(dy \mid p, \theta)$ has a density $p_{e_{\text{src}}}(y \mid p, \theta)$, then

$$b_{t+1}(\theta) = \frac{b_t(\theta)\, p_{e_{\text{src}}}(y_t \mid p_t, \theta)}{\int_\Theta b_t(\theta')\, p_{e_{\text{src}}}(y_t \mid p_t, \theta')\, d\theta'}.$$

Deterministic evaluator (indicator/filter form). If $y = g_{e_{\text{src}}}(p; \theta)$ deterministically, then the update becomes

$$b_{t+1}(d\theta) \;\propto\; \mathbf{1}\{g_{e_{\text{src}}}(p_t; \theta) = y_t\}\, b_t(d\theta),$$

i.e. the posterior is the prior restricted to hypotheses consistent with the observed outcome.

E Proof of Theorem 3.1 (Structured experience reuse)

Throughout this section we fix a *single* static environment (drop e from notation), and assume $\mathcal{P} = \{p_1, \ldots, p_K\}$ is finite.

E.1 Protocol and performance criterion

Experiment–then–commit protocol. A policy π interacts for T rounds. At each round $t = 0, \ldots, T-1$ it selects a program $p_t \in \mathcal{P}$ (possibly randomized) based on the past history h_t, then observes $y_t \in \mathbb{R}$. After T evaluations it outputs a final recommendation $\hat{p} \in \mathcal{P}$.

Simple regret. Let $f(p)$ denote the true mean performance of program p in this environment. Define the (random) simple regret

$$\mathrm{SR}_T := \max_{p \in \mathcal{P}} f(p) \; - \; f(\hat{p}). \tag{3}$$

(ϵ, δ)-correctness (uniform). Fix $\epsilon > 0$ and $\delta \in (0, 1)$. We say a policy π is (ϵ, δ)-*correct uniformly on a hypothesis class* $\mathcal{H}$ if for every instance in $\mathcal{H}$,

$$\Pr\left(\mathrm{SR}_T \le \epsilon\right) \ge 1 - \delta.$$

"Uniformly" means the guarantee must hold for *all* instances in the class, not only on average.

E.2 Two hypothesis classes

(1) Structured class. Each program p has a known representation $\phi(p) \in \mathbb{R}^d$ with $\|\phi(p)\|_2 \le 1$, e.g., a faithful linear idea representation $\phi(p) = Z(p)$ (Sec. 3.2). The unknown instance is a weight vector $w^\star \in \mathbb{R}^d$ and

$$f(p) = \langle \phi(p), w^\star \rangle. \tag{4}$$

Observations follow a Gaussian noise model

$$y_t = f(p_t) + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \text{ i.i.d.} \tag{5}$$

Coverage condition. Assume there exist d basis programs $p^{(1)}, \ldots, p^{(d)}$ whose representations are the standard basis:

$$\phi(p^{(i)}) = e_i, \qquad i = 1, \ldots, d. \tag{6}$$

(2) Unstructured black-box class (baseline). The unknown instance is an arbitrary vector of means

$$\mu = (\mu_1, \ldots, \mu_K) \in \mathbb{R}^K, \qquad f(p_i) = \mu_i,$$

and observations are

$$y_t = \mu_{I_t} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \text{ i.i.d.}, \tag{7}$$

where $I_t \in \{1, \ldots, K\}$ is the index of the chosen program $p_t = p_{I_t}$. Crucially, there is *no assumed relation* between μ_i and μ_j for $i \ne j$.

E.3 Formal statement and proof

Theorem E.1 (Formal version of Theorem 3.1)**.** *Fix $\epsilon > 0$ and $\delta \in (0, 1/4)$.*

1. ***(Upper bound under the structured linear class).*** *Under equation 4–equation 6 and equation 5, there exists a policy π_{lin} such that*

$$\Pr\left(\mathrm{SR}_T \le 2\epsilon\right) \ge 1 - \delta \quad \textit{whenever} \quad T \; \ge \; 2d\,\frac{\sigma^2}{\epsilon^2}\log\!\Big(\frac{2K}{\delta}\Big).$$

2. ***(Lower bound for the unstructured black-box class).*** *For the black-box class equation 7, any policy that is (ϵ, δ)-correct* uniformly for all $\mu \in \mathbb{R}^K$ *must satisfy*

$$T \; \ge \; (K-1)\,\frac{\sigma^2}{8\epsilon^2}\,\log\!\Big(\frac{1}{2\delta}\Big).$$

Replacing ϵ with $\epsilon/2$ in part (1) gives the informal statement in Theorem 3.1.

Proof. We prove the two parts separately.

Part (1): constructive upper bound (estimate $w^\star$ then commit). Evaluate each basis program $p^{(i)}$ exactly n times (total $T = nd$). Let $y_1^{(i)}, \dots, y_n^{(i)}$ be the observations for basis i, and define

$$\hat{w}_i := \frac{1}{n} \sum_{j=1}^{n} y_j^{(i)}.$$

By equation 4–equation 6, $f(p^{(i)}) = w_i^\star$. By equation 5, $\hat{w}_i \sim \mathcal{N}(w_i^\star, \sigma^2/n)$ and these coordinates are independent.

Define for any program p:

$$\widehat{f}(p) := \langle \phi(p), \hat{w} \rangle, \qquad \hat{w} = (\hat{w}_1, \dots, \hat{w}_d).$$

Then

$$\widehat{f}(p) - f(p) = \langle \phi(p), \hat{w} - w^\star \rangle \sim \mathcal{N}\Big(0,\ \frac{\sigma^2}{n} \|\phi(p)\|_2^2\Big),$$

so since $\|\phi(p)\|_2 \le 1$,

$$\Pr\left(|\widehat{f}(p) - f(p)| \ge \epsilon\right) \le 2 \exp\Big(-\frac{n\epsilon^2}{2\sigma^2}\Big).$$

Union bound over K programs gives

$$\Pr\left(\max_{p \in \mathcal{P}} |\widehat{f}(p) - f(p)| \ge \epsilon\right) \le 2K \exp\Big(-\frac{n\epsilon^2}{2\sigma^2}\Big).$$

Choose

$$n \ \ge\ 2\frac{\sigma^2}{\epsilon^2} \log\Big(\frac{2K}{\delta}\Big),$$

so that with probability at least $1 - \delta$ we have $\max_p |\widehat{f}(p) - f(p)| \le \epsilon$.

Now output $\hat{p} := \arg\max_{p \in \mathcal{P}} \widehat{f}(p)$. Let $p^\star := \arg\max_p f(p)$. On the above high-probability event,

$$f(p^\star) - f(\hat{p}) \le \big(f(p^\star) - \widehat{f}(p^\star)\big) + \big(\widehat{f}(\hat{p}) - f(\hat{p})\big) \le \epsilon + \epsilon = 2\epsilon.$$

Thus $\Pr(\mathrm{SR}_T \le 2\epsilon) \ge 1 - \delta$ for $T = nd$ as stated.

Part (2): lower bound for the black-box class. We construct K hard instances and lower bound any uniformly (ϵ, δ)-correct policy.

Let the programs be $p_1, \dots, p_K$. Define a base instance $\mu^{(0)} \in \mathbb{R}^K$:

$$\mu_1^{(0)} = 0, \qquad \mu_i^{(0)} = -2\epsilon \ \ (i = 2, \dots, K).$$

For each $i \in \{2, \dots, K\}$, define an alternative instance $\mu^{(i)}$:

$$\mu_1^{(i)} = 0, \qquad \mu_i^{(i)} = +2\epsilon, \qquad \mu_j^{(i)} = -2\epsilon \ \ (j \notin \{1, i\}).$$

Under $\mu^{(0)}$, the unique best program is p_1, and choosing any p_i with $i \ge 2$ incurs regret $2\epsilon > \epsilon$. Under $\mu^{(i)}$, the unique best program is p_i, and choosing p_1 incurs regret $2\epsilon > \epsilon$.

