MESH: CO-EVOLVING BELIEFS AND SOLUTIONS FOR RECURSIVELY SELF-IMPROVING DISCOVERY

Anonymous authors
Paper under double-blind review

ABSTRACT

Language-model agents increasingly automate discovery by iteratively proposing, implementing, and evaluating candidate solutions. Each execution consumes part of a limited budget and reveals information about the task, and using this information to choose subsequent executions enables recursive self-improvement. An evaluation outcome not only measures the quality of a candidate but also provides evidence about the hypothesis that motivated it. However, existing tree-search agents rely primarily on scores and overlook how outcomes support or challenge the underlying hypotheses. To address this limitation, we propose MESH, a framework in which candidate solutions co-evolve with the explicit beliefs that direct them. At the high level, an agent maintains a global belief state of working hypotheses, each specifying a target candidate, a mechanism grounded in prior evidence, an intervention, and an expected outcome. At the low level, a search process develops candidate solutions. Beliefs compete for the execution budget according to the improvements they have produced. Each selected belief directs a bounded sequence of executions that iteratively apply and refine its intervention, allowing a promising idea to develop beyond a weak initial implementation. The high-level agent periodically revises all beliefs jointly based on the shared search history, so that even failed executions can refine, retarget, or replace hypotheses across the search. The resulting belief-guided recursive loop operates within a single discovery run and is grounded in realized progress rather than the agent's self-assessment. Experiments on 19 tasks from MLGym and MLE-bench show that MESH attains the best score on 17 of them, including all eight MLE-bench competitions, when compared with linear, tree-structured, and graph-structured search controllers under the same worker models and execution budgets. Further analyses show that belief-directed executions improve the incumbent three times as often as score-driven ones, with the gap widening as score-driven search stalls, and that freezing the beliefs forfeits up to 59% of the gain.

1 INTRODUCTION

Language-model agents increasingly automate discovery: they iteratively propose, implement, and evaluate artifacts such as programs (Romera-Paredes et al., 2024; Novikov et al., 2025), algorithmic heuristics (Zheng et al., 2025), and computation pipelines (Jiang et al., 2025; Chan et al., 2025; Nathani et al., 2025; Yamada et al., 2025). Each execution consumes part of a finite budget and reveals something about the task. An agent then turns this knowledge into better choices of what to execute next and learns more from those choices, resulting in recursive self-improvement (Schmidhuber, 2007). This allows an agent to evolve, for example by updating its code (Zelikman et al., 2024; Zhang et al., 2026), prompts (Agrawal et al., 2026), or design (Hu et al., 2025; Xiong et al., 2026), and to evaluate each update across more tasks.

An evaluation outcome serves two roles: it quantifies candidate quality through a score, and it provides evidence about the hypothesis that motivated the candidate. Existing tree-search agents primarily rely on evaluation scores to guide this self-improvement process, for example by selecting which candidates to develop (Jiang et al., 2025; Inoue et al., 2025; Toledo et al., 2025; Chen et al., 2026). They typically overlook how outcomes support or challenge the underlying hypotheses and how this evidence can guide subsequent exploration. A failed execution, for instance, may refute a working hypothesis, expose an implementation limitation, or indicate that an intervention requires

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.

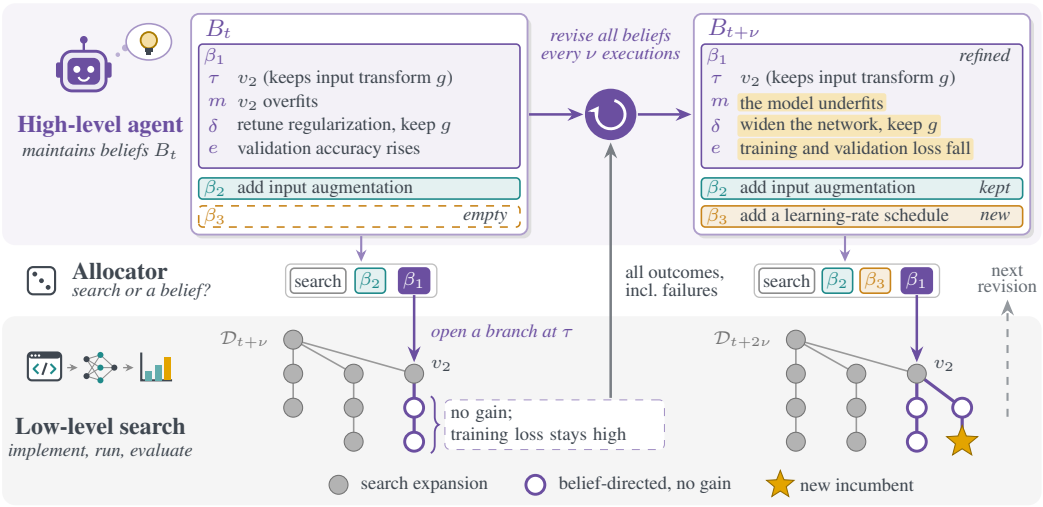


Figure 1: One round of the recursion in MESH. The high-level agent's beliefs (top), each with a target τ, mechanism m, directive δ, and expected outcome e, direct the low-level search (bottom). Every ν executions, all outcomes, including the failed branch of β_1, revise the beliefs, and the refined β_1 then yields a new incumbent.

further refinement, distinctions that an evaluation score alone does not capture. First, an initial implementation of a promising idea may receive a low score yet benefit from targeted refinement, such as debugging operations that repair program crashes (Jiang et al., 2025; Toledo et al., 2025). Second, evidence from one execution branch or agent, such as a drop in validation accuracy after removing an input transformation, can shape the refinement of candidates and working hypotheses across the broader discovery process. Cross-branch memory (Du et al., 2026; Chen et al., 2026), curated pools and trees of ideas and tested hypotheses (Yan et al., 2026; Weng et al., 2026; Jin et al., 2026), and evolved search strategies (Liu et al., 2026) provide relevant mechanisms. What remains missing is a controller that allocates the search budget through an explicit belief about the system, which compactly represents the current understanding, working hypotheses, and expected outcomes. Each hypothesis then earns further budget only through demonstrated improvements and is revised as evidence accumulates across the search.