Let P_0 be the distribution of the full transcript $\mathcal{T} := (p_{0:T-1}, y_{0:T-1}, \hat{p})$ under $\mu^{(0)}$, and P_i the analogous distribution under $\mu^{(i)}$. Uniform (ϵ, δ)-correctness implies

$$P_0(\hat{p} = p_1) \ge 1 - \delta, \qquad P_i(\hat{p} = p_1) \le \delta \quad (i = 2, \dots, K).$$

Step 1: a KL lower bound from an event. For any event A and distributions P, Q, one has

$$\mathrm{KL}(P\|Q) \ge P(A) \log \frac{P(A)}{Q(A)} + (1 - P(A)) \log \frac{1 - P(A)}{1 - Q(A)}.$$

Apply it with $A = \{\hat{p} = p_1\}$, $P = P_0$, $Q = P_i$. Let $p := P_0(A) \ge 1 - \delta$ and $q := P_i(A) \le \delta$. For $\delta \in (0, 1/4)$ this yields

$$\mathrm{KL}(P_0\|P_i) \ \ge\ \log\Big(\frac{1}{2\delta}\Big). \tag{8}$$

Step 2: compute $\mathrm{KL}(P_0\|P_i)$ via number of pulls of arm i. Under $\mu^{(0)}$ and $\mu^{(i)}$, the policy is identical; only observations when playing p_i differ:

$$y \sim \mathcal{N}(-2\epsilon, \sigma^2) \text{ under } \mu^{(0)}, \qquad y \sim \mathcal{N}(+2\epsilon, \sigma^2) \text{ under } \mu^{(i)}.$$

For Gaussians with equal variance, $\mathrm{KL}(\mathcal{N}(m_0, \sigma^2)\|\mathcal{N}(m_1, \sigma^2)) = \frac{(m_0-m_1)^2}{2\sigma^2}$, so each pull of p_i contributes KL $\frac{(4\epsilon)^2}{2\sigma^2} = \frac{8\epsilon^2}{\sigma^2}$.

Let N_i be the (random) number of times p_i is evaluated in T rounds. Additivity of log-likelihood ratios over independent Gaussian samples yields

$$\mathrm{KL}(P_0\|P_i) = \frac{8\epsilon^2}{\sigma^2}\,\mathbb{E}_{P_0}[N_i]. \tag{9}$$

Step 3: conclude the lower bound on T. Combine equation 8 and equation 9:

$$\mathbb{E}_{P_0}[N_i] \;\geq\; \frac{\sigma^2}{8\epsilon^2}\log\Big(\frac{1}{2\delta}\Big), \qquad i = 2, \ldots, K.$$

Summing over $i = 2, \ldots, K$ gives

$$T = \sum_{i=1}^{K} N_i \;\geq\; \sum_{i=2}^{K} \mathbb{E}_{P_0}[N_i] \;\geq\; (K-1)\frac{\sigma^2}{8\epsilon^2}\log\Big(\frac{1}{2\delta}\Big).$$

This completes the proof. □

Remark (deterministic evaluator). If $\sigma = 0$, the structured linear class can recover $w^\star$ exactly from d basis evaluations and achieve $\mathrm{SR}_T = 0$, while in the unstructured black-box class a uniform worst-case guarantee requires evaluating all K programs at least once.

Relation to best-arm identification. The lower bound follows the standard change-of-measure argument for best-arm identification in K-armed Gaussian bandits (Kaufmann et al., 2016), and the upper bound is a simple instance of best-arm identification in linear bandits (Soare et al., 2014).

F Proof of Theorem 3.3 (Degenerated exploration of ideas)

F.1 Formal statement and proof

For a policy that interacts for T rounds and outputs $\hat{p}$, define the simple regret under mechanism θ as $\mathrm{SR}_T(\theta) := \max_{p\in\mathcal{P}} F(p;\theta) - F(\hat{p};\theta)$.

Theorem F.1 (Formal version of Theorem 3.3)**.** *Let $z_0, z_1 : \mathcal{P} \to \mathcal{V}$ be two ideas, and let $A := \{p \in \mathcal{P} : z_0(p) = z_1(p)\}$ be the set of programs on which they agree. Assume there exist $\theta_0, \theta_1 \in \Theta$ such that:*

- *(i) **(Idea-level mechanisms)** $F(p;\theta_i) = g(Z(p), z_i(p))$ for all $p \in \mathcal{P}$ and $i \in \{0,1\}$, for a common representation Z and function g, and each evaluation returns $y = F(p;\theta_i) + \varepsilon$ with the same noise distribution under θ_0 and θ_1;*
- *(ii) **(Optimal program flips with margin Δ)** there exist $p_0 \neq p_1$ and $\Delta > 0$ such that $F(p_i;\theta_i) - F(p;\theta_i) \geq \Delta$ for all $p \neq p_i$ and $i \in \{0,1\}$.*

Then for any policy that only evaluates programs in A, there exists $i \in \{0,1\}$ such that for every budget T,

$$\mathbb{E}\big[\mathrm{SR}_T(\theta_i)\big] \;\geq\; \Delta/2.$$

Moreover, at least one of p_0 and p_1 lies outside A.

Proof. For every $p \in A$, $z_0(p) = z_1(p)$ gives $F(p;\theta_0) = g(Z(p), z_0(p)) = g(Z(p), z_1(p)) = F(p;\theta_1)$. Together with the common noise distribution in (i), the observation model $O(\cdot \mid \theta_i, p)$ is identical under θ_0 and θ_1 for every $p \in A$.

Let $\mathbb{P}_i$ be the distribution of the transcript $(p_{0:T-1}, y_{0:T-1}, \hat{p})$ when the true mechanism is θ_i. Since the policy only evaluates programs in A, for any history and any chosen $p_t \in A$, the conditional distribution of y_t is identical under θ_0 and θ_1, and the policy chooses p_{t+1} and $\hat{p}$ with the same randomization given the same history. By induction on t, $\mathbb{P}_0 = \mathbb{P}_1$. In particular, the output $\hat{p}$ has the same distribution Q on $\mathcal{P}$ under both mechanisms.

By (ii), any output $\hat{p} \neq p_i$ incurs regret at least Δ under θ_i, so

$$\mathbb{E}[\mathrm{SR}_T(\theta_0)] \;\geq\; \Delta\,\big(1 - Q(\hat{p} = p_0)\big), \qquad \mathbb{E}[\mathrm{SR}_T(\theta_1)] \;\geq\; \Delta\,\big(1 - Q(\hat{p} = p_1)\big).$$

Since $p_0 \neq p_1$, $Q(\hat{p} = p_0) + Q(\hat{p} = p_1) \leq 1$, so at least one of the two probabilities is at most $1/2$, and the corresponding expected regret is at least $\Delta/2$.

For the last claim, suppose both $p_0, p_1 \in A$. Then $F(p_1; \theta_0) = F(p_1; \theta_1) \geq F(p_0; \theta_1) + \Delta = F(p_0; \theta_0) + \Delta$, which contradicts the optimality of p_0 under θ_0. □

The theorem holds for any budget, since the barrier comes from the programs that the policy evaluates. To identify which of z_0 and z_1 matters, the policy has to evaluate a program outside A, i.e., a program on which the two ideas differ. In `ExpEvolve`, this is the role of plateau-aware revision, which changes the proposal distribution to evaluate programs where the ideas shared by the leading programs differ.

G Proof of Theorem C.2 (Non-identifiability under environment shifts)

G.1 Setup: source interaction, target evaluation, and target regret

The agent can only interact with the *source* environment e_{src}:

$$y_t \sim P_{e_{\mathrm{src}}}(\cdot \mid p_t, \theta^\star).$$

After T rounds it outputs a final program $\hat{p}$. Performance is judged in a *target* environment e_{tgt} via $F_{e_{\mathrm{tgt}}}(p; \theta^\star)$. Define the target (simple) regret:

$$\mathrm{GR}_T(\theta^\star) := \max_{p \in \mathcal{P}} F_{e_{\mathrm{tgt}}}(p; \theta^\star) - F_{e_{\mathrm{tgt}}}(\hat{p}; \theta^\star).$$

G.2 Formal statement and proof

Theorem G.1 (Non-identifiability barrier under shifts)**.** *Fix $e_{\mathrm{src}}, e_{\mathrm{tgt}} \in \mathcal{E}$. Assume there exist two hypotheses $\theta_0, \theta_1 \in \Theta$ such that:*

$$\textbf{\textit{(Source indistinguishability)}} \quad P_{e_{\mathrm{src}}}(\cdot \mid p, \theta_0) = P_{e_{\mathrm{src}}}(\cdot \mid p, \theta_1), \quad \forall p \in \mathcal{P}. \tag{10}$$

$$\textbf{\textit{(Target optimal action flips with margin } }\Delta\textbf{\textit{)}} \quad \exists\, p_0, p_1 \in \mathcal{P} \text{ and } \Delta > 0 \text{ s.t.} \tag{11}$$

$$p_0 \in \arg\max_{p \in \mathcal{P}} F_{e_{\mathrm{tgt}}}(p; \theta_0), \quad p_1 \in \arg\max_{p \in \mathcal{P}} F_{e_{\mathrm{tgt}}}(p; \theta_1),$$
$$F_{e_{\mathrm{tgt}}}(p_0; \theta_0) - F_{e_{\mathrm{tgt}}}(p; \theta_0) \geq \Delta,\ \forall p \neq p_0, \quad F_{e_{\mathrm{tgt}}}(p_1; \theta_1) - F_{e_{\mathrm{tgt}}}(p; \theta_1) \geq \Delta,\ \forall p \neq p_1. \tag{12}$$

Then for any policy π that can interact only with e_{src}, there exists $i \in \{0, 1\}$ such that for every budget T,

$$\mathbb{E}\big[\mathrm{GR}_T(\theta_i)\big] \;\geq\; \Delta/2.$$

This impossibility holds whether the evaluator is stochastic or deterministic, since equation 10 is stated at the level of the full observation model $P_{e_{\mathrm{src}}}$.

Proof. Let $\mathbb{P}_i$ be the distribution over the full transcript

$$\mathcal{T} := (p_{0:T-1}, y_{0:T-1}, \hat{p})$$

when the true hypothesis is θ_i and interaction is only with e_{src}.

By equation 10, for any history and any chosen action p_t, the conditional distribution of y_t is identical under θ_0 and θ_1. By induction on t, the entire transcript distribution is identical:

$$\mathbb{P}_0 = \mathbb{P}_1.$$

In particular, the marginal distribution of the final output $\hat{p}$ is the same under θ_0 and θ_1. Let this common distribution be denoted by Q on $\mathcal{P}$.

Now consider the expected target regret under θ_0: by equation 12, any output $\hat{p} \neq p_0$ incurs regret at least Δ under θ_0:

$$\text{GR}_T(\theta_0) = F_{e_{\text{tgt}}}(p_0; \theta_0) - F_{e_{\text{tgt}}}(\hat{p}; \theta_0) \;\geq\; \Delta \cdot \mathbf{1}\{\hat{p} \neq p_0\}.$$

Taking expectation w.r.t. Q yields

$$\mathbb{E}[\text{GR}_T(\theta_0)] \;\geq\; \Delta \cdot (1 - Q(\hat{p} = p_0)).$$

Similarly,

$$\mathbb{E}[\text{GR}_T(\theta_1)] \;\geq\; \Delta \cdot (1 - Q(\hat{p} = p_1)).$$

Since $Q(\hat{p} = p_0) + Q(\hat{p} = p_1) \leq 1$, at least one of these probabilities is at most $1/2$, so at least one of the two expected regrets is at least $\Delta/2$:

$$\max\{\mathbb{E}[\text{GR}_T(\theta_0)],\; \mathbb{E}[\text{GR}_T(\theta_1)]\} \;\geq\; \Delta/2.$$

This proves the claim. □

G.3 Concrete examples satisfying the conditions

We give two illustrative examples where source data cannot distinguish two hypotheses, yet the target-optimal decision differs.

Example 1: public test vs private (distribution shift / shortcut feature). Let $\theta \in \{\theta_0, \theta_1\}$ encode which feature is truly stable/causal. Programs correspond to two model families: p_0 uses a stable causal feature; p_1 uses a shortcut feature. In the source environment (public benchmark), the shortcut feature is perfectly correlated with labels, so both hypotheses yield the same evaluator distribution for every program, satisfying equation 10. In the target environment (deployment/private), the shortcut correlation breaks: under θ_0, p_0 is uniquely optimal; under θ_1, p_1 is uniquely optimal, with margin Δ, satisfying equation 12. No amount of interaction with e_{src} can identify which world holds.

Example 2: relaxed verification (slack) vs exact verification (constraint shift). In combinatorial optimization, it is common to evaluate candidate programs using a *relaxed* verifier (e.g., allowing numerical slack), then validate with an *exact* verifier. For instance, in circle packing, one may verify non-overlap with a numerical slack such as 10^{-6}, and later validate with an exact checker; converting a relaxed-feasible solution into an exact-feasible one may require tiny but nonzero modifications, and rankings can change when switching verifiers. This is explicitly discussed in the context of circle packing verification with slack vs exact validation. The source environment e_{src} can correspond to the relaxed evaluator, while the target environment e_{tgt} corresponds to the exact evaluator. Then two hypotheses θ_0, θ_1 can be constructed so that they are indistinguishable under the relaxed evaluator for all queried programs, yet the exact evaluator reverses which program is truly best (with gap Δ), matching Theorem G.1.

H Implementation Details of ExpEvolve

This section describes the implementation of `ExpEvolve` used in our experiments. Alg. 1 summarizes the search loop, and Alg. 2 details the discovery of idea factors. We implement `ExpEvolve` on the backbone of `SCS` (Gideoni et al., 2026). In each generation, `ExpEvolve` generates B programs in parallel slots. The prompt of each slot contains the task instruction, an optional strategy instruction from an idea-guided action, a summary of the idea memory, and a few example programs, and the mutator LLM G writes a new program from the prompt. All LLM calls use DeepSeek-V4-Flash, and the hyperparameters are listed in Table 3. Each generated program is evaluated once. A program

Algorithm 1 `ExpEvolve`

Require: initial archive $\mathcal{A}_0$ of the task; mutator LLM G; generations N; batch size B; random-exploration fraction ρ; plateau window κ

$\mathcal{M}_0 \leftarrow \varnothing$
for $t = 0, \ldots, N-1$ **do**
  Exploit idea factors (Sec. 4.2)
  $(u_{t,1}, \ldots, u_{t,B}) \leftarrow \textsc{Allocate}(\mathcal{M}_t, B, \rho)$ {$\perp$: random exploration}
  for $b = 1, \ldots, B$ **do**
    $p_{t,b} \sim G\big(\cdot \mid \textsc{BuildContext}(u_{t,b}, \mathcal{A}_t, \mathcal{M}_t)\big)$; $y_{t,b} \leftarrow \textsc{Evaluate}(p_{t,b})$
  end for
  Discover ideas from experience (Sec. 4.1)
  $\mathcal{A}_{t+1} \leftarrow \mathcal{A}_t \cup \{(p_{t,b}, y_{t,b}, u_{t,b})\}_{b=1}^{B}$
  $\mathcal{M}_{t+1} \leftarrow \textsc{UpdateMemory}(\mathcal{M}_t, \mathcal{A}_{t+1})$ {annotate; merge; update rewards}
  if $\textsc{DiscoverScheduled}(t)$ **then**
    $\mathcal{M}_{t+1} \leftarrow \textsc{DiscoverIdeas}(\mathcal{A}_{t+1}, \mathcal{M}_{t+1})$ {Alg. 2}
  end if
  Revise at plateaus (Sec. ??)
  if $\textsc{EntersPlateau}(\mathcal{A}_{t+1}, \kappa)$ **then**
    $\mathcal{M}_{t+1} \leftarrow \textsc{ReviseAtPlateau}(\mathcal{A}_{t+1}, \mathcal{M}_{t+1})$ {probe; judge; mask}
  end if
end for
return $\arg\max_{(p,y,\cdot)\in\mathcal{A}_N} y$ and $\mathcal{M}_N$

Algorithm 2 DISCOVERIDEAS

Require: archive $\mathcal{A}$; idea memory $\mathcal{M}$; pair budget ℓ; minimum support $n_{\min}$; admission threshold η
Ensure: updated idea memory $\mathcal{M}$

Select unexplained contrasts
if $\mathcal{M} = \varnothing$ **then**
  $Q \leftarrow \textsc{LargestScoreGaps}(\mathcal{A}, \ell)$
else
  $Q \leftarrow \textsc{MisrankedPairs}(\mathcal{A}, \mathcal{M}, \ell)$ {pairwise ranker on $Z(p)$}
end if
Propose candidate ideas
$\widetilde{\mathcal{M}} \leftarrow \textsc{ProposeIdeas}(Q)$ {name, description, type, values}
Annotate, test, and admit
for each $z \in \widetilde{\mathcal{M}}$ **do**
  $z(p) \leftarrow \textsc{Annotate}(p, z)$ for all $p \in \mathcal{A}$ {in $\mathcal{V} \cup \{\mathsf{unknown}\}$}
  if $\textsc{Supported}(z, \mathcal{A}, n_{\min})$ **and** $|\textsc{ScoreContrast}(z, \mathcal{A})| \geq \eta$ **and not** $\textsc{Duplicate}(z, \mathcal{M})$ **then**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{z\}$
  end if
end for
return $\mathcal{M}$

is treated as failed if the evaluation fails or returns a non-finite objective, and failed programs are neither repaired nor retried. The LLM calls for generation are not retried in our experiments either, so a failed call, e.g., due to rate limits, yields a failed program that still counts toward the evaluation budget.

H.1 IDEA MEMORY AND ANNOTATION

Idea factors. Each idea factor z in the idea memory $\mathcal{M}_t$ is stored with a name, a description of its code pattern, a type, i.e., binary or categorical, and a value set $\mathcal{V}$. The description is also used by the annotator LLM to recognize the idea factor from the source code. The annotator LLM labels each program with $z(p) \in \mathcal{V} \cup \{\mathsf{unknown}\}$, where the label is unknown if the idea factor cannot be recognized. The annotation of a program is a separate LLM call, independent of the proposal of idea factors, with the source code truncated to 4000 characters.

Score contrast. For a binary idea factor, the score contrast is the difference of the mean scores between the successful programs labeled true and those labeled false. For a categorical idea factor, the score contrast is the difference of the mean scores between the value with the highest mean score

and the value with the lowest mean score. Programs labeled unknown are excluded from the score contrast.