We propose MESH, which closes the loop through explicit beliefs about the discovery process, so that candidate solutions co-evolve with the beliefs that direct them (Figure 1). MESH operates at two levels. At the high level, an agent maintains a global belief state of working hypotheses, each naming a target candidate in the search history, a mechanism grounded in prior evidence, an intervention, and an expected outcome. At the low level, a search process develops candidates, with workers implementing changes and an evaluator running and scoring them. Beliefs connect the two levels and govern the search. A selected belief directs a bounded sequence of executions that apply and refine its intervention, letting a promising idea develop beyond a weak first implementation, and each belief is pursued according to the improvements it has produced. Periodically, the agent jointly revises its beliefs from the shared search history, so that any outcome, including one that does not improve the incumbent, can refine, retarget, or replace hypotheses across the search.

The result is a belief-guided recursive loop within a single discovery run, in which MESH refines its explicit beliefs about the task and uses them to direct the search that produces the next evidence. The recursion is grounded in realized progress rather than the agent's self-assessment. Beliefs earn credit only for the improvements they produce, repeatedly unsuccessful beliefs are retired, and the low-level search keeps expanding candidates that no belief selects.

We evaluate MESH on 19 tasks from MLGym (Nathani et al., 2025) and MLE-bench (Chan et al., 2025) against linear, tree-structured, and graph-structured search controllers (Karpathy, 2026; Kocsis & Szepesvári, 2006; Toledo et al., 2025; Du et al., 2026; Inoue et al., 2025) under the same workers and execution budgets. MESH attains the best score on 17 tasks and raises the interquartile mean of normalized scores from 0.445 to 0.463 over the strongest baseline. On MLE-bench, belief-

directed executions improve the incumbent three times as often as the same search without beliefs, a ratio growing from 2.2 in the first half of the budget to 5.4 in the second, and 16 of 41 improving branches succeed only after a failed first execution. Removing any module forfeits more than 40% of the gain over the low-level search on some task, and freezing beliefs forfeits up to 59%.

Our contributions are as follows:

- We propose explicit belief updates driven by discovery search, capturing what a run learns about the task, including from failed executions, as revisable working hypotheses.
- We propose MESH, which integrates belief updates and search in a unified discovery process: beliefs direct which candidates the search develops and how, while search outcomes revise the beliefs.
- We show that MESH attains the best score on 17 of 19 MLGym and MLE-bench tasks, and that removing any one of its modules forfeits over 40% of its advantage on some task.

2 RELATED WORK

Recursive self-improvement. Recursive self-improvement refers to systems that improve the mechanisms of their own improvement (Schmidhuber, 2007). LLM systems pursue it by rewriting their own code (Zelikman et al., 2024; Zhang et al., 2026) or searching over agent designs, prompts, and memory (Hu et al., 2025; Agrawal et al., 2026; Xiong et al., 2026), optimizing performance across many task instances. EvoX evolves its search strategy with the candidate solutions within a single run (Liu et al., 2026). MESH instead evolves an explicit model of the task within a run, whose hypotheses decide which candidate to develop and what to change, and state why.

Search for LLM-driven discovery. Tree search structures the reasoning and acting of LLMs (Yao et al., 2023; Zhou et al., 2024), and learned policies decide when to expand, branch, or backtrack (Li, 2025). In discovery, LLM-guided evolution searches over programs (Romera-Paredes et al., 2024; Novikov et al., 2025), Monte Carlo tree search over heuristics (Zheng et al., 2025), and research agents over trees or graphs of programs and experiments (Jiang et al., 2025; Yamada et al., 2025; Toledo et al., 2025; Chen et al., 2026; Du et al., 2026). Some controllers adapt the search itself, widening or deepening from scores (Inoue et al., 2025) or backtracking when progress stalls (Yan et al., 2026). In MESH, explicit hypotheses choose both the candidate and the change.

Hypotheses and experience. Agents reuse experience as context via verbal reflections (Shinn et al., 2023), records of past plans and outcomes (Du et al., 2026), lessons shared across branches (Chen et al., 2026), and rewinds that keep lessons from discarded attempts (Zhuang et al., 2026). Closer to MESH, some systems manage hypotheses explicitly. PACEvolve curates a pool of ideas and tested hypotheses (Yan et al., 2026), Co-Scientist ranks evolving hypotheses in an Elo tournament (Gottweis et al., 2026), DeepScientist selects hypotheses by Bayesian optimization over stored findings (Weng et al., 2026), and Arbor refines a persistent tree of hypotheses and insights (Jin et al., 2026). In MESH, each belief targets any candidate, draws budget from its slot's realized improvement, receives a bounded multi-step commitment, and is revised jointly with other beliefs.

3 PRELIMINARIES

3.1 PROBLEM FORMULATION

We consider discovery tasks that fix a space $\mathcal{X}$ of executable artifacts, such as programs (Romera-Paredes et al., 2024), heuristics (Zheng et al., 2025), or machine-learning pipelines (Chan et al., 2025), and an evaluator $\mathcal{E}$. Evaluating x_v returns a score $s_v \in \mathbb{R}$, where higher is better, and diagnostic feedback f_v such as execution errors or task-specific measurements. An execution is *valid* if the artifact runs to completion and is scored