Updating the memory. After each generation, UPDATEMEMORY labels the new programs with all existing idea factors, recomputes the score contrasts, and merges two idea factors if the Pearson correlation of their labels on the archive exceeds 0.85 (Alg. 1, line 8). Degenerate evaluations, whose scores collapse to extreme values, are excluded from the statistics. It also updates the statistics of the actions (Appendix H.3).

Memory summary in the prompt. Every slot, including random exploration, receives a summary of the idea memory (line 5). The summary lists the six idea factors with the largest absolute score contrasts, each with its name, description, direction of effect, numbers of programs with and without it, and score contrast, and gives the mean score of each value for categorical idea factors. It also contains the outlier programs that the current idea factors fail to explain, i.e., one program with a large positive residual and one with a large negative residual ($|z| \geq 1.5$), with their source code truncated to 1500 characters.

H.2 IDEA DISCOVERY

Trigger. Discovery is triggered every n_{disc} generations after the first generation, when the archive contains at least 15 successful programs and the idea memory contains fewer than 12 idea factors, excluding the background idea factors (Alg. 1, line 9).

Pairwise ranker. Let σ_Y be the standard deviation of the scores of the successful programs in the archive. `ExpEvolve` builds all pairs of successful programs (p, p') with $|y(p) - y(p')| \geq 0.25\,\sigma_Y$, and labels each pair by $\operatorname{sign}(y(p) - y(p'))$. The feature of a pair is the difference of the binary encodings of the idea factors, where a factor is encoded as 1 if it is labeled true and 0 otherwise, together with the standardized difference of the code lengths as a covariate. The ranker is a ridge least-squares model without intercept ($\lambda = 10^{-3}$), refit in each discovery round.

Selecting unexplained pairs. `ExpEvolve` selects the pairs that the ranker orders incorrectly (Alg. 2, line 4), and excludes the pairs on which the current idea factors take identical values. The selected pairs are ranked by $\mathrm{acc}_0(p, p') \cdot w(p, p')$, where $w(p, p') = 1.5$ if the two programs come from the same generation and 1 otherwise. The null accuracy $\mathrm{acc}_0(p, p')$ is the fraction of 500 random permutations of the scores over programs, under which a ranker refit on the permuted labels still predicts the true order of the pair. A pair with a high null accuracy is easy to order from the idea factors, so that its misranking indicates a missing idea factor. The null accuracy uses the differences of the idea factors without the code-length covariate. Up to $\ell = 6$ pairs are used, with each program appearing in at most two pairs. If fewer than 3 pairs are available, the discovery round is skipped. When the idea memory is empty, `ExpEvolve` selects the ℓ pairs with the largest score gaps under the same constraints (lines 1–2).

Proposing candidates. For each selected pair, the prompt of the proposer LLM contains the score gap in raw units and in units of σ_Y, the scores, the predicted scores and the residuals of both programs, the order predicted by the ranker, the idea factors on which the two programs differ, and the source code of both programs truncated to 1500 characters each. The proposer LLM is asked to explain the unexpected order, and proposes 8 candidates in each round, each with a name, a description, a type, a value set, and a predicted direction (line 6).

Annotation and admission. Each candidate is annotated on the whole archive (line 8). `ExpEvolve` admits a candidate if the two values compared by its score contrast are each taken by at least $n_{\min} = 5$ programs, the absolute score contrast is at least $\eta = \max(0.25\,\sigma_Y,\, 10^{-3})$, and the candidate does not duplicate an idea factor in the memory, an idea factor removed before, or another selected candidate (line 9). Two idea factors are considered duplicates if their names are identical, the token Jaccard similarity of their names is at least 0.6, or the token overlap of their descriptions is at least 0.45. Since the archive is collected adaptively by the search, the score contrast is an association within the archive and does not identify the effect of the idea factor. Admission therefore makes an idea factor available for testing, and the search examines it through the actions (Appendix H.3).

H.3 IDEA-GUIDED ACTIONS

Actions. ALLOCATE compiles the idea factors into actions, i.e., $\mathsf{ADD}(z)$ and $\mathsf{REMOVE}(z)$ for a binary idea factor, and $\mathsf{SET}(z{=}v)$ for each value v of a categorical idea factor (Alg. 1, line 3). The action set is rebuilt in every generation, and keeps at most three actions of the idea factors with the largest absolute score contrasts, together with a baseline action without instruction. An action that has stayed in the action set for N_{rot} consecutive generations is rotated out to give way to other actions, and its records are kept for when it enters the action set again. Before an admitted idea factor is compiled into actions, its absolute score contrast must exceed the 95th percentile of a null distribution obtained from 200 permutations of the scores, and a threshold on the absolute score contrast is used instead when the permutation test cannot be run. Background idea factors (Appendix H.4) are not compiled into actions, until they satisfy the admission conditions of Appendix H.2 and are promoted to regular idea factors.

Prompt of an action. An idea-guided slot differs from random exploration in two parts of the prompt (line 5). First, a strategy instruction asks the mutator LLM to use or avoid the pattern of z, or to set z to v, with the description of z and its estimated effect. Second, the example programs consist of the best program in the archive as an anchor, followed by the best programs in the target state of the action, where ties between programs with similar scores are broken in favor of programs with rare combinations of idea factors, and $k = 3$ example programs are used in total. There is no explicit parent program, since `SCS` conditions the mutator LLM on a set of example programs.

Random exploration. A random exploration slot uses the prompt of `SCS` without strategy instruction, with 3 example programs sampled uniformly at random from the successful programs, and the summary of the idea memory.

Diversification in the warm-up. In the warm-up generations, the slots of the baseline action in the second half of each generation receive a diversification instruction, which asks the mutator LLM to first list three to five distinct families of well-established approaches to the task as code comments, and to note which families the example programs use.

Allocating slots. In each generation, ALLOCATE assigns $\max(2, \mathrm{round}(\rho B))$ slots to random exploration, and the remaining slots to idea-guided actions, where $u_{t,b}$ denotes the action of slot b and $u_{t,b} = \perp$ denotes random exploration (line 3). An idea-guided slot falls back to random exploration if no action is available, e.g., before any idea factor is admitted, or if the actions have reached the limit of slots per generation. The fraction of random exploration is thus at least ρ.

Bandit. The bandit selects the actions of idea-guided slots with a soft UCB rule,

$$s(a) = \tilde{r}_{\max}(a) + c\sqrt{\frac{\ln(N_{\mathrm{tot}}+1)}{n_a}}, \qquad \Pr(a) \propto \exp\big(s(a)\big),$$

where n_a is the number of times that a has been selected, $N_{\mathrm{tot}} = \sum_a n_a$, and $c = 1$. Actions that have not been selected are tried first. The value $\tilde{r}_{\max}(a)$ is the maximum reward of a, minus a bootstrap estimate of the expected maximum of n_a rewards drawn from all actions, which corrects the growth of the maximum with the number of samples. The reward of a program generated under an action follows Eq. (**??**), where the standard deviation is lower bounded as $\sigma_{\mathrm{rand}} = \max\big(\mathrm{std}(y_{\mathrm{rand}}),\, 0.05\,|\bar{y}_{\mathrm{rand}}|,\, 10^{-6}\big)$ (line 8). If fewer than two random exploration programs are available in the generation, the adjacent generations are included. A failed program receives a penalty score below the worst successful program.

H.4 IDEA-GUIDED EXPLORATION AT PLATEAUS

Entering and leaving a plateau. `ExpEvolve` records the best-so-far score at the end of each generation, and the search is in a plateau if the best-so-far scores of the last $\kappa + 1$ generations are equal up to a relative tolerance of 10^{-4}, after the warm-up generations. When the search enters a plateau, REVISEATPLATEAU runs the background probe and the dependency judge below (Alg. 1, lines 12–13). The plateau ends as soon as the best-so-far score improves, and all the changes during the plateau are reverted in the same generation. Each time the search enters a plateau again, the probe and the judge are run again, since the background of a new plateau can differ from the previous one.

Background probe. An LLM reads the 8 successful programs with the highest scores, with the source code truncated to 1500 characters and the names of the current idea factors, and proposes 2 to 4 binary properties that these programs share. Each property is annotated on the archive, and is recorded as a background idea factor if at least 5 of the 8 top programs are annotated and at least 80% of them realize the property. A background idea factor stays in the memory after the plateau, and is promoted to a regular idea factor once enough programs without it appear in the archive and its score contrast passes the admission threshold.

Dependency judge. The background is described by the three idea factors with the largest absolute score contrasts. For each idea factor in the memory, and for each value of a categorical idea factor, an LLM judge decides, with three calls, whether it depends on the background, i.e., loses its meaning or reverses its effect once the defining property of the background no longer holds.

Changes during a plateau. While the search is in a plateau, `ExpEvolve` applies the following changes.

- *Reversing and masking actions.* For a binary idea factor judged dependent, $\mathsf{ADD}(z)$ is replaced by $\mathsf{REMOVE}(z)$ in the action set, with the same description and score contrast, so that the bandit can only move away from z. For a categorical idea factor, the actions of the values judged dependent are masked, while the other values are kept as escape actions. The dependent idea factors are also removed from the summary of the idea memory. After the plateau, the original actions are restored, and the records of the reversed and masked actions are kept.
- *Stall note.* A note on the current state is prepended to the prompt of every slot. The note reports the best score and the number of generations without improvement, lists the dependent idea factors with their names and shortened descriptions, states that the search has stalled on this combination, and asks whether other constructions or combinations can reach a higher score.
- *More random exploration.* Each idea-guided slot is reassigned to random exploration with probability 0.9. Since the judge runs at the end of the first plateau generation, the probability is lowered to 0.3 from the next generation.
- *Diversified demonstrations.* Random exploration uses the best program of the background as the first example, and fills the other examples with programs outside the background, which are selected greedily to diversify their idea factors. A program is considered in the background if it realizes any dependent idea factor.
- *Exploration of the bandit.* In the first 2 generations of a plateau, the bandit selects the actions by the exploration bonus only.

I Experimental Details

I.1 Tasks

Table 4 summarizes the 11 tasks. Nine tasks are the mathematical problems studied by `SCS` (Gideoni et al., 2026), which are taken from `AlphaEvolve` (Novikov et al., 2025), and we report their reference values as collected by Gideoni et al. (2026). All methods use the same evaluator for each task, and each evaluation is limited to 60 seconds, except the uncertainty inequality with 300 seconds.

First autocorrelation inequality. Let C_1 be the largest constant such that $\max_t (f * f)(t) \geq C_1 \big(\int f \big)^2$ for all non-negative functions f supported on $[-1/4, 1/4]$, where $f * f$ is the autoconvolution. A non-negative step function f gives an upper bound $C_1 \leq \max_t (f * f)(t) / (\int f)^2$, which the task minimizes. The number of steps is chosen by the program, and for a step function with n steps and values h, the score is computed as $2n \max(h * h) / (\sum h)^2$, which is invariant to the scale of h.

Second autocorrelation inequality. Let C_2 be the smallest constant such that $\|f * f\|_2^2 \leq C_2 \|f * f\|_1 \|f * f\|_\infty$ for all non-negative functions f. A non-negative step function f gives a lower bound $C_2 \geq \|f * f\|_2^2 / (\|f * f\|_1 \|f * f\|_\infty)$, which the task maximizes. The evaluator rejects step functions with more than 10^4 steps.

Table 3: Hyperparameters of `ExpEvolve` in the main experiments. The budget of 200 evaluations is run with two configurations of generations N and batch size B. Both configurations are run on all 11 tasks, each with 3 seeds.

Hyperparameter	$N = 20, B = 10$	$N = 10, B = 20$
Fraction of random exploration ρ	0.6	0.4
Discovery interval n_{disc} (generations)	2	1
Plateau window κ (generations)	4	2
Warm-up generations	2	1
Action rotation N_{rot} (generations)	6	3
Shared by both configurations		
Minimum archive size for discovery	15	
Maximum number of idea factors	12	
Score gap of candidate pairs	$0.25\,\sigma_Y$	
Pair budget ℓ	6	
Weight of same-generation pairs	1.5	
Permutations for the null accuracy	500	
Permutations and quantile of the action gate	200, 0.95	
Candidates per discovery round	8	
Minimum support $n_{\min}$	5	
Admission threshold η	$\max(0.25\,\sigma_Y, 10^{-3})$	
Actions in the action set	≤ 3 and a baseline action	
Example programs per prompt k	3	
UCB coefficient c	1	
Top programs in the background probe	8	
Reassignment probability at plateaus	0.9, then 0.3	
Bandit exploration window (generations)	2	
LLM of all calls	DeepSeek-V4-Flash	
Temperature of proposer / annotator / judge	0.7 / 0.2 / 0.3	

Table 4: Tasks, optimization directions, and reference values. The previous best values and the values of `AlphaEvolve` are collected by Gideoni et al. (2026). For the uncertainty inequality, all methods in Gideoni et al. (2026) reach 0.3482 under their reformulated parameterization, which we use in our experiments. For the Hadamard matrix, the previous best is the normalized score of the best-known construction.

Task	Direction	Previous best	`AlphaEvolve`
First autocorrelation inequality	↓	1.5098	1.5053
Second autocorrelation inequality	↑	0.88922	0.8962
Uncertainty inequality	↓	0.3523	0.3521
Erdős minimum overlap	↓	0.380927	0.380924
Sums and differences of sets	↑	1.14465	1.1584
Max–min distance ratio (16 points)	↓	12.890	12.88927
Heilbronn for triangles ($n = 11$)	↑	0.036	0.0365
Kissing number (11 dimensions)	↑	592	593
Circle packing (26 circles)	↑	2.634	2.63586
Circle packing via LP (26 circles)	↑	–	–
Hadamard matrix ($n = 29$)	↑	0.9357	–

Uncertainty inequality. For a function g, let $A(g) = \inf\{r > 0 : g(x) \geq 0 \text{ for all } |x| \geq r\}$. For even functions f with $f(0) \leq 0$ and $\hat{f}(0) \leq 0$, the task minimizes $A(f)A(\hat{f})$, which upper bounds the largest constant C such that $A(f)A(\hat{f}) \geq C$. Following the reformulation of Gideoni et al. (2026), f is a Hermite–Gaussian expansion over the monic (probabilist's) Hermite polynomials $H_0, H_4, \ldots, H_{28}$ with 7 free coefficients, which represents a larger class of functions than the original formulation over H_0, H_4, H_8, H_{12} with 3 free coefficients. The sign conditions are checked on a grid of 2×10^5 points in $[10^{-6}, 30]$, which is refined to 2×10^6 points with extended precision when the score is below 0.6, and each evaluation is limited to 300 seconds. Since the reformulation enlarges the search space, the resulting values are not comparable with the values of the original formulation

in Table 4. In the implementation, the score is $r^2/(2\pi)$, where r is the largest root at which f changes sign.

Erdős minimum overlap problem. For a partition of $\{1, \dots, 2n\}$ into two sets A and B of equal size, let $M(n) = \min_{A,B} \max_k |\{(a, b) \in A \times B : a - b = k\}|$. The task constructs a step function h with values in $[0, 1]$, whose number of steps is chosen by the program, and minimizes the upper bound on $\lim_{n\to\infty} M(n)/n$ given by twice the maximum correlation between h and $1 - h$ divided by the number of steps.

Sums and differences of sets. Following the framework of Gyarmati, Hennecart, and Ruzsa, the task constructs a set $U \subset \mathbb{N}_{\geq 0}$ with $0 \in U$ that maximizes $\theta(U) = 1 + \log(|U - U|/|U + U|)/\log(2\max(U) + 1)$, which lower bounds the largest exponent θ such that $|A - B| \geq c(K)|A + B|^\theta$ for arbitrarily large sets A, B with $|A + B| \leq K|A|$.

Max–min distance ratio. The task places 16 points in the plane to minimize the squared ratio between the maximum and the minimum pairwise distances. For example, the squared ratios 12.889 and 13.0 in Table 1 correspond to the ratios 3.5902 and 3.6056.

Heilbronn problem for triangles. The task places 11 points in the equilateral triangle with vertices $(0, 0)$, $(1, 0)$, and $(1/2, \sqrt{3}/2)$ to maximize the smallest area of the triangles formed by any three of the points, normalized by the area $\sqrt{3}/4$ of the equilateral triangle.

Kissing number in 11 dimensions. A set of points $C \subset \mathbb{R}^{11}$ with $0 \notin C$ and $\min_{x\neq y} \|x - y\| \geq \max_x \|x\|$ over $x, y \in C$ yields a kissing configuration of $|C|$ unit spheres. The task constructs such a set, and the score is the number of points $|C|$. The distances and norms are only used to check the validity of the configuration, and a configuration with more points than the known upper bound 868 is rejected.

Circle packing. The task places 26 non-overlapping circles in the unit square to maximize the sum of their radii. The evaluator checks the non-overlap and the containment of all circles, including that every radius is at most 0.5 and every center lies in the unit square, and checks that the sum of radii reported by the program agrees with the radii up to a relative and absolute tolerance of 10^{-3}.

Circle packing via linear programming. A variant of circle packing where the program proposes only the 26 centers $C_i \in [0, 1]^2$, and the radii are recovered by a linear program that maximizes $\sum_i r_i$ subject to $r_i + r_j \leq \|C_i - C_j\|$ and $r_i \leq \min(C_i^x, C_i^y, 1 - C_i^x, 1 - C_i^y)$. The variant isolates the layout of the circles from the assignment of the radii. The starting code in the prompt solves the linear program with the HiGHS solver of SciPy.

Hadamard matrix. The task constructs a matrix $H \in \{\pm 1\}^{29\times 29}$ that maximizes $|\det(H)|/(2^{28} \cdot 7^{12} \cdot 342)$. Following the convention of Wang et al. (2026), the best-known construction with $|\det(H)| = 2^{28} \cdot 7^{12} \cdot 320$ scores $320/342 \approx 0.9357$.

I.2 Baselines

`SCS`. We use the implementation of `SCS` (Gideoni et al., 2026), which generates each program from a prompt containing the task instruction and 3 example programs sampled uniformly at random without replacement from the archive. `SCS` does not maintain an idea memory, and its prompt does not contain the summary of the idea memory used by `ExpEvolve`.

SCS+meta. SCS+meta adds the meta-summary of `ShinkaEvolve` to the prompt of `SCS`, i.e., a textual summary of the insights of recent programs with at most 5 recommendations, which is rewritten by DeepSeek-V4-Flash at temperature 0 after every generation, without the idea memory, the idea-guided actions, and the exploration at plateaus of `ExpEvolve`. It is run on First autocorrelation, Heilbronn, and Kissing with 10 generations of 20 programs, and on Uncertainty inequality with 20 generations of 10 programs.

`ShinkaEvolve`. We run `ShinkaEvolve` (Lange et al., 2026) with 5 islands and migration every 10 generations, weighted parent sampling, an archive of 40 programs, a meta-summary every 10 generations with 5 recommendations, and both diff-based and full rewrites. `ShinkaEvolve` generates one program per generation, and does not distinguish the two configurations of `ExpEvolve` and `SCS`.

I.3 Budget and Resources

All methods evaluate each generated program once, and a failed program consumes the budget. The LLM calls of `ExpEvolve` for proposing, annotating, probing, and judging idea factors do not evaluate programs and do not consume the budget. Each run of `ShinkaEvolve` starts from 5 copies of the initial program, one for each island, and then generates 199 programs, so its final performance is reported at 199 generated programs. The evaluations of `ExpEvolve` and `SCS` are not limited in CPUs, while the runs of `ShinkaEvolve` on First autocorrelation, Second autocorrelation, Kissing, and Max–min distance ratio evaluate each program with at most 8 CPUs. The LLM is accessed either through the official API of DeepSeek or through OpenRouter, depending on the time of the run.

J Additional Results

Per-seed results. Tables 5 and 6 report the final best score of each seed for the cells of Table 1, and Tables 7 and 8 report the best-so-far scores along the budget.

Sample efficiency. For each setting, $n_{\texttt{ExpEvolve}}$ is the first number of evaluated programs at which the best-so-far score of `ExpEvolve`, averaged over the seeds that have produced a successful program, reaches the final best score of `SCS` averaged over seeds, and $n_{\texttt{SCS}}$ is the first number of evaluated programs at which the best-so-far score of `SCS`, averaged in the same way, reaches its final value. The programs within a generation are ordered by their slots. Table 9 reports the four tasks in Fig. 6.

K Prompts

Prompt of a slot. The prompt of each slot is assembled as follows, where the strategy instruction is present only for idea-guided slots, and the summary of the idea memory is present for all slots.

```
{base_instruction}

# Strategy for this attempt
{directive}

{causal_info_block}

# Successful examples (to improve on):

## Solution {i+1} ({metric} = {value}):
```
{code}
```

Please now provide your new, better implementation of the required
function (exact signature from the task statement above). Address the
strategy above and avoid the failure patterns shown.
```

Strategy instructions. The strategy instruction of an action on a binary idea factor and on a categorical idea factor are as follows, with lines wrapped for display.

```
STRATEGY (this batch is targeting a discovered code factor):
{USE|AVOID} the `{feature_name}` pattern{ ({desc})}. Past data on this
run estimates its effect on {conf_metric_name} at ATE={+}{ate:.3f}.
The first example below is the highest-Y program (anchor, regardless
of this pattern); subsequent examples {USE this pattern|DO NOT use this
pattern}. Keep the pattern direction while improving on the anchor's
overall quality.
```

```
STRATEGY (this batch is targeting a categorical code factor): SET the
`{feature_name}` factor to value `{level}`{ ({desc})}. Past data shows
programs with this exact level gain {+}{ate:.3f} {conf_metric_name} vs
other levels of the factor. The first example below is the highest-Y
program (anchor); the other examples USE this exact level. Adopt the
`{level}` setting and improve on the anchor.
```

Table 5: Final best score of each seed at the budget of 200, and the mean ± standard deviation over seeds.

Task	Method	Config.	Seed 1	Seed 2	Seed 3	Mean
Hadamard ↑	ExpEvolve	10×20	0.9211	0.9357	0.9211	0.9259 ± 0.0084
	ExpEvolve	20×10	0.9211	0.9211	0.9211	0.9211 ± 0.0000
	SCS	10×20	0.8596	0.9211	0.7240	0.8349 ± 0.1008
	SCS	20×10	0.8596	0.5665	0.5662	0.6641 ± 0.1693
	ShinkaEvolve	–	0.5603	0.5733	0.8066	0.6467 ± 0.1386
Pack ↑	ExpEvolve	10×20	2.63304	2.63598	2.63595	2.63499 ± 0.00169
	ExpEvolve	20×10	2.63598	2.63429	2.63598	2.63542 ± 0.00098
	SCS	10×20	2.62760	2.63104	2.61050	2.62305 ± 0.01100
	SCS	20×10	2.63017	2.63572	2.58501	2.61697 ± 0.02781
	ShinkaEvolve	–	2.61408	2.54142	2.59052	2.58201 ± 0.03707
Linprog ↑	ExpEvolve	10×20	2.63598	2.63429	2.63598	2.63542 ± 0.00098
	ExpEvolve	20×10	2.63429	2.63598	2.63598	2.63542 ± 0.00098
	SCS	10×20	2.55342	2.54533	2.55608	2.55161 ± 0.00560
	SCS	20×10	2.63598	2.63429	2.63598	2.63542 ± 0.00098
	ShinkaEvolve	–	2.51192	2.54142	2.48969	2.51434 ± 0.02595
SumDiff ↑	ExpEvolve	10×20	1.12281	1.12142	1.12250	1.12224 ± 0.00073
	ExpEvolve	20×10	1.11937	1.10954	1.11937	1.11609 ± 0.00567
	SCS	10×20	1.11033	1.12006	1.11786	1.11608 ± 0.00510
	SCS	20×10	1.11143	1.11123	1.10995	1.11087 ± 0.00080
	ShinkaEvolve	–	1.10954	1.10954	1.10954	1.10954 ± 0.00000
Heilbronn ↑	ExpEvolve	10×20	0.0359	0.0365	0.0351	0.0358 ± 0.0007
	ExpEvolve	20×10	0.0357	0.0359	0.0342	0.0353 ± 0.0009
	SCS	10×20	0.0343	0.0333	0.0331	0.0336 ± 0.0006
	SCS	20×10	0.0340	0.0288	0.0312	0.0313 ± 0.0026
	ShinkaEvolve	–	0.0303	0.0324	0.0283	0.0303 ± 0.0021
Min overlap ↓	ExpEvolve	10×20	0.38096	0.38102	0.38095	0.38098 ± 0.00004
	ExpEvolve	20×10	0.38096	0.38095	0.38100	0.38097 ± 0.00003
	SCS	10×20	0.38130	0.38094	0.38097	0.38107 ± 0.00020
	SCS	20×10	0.38104	0.38210	0.38154	0.38156 ± 0.00053
	ShinkaEvolve	–	0.38239	0.38243	0.38246	0.38242 ± 0.00003

Table 6: Final best score of each seed (continued) at the budget of 200 evaluated programs (199 for `ShinkaEvolve`), and the mean ± standard deviation over seeds.

Task	Method	Config.	Seed 1	Seed 2	Seed 3	Mean
Kissing ↑	`ExpEvolve`	10 × 20	582	582	582	582 ± 0
	`ExpEvolve`	20 × 10	414	582	582	526 ± 97
	`SCS`	10 × 20	582	582	582	582 ± 0
	`SCS`	20 × 10	406	438	394	413 ± 23
	`ShinkaEvolve`	–	414	406	566	462 ± 90
Second autocorr. ↑	`ExpEvolve`	10 × 20	0.8825	0.8825	0.8825	0.8825 ± 0.0000
	`ExpEvolve`	20 × 10	0.8825	0.8825	0.8825	0.8825 ± 0.0000
	`SCS`	10 × 20	0.8825	0.8825	0.8825	0.8825 ± 0.0000
	`SCS`	20 × 10	0.8499	0.8485	0.8210	0.8398 ± 0.0163
	`ShinkaEvolve`	–	0.8739	0.8825	0.8671	0.8745 ± 0.0077
First autocorr. ↓	`ExpEvolve`	10 × 20	1.5151	1.5176	1.5257	1.5195 ± 0.0055
	`ExpEvolve`	20 × 10	1.5104	1.5098	1.5137	1.5113 ± 0.0021
	`SCS`	10 × 20	1.5587	1.5560	1.5378	1.5508 ± 0.0113
	`SCS`	20 × 10	1.5151	1.5354	1.5202	1.5235 ± 0.0106
	`ShinkaEvolve`	–	1.5203	1.5708	1.5708	1.5540 ± 0.0292
Max–min ratio ↓	`ExpEvolve`	10 × 20	12.8892	12.8892	12.8892	12.8892 ± 0.0000
	`ExpEvolve`	20 × 10	13.0000	12.8892	12.8892	12.9262 ± 0.0640
	`SCS`	10 × 20	12.9425	13.0000	13.0000	12.9808 ± 0.0332
	`SCS`	20 × 10	13.0229	12.8892	12.8892	12.9338 ± 0.0772
	`ShinkaEvolve`	–	13.0000	13.0000	13.0000	13.0000 ± 0.0000
Hermite ↓	`ExpEvolve`	10 × 20	0.34820	0.35172	0.35274	0.35089 ± 0.00238
	`ExpEvolve`	20 × 10	0.35321	0.35263	0.35834	0.35473 ± 0.00314
	`SCS`	10 × 20	0.35394	0.36109	0.36365	0.35956 ± 0.00503
	`SCS`	20 × 10	0.46232	0.35945	0.36116	0.39431 ± 0.05890
	`ShinkaEvolve`	–	6.91428	0.35429	6.91480	4.72779 ± 3.78756

Table 7: Best-so-far scores along the budget, as the mean ± standard deviation over 3 seeds after 40, 80, 120, 160, and 200 evaluated programs.

Task	Method	Config.	40	80	120	160	200
Hadamard ↑	`ExpEvolve`	10 × 20	0.8801 ± 0.0355	0.9211 ± 0.0000	0.9259 ± 0.0084	0.9259 ± 0.0084	0.9259 ± 0.0084
	`ExpEvolve`	20 × 10	0.9006 ± 0.0355	0.9006 ± 0.0355	0.9211 ± 0.0000	0.9211 ± 0.0000	0.9211 ± 0.0000
	`SCS`	10 × 20	0.5047 ± 0.0113	0.6446 ± 0.1866	0.6494 ± 0.1828	0.7144 ± 0.1502	0.8349 ± 0.1008
	`SCS`	20 × 10	0.6506 ± 0.1821	0.6506 ± 0.1821	0.6556 ± 0.1771	0.6641 ± 0.1693	0.6641 ± 0.1693
	`ShinkaEvolve`	–	0.4909 ± 0.0171	0.5093 ± 0.0093	0.5297 ± 0.0258	0.5588 ± 0.0131	0.6467 ± 0.1386
Pack ↑	`ExpEvolve`	10 × 20	2.62275 ± 0.00820	2.62721 ± 0.00567	2.63201 ± 0.00090	2.63499 ± 0.00169	2.63499 ± 0.00169
	`ExpEvolve`	20 × 10	2.63046 ± 0.00347	2.63289 ± 0.00397	2.63326 ± 0.00336	2.63485 ± 0.00098	2.63542 ± 0.00098
	`SCS`	10 × 20	2.60036 ± 0.03729	2.60307 ± 0.03920	2.60423 ± 0.04016	2.61452 ± 0.02569	2.62305 ± 0.01100
	`SCS`	20 × 10	2.60046 ± 0.02223	2.61200 ± 0.02384	2.61383 ± 0.02605	2.61697 ± 0.02781	2.61697 ± 0.02781
	`ShinkaEvolve`	–	2.51452 ± 0.04659	2.53382 ± 0.01316	2.56349 ± 0.03437	2.57166 ± 0.03783	2.58201 ± 0.03707
Linprog ↑	`ExpEvolve`	10 × 20	2.63043 ± 0.00962	2.63435 ± 0.00282	2.63542 ± 0.00098	2.63542 ± 0.00098	2.63542 ± 0.00098
	`ExpEvolve`	20 × 10	2.63165 ± 0.00241	2.63444 ± 0.00148	2.63542 ± 0.00098	2.63542 ± 0.00098	2.63542 ± 0.00098
	`SCS`	10 × 20	2.52974 ± 0.01579	2.54272 ± 0.00225	2.54672 ± 0.00612	2.55161 ± 0.00560	2.55161 ± 0.00560
	`SCS`	20 × 10	2.63328 ± 0.00332	2.63486 ± 0.00098	2.63486 ± 0.00098	2.63542 ± 0.00098	2.63542 ± 0.00098
	`ShinkaEvolve`	–	2.46507 ± 0.09226	2.48623 ± 0.06618	2.48853 ± 0.06724	2.51167 ± 0.02988	2.51434 ± 0.02595
SumDiff ↑	`ExpEvolve`	10 × 20	1.11026 ± 0.00102	1.11364 ± 0.00470	1.11948 ± 0.00220	1.12089 ± 0.00065	1.12224 ± 0.00073
	`ExpEvolve`	20 × 10	1.10968 ± 0.00023	1.10968 ± 0.00023	1.11413 ± 0.00480	1.11473 ± 0.00484	1.11609 ± 0.00567
	`SCS`	10 × 20	1.10981 ± 0.00023	1.10990 ± 0.00009	1.11387 ± 0.00473	1.11564 ± 0.00497	1.11608 ± 0.00510
	`SCS`	20 × 10	1.10968 ± 0.00023	1.11031 ± 0.00099	1.11031 ± 0.00099	1.11031 ± 0.00099	1.11087 ± 0.00080
	`ShinkaEvolve`	–	1.10954 ± 0.00000	1.10954 ± 0.00000	1.10954 ± 0.00000	1.10954 ± 0.00000	1.10954 ± 0.00000
Heilbronn ↑	`ExpEvolve`	10 × 20	0.0315 ± 0.0008	0.0349 ± 0.0012	0.0350 ± 0.0014	0.0354 ± 0.0014	0.0358 ± 0.0007
	`ExpEvolve`	20 × 10	0.0336 ± 0.0016	0.0339 ± 0.0013	0.0342 ± 0.0013	0.0349 ± 0.0010	0.0353 ± 0.0009
	`SCS`	10 × 20	0.0310 ± 0.0025	0.0325 ± 0.0003	0.0326 ± 0.0005	0.0331 ± 0.0002	0.0336 ± 0.0006
	`SCS`	20 × 10	0.0294 ± 0.0021	0.0309 ± 0.0025	0.0309 ± 0.0025	0.0311 ± 0.0022	0.0313 ± 0.0026
	`ShinkaEvolve`	–	0.0261 ± 0.0007	0.0268 ± 0.0012	0.0290 ± 0.0024	0.0297 ± 0.0024	0.0303 ± 0.0021
Min overlap ↓	`ExpEvolve`	10 × 20	0.38109 ± 0.00012	0.38107 ± 0.00010	0.38101 ± 0.00007	0.38098 ± 0.00004	0.38098 ± 0.00004
	`ExpEvolve`	20 × 10	0.38121 ± 0.00009	0.38116 ± 0.00016	0.38106 ± 0.00014	0.38105 ± 0.00013	0.38097 ± 0.00003
	`SCS`	10 × 20	0.38141 ± 0.00029	0.38114 ± 0.00017	0.38108 ± 0.00019	0.38107 ± 0.00020	0.38107 ± 0.00020
	`SCS`	20 × 10	0.38351 ± 0.00315	0.38344 ± 0.00323	0.38344 ± 0.00323	0.38156 ± 0.00053	0.38156 ± 0.00053
	`ShinkaEvolve`	–	0.38285 ± 0.00065	0.38242 ± 0.00003	0.38242 ± 0.00003	0.38242 ± 0.00003	0.38242 ± 0.00003

Stall note. At plateaus, the following note is prepended to the prompt of every slot, where `{blur}` lists the dependent idea factors as their names followed by the first 90 characters of their descriptions, separated by semicolons, and `{stall}` is "best has stayed at `{best}` for `{streak}` generations".

```
CURRENT-STATE NOTE (read first): the most common combination among the
current top programs is: {blur}. Progress with this combination has
stalled -- {stall}. Consider: are there OTHER constructions or
combinations, or variants of the current one, that could reach higher?
If you know one, this attempt is a good moment to try it.
```

Annotator. The annotator LLM labels the idea factors of a program with the following prompt, where the descriptions are truncated to 200 characters and the source code to 4000 characters.

```
[system]
You annotate program code for the presence of specific algorithmic
factors. Output STRICT JSON only with key 'assignments'.

[user]
Factors to annotate:
- <name> (binary): <description>
- <name> (categorical, values: [...]): <description>

Program code:
```python
<code>
```

Output JSON: { "assignments": { "<factor_name>": <value>, ... } }
For binary factors, value should be true or false. For categorical,
value should be one of the listed values. Use null only if unknown.
```

Background probe. At the entry of a plateau, the background probe reads the 8 successful programs with the highest scores, with the source code truncated to 1500 characters, and the names of the current idea factors.

```
These are the TOP-SCORING programs of a search that has plateaued --
they are the stuck family.

## Program {i} ({metric}={score}):
```
{code}
```
...

Factors already tracked (do NOT repeat these or trivial rephrasings):
- {name}
...

Name the structural / algorithmic properties that these top programs
SHARE -- especially the construction or representation they all rely
on. These will be used to describe the stuck family so the search can
deliberately depart from it. Binary properties only.
Return ONLY a JSON array: [{"name":"<snake_case>","description":
"<one sentence, concrete>"}] (2 to 4 entries)
```

K.1 More Details of Experiments on Machine Learning Engineering

We further study whether the effectiveness of `ExpEvolve` transfers to machine learning engineering (MLE). We integrate its idea discovery and search control into OpenRSI (Yang et al., 2026b) and evaluate the resulting agent on MLS-Bench (Lyu et al., 2026a). This setting examines the use of idea factors within a harness that supports evolution on ML algorithms towards a specific task objective.

Tasks and data. MLS-Bench evaluates agents on improving components of ML algorithms under fixed task interfaces and evaluation protocols. Here we consider **6** tasks: *Online bandit*, *Anomaly detection*, *Clustering*, *Multi-objective optimization*, *Subgroup calibration shift*, and *Evolution strategy*. Search leverages the scores and execution feedback exposed by each task.

Backbone and budget. We use the evolutionary harness from OpenRSI, which organizes search through Draft, Improve, Debug, and Crossover operators. For each task, we retain a fixed

history containing 20 completed search slots from an earlier OpenRSI run. The OpenRSI baseline and OpenRSI equipped with `ExpEvolve` continue from this same history for **50** additional evaluator calls, with **3** seeds. Both methods use **5** search slots per generation. All program generation uses Qwen3.6-35B-A3B. The same checkpoint performs factor discovery and annotation in `ExpEvolve`.

Integrating idea factors with OpenRSI. `ExpEvolve` initializes its idea memory from the shared history and updates it with newly evaluated programs. Consistent with Sec. 4, Discovery examines program pairs whose score differences are insufficiently explained by the current factors. The LLM reads their source and proposes reusable mechanisms. Factors z enter the action catalog when both states have sufficient support and exhibit a score contrast in the archive. Each generation reserves two slots for OPEN exploration and schedules three for GUIDED search. Specifically, A guided slot first selects an action, such as $\mathsf{ADD}(z)$ or $\mathsf{REMOVE}(z)$, and then the harness selects a high-scoring parent compatible with that change. We use improvement over the best selected parent as action feedback, adapting the reward reference to OpenRSI's parent–child structure. When the cumulative best score remains unchanged within relative tolerance 10^{-4} across three generation boundaries, `ExpEvolve` switches to idea-guided exploration at plateaus, aligning with Sec. 4.2.

Evaluation and results. We evaluate search performance using the best-so-far (BSF) score. For each task and method, we average the final BSF score over three seeds, starting from the same 20-slot history and continuing for 50 additional evaluations. Fig. 9(a) compares OpenRSI with OpenRSI+EXPEVOLVE across the six tasks. The reported configurations improve the mean final BSF score on all six tasks, with the largest gains on Online bandit and Anomaly detection.

Table 8: Best-so-far scores along the budget (continued), as the mean ± standard deviation over 3 seeds after 40, 80, 120, 160, and 200 evaluated programs. For `ShinkaEvolve`, the last column reports 199 programs.

Task	Method	Config.	40	80	120	160	200
Kissing ↑	`ExpEvolve`	10×20	578 ± 6	582 ± 0	582 ± 0	582 ± 0	582 ± 0
	`ExpEvolve`	20×10	348 ± 0	453 ± 113	523 ± 103	523 ± 103	526 ± 97
	`SCS`	10×20	564 ± 3	582 ± 0	582 ± 0	582 ± 0	582 ± 0
	`SCS`	20×10	394 ± 38	394 ± 38	402 ± 34	402 ± 33	413 ± 23
	`ShinkaEvolve`	–	426 ± 68	459 ± 92	459 ± 92	461 ± 91	462 ± 90
Second autocorr. ↑	`ExpEvolve`	10×20	0.8825 ± 0.0000	0.8825 ± 0.0000	0.8825 ± 0.0000	0.8825 ± 0.0000	0.8825 ± 0.0000
	`ExpEvolve`	20×10	0.8649 ± 0.0154	0.8823 ± 0.0004	0.8824 ± 0.0002	0.8825 ± 0.0000	0.8825 ± 0.0000
	`SCS`	10×20	0.8825 ± 0.0000	0.8825 ± 0.0000	0.8825 ± 0.0000	0.8825 ± 0.0000	0.8825 ± 0.0000
	`SCS`	20×10	0.8492 ± 0.0010	0.8398 ± 0.0163	0.8398 ± 0.0163	0.8398 ± 0.0163	0.8398 ± 0.0163
	`ShinkaEvolve`	–	0.8577 ± 0.0215	0.8650 ± 0.0172	0.8732 ± 0.0096	0.8732 ± 0.0096	0.8745 ± 0.0077
First autocorr. ↓	`ExpEvolve`	10×20	1.5528 ± 0.0113	1.5316 ± 0.0233	1.5217 ± 0.0093	1.5216 ± 0.0091	1.5195 ± 0.0055
	`ExpEvolve`	20×10	1.5196 ± 0.0021	1.5169 ± 0.0053	1.5124 ± 0.0023	1.5119 ± 0.0019	1.5113 ± 0.0021
	`SCS`	10×20	1.5674 ± 0.0056	1.5626 ± 0.0076	1.5549 ± 0.0165	1.5508 ± 0.0113	1.5508 ± 0.0113
	`SCS`	20×10	1.5482 ± 0.0284	1.5262 ± 0.0143	1.5262 ± 0.0143	1.5235 ± 0.0106	1.5235 ± 0.0106
	`ShinkaEvolve`	–	1.5664 ± 0.0076	1.5664 ± 0.0076	1.5622 ± 0.0149	1.5554 ± 0.0268	1.5540 ± 0.0292
Max–min ratio ↓	`ExpEvolve`	10×20	12.8892 ± 0.0000	12.8892 ± 0.0000	12.8892 ± 0.0000	12.8892 ± 0.0000	12.8892 ± 0.0000
	`ExpEvolve`	20×10	12.9266 ± 0.0636	12.9262 ± 0.0640	12.9262 ± 0.0640	12.9262 ± 0.0640	12.9262 ± 0.0640
	`SCS`	10×20	14.8410 ± 1.5646	13.8205 ± 1.4212	13.0557 ± 0.0965	12.9808 ± 0.0332	12.9808 ± 0.0332
	`SCS`	20×10	13.4169 ± 0.9139	13.2822 ± 0.6807	13.2420 ± 0.6110	12.9629 ± 0.1277	12.9338 ± 0.0772
	`ShinkaEvolve`	–	13.0000 ± 0.0000	13.0000 ± 0.0000	13.0000 ± 0.0000	13.0000 ± 0.0000	13.0000 ± 0.0000
Hermite ↓	`ExpEvolve`	10×20	0.39730 ± 0.06957	0.35841 ± 0.00503	0.35738 ± 0.00659	0.35117 ± 0.00203	0.35089 ± 0.00238
	`ExpEvolve`	20×10	2.81478 ± 2.14897	1.51139 ± 1.88727	0.40779 ± 0.06331	0.35505 ± 0.00285	0.35473 ± 0.00314
	`SCS`	10×20	1.14201 ± 1.25089	0.35956 ± 0.00503	0.35956 ± 0.00503	0.35956 ± 0.00503	0.35956 ± 0.00503
	`SCS`	20×10	2.42124 ± 2.13014	0.96193 ± 0.95033	0.39580 ± 0.05769	0.39431 ± 0.05890	0.39431 ± 0.05890
	`ShinkaEvolve`	–	4.72792 ± 3.78744	4.72791 ± 3.78744	4.72788 ± 3.78741	4.72783 ± 3.78749	4.72779 ± 3.78756

Table 9: Numbers of evaluated programs that `ExpEvolve` and `SCS` need to reach the final best score of `SCS`, averaged over 3 seeds.

Task	Config.	Final of `SCS`	$n_{\texttt{SCS}}$	$n_{\texttt{ExpEvolve}}$	$n_{\texttt{SCS}}/n_{\texttt{ExpEvolve}}$
Hadamard	10×20	0.8349	186	10	18.6
Hadamard	20×10	0.6641	131	5	26.2
SumDiff	10×20	1.11608	180	81	2.2
SumDiff	20×10	1.11087	181	97	1.9
Heilbronn	10×20	0.0336	199	52	3.8
Heilbronn	20×10	0.0313	169	27	6.3
Min overlap	10×20	0.38107	141	84	1.7
Min overlap	20×10	0.38156	144	13	11.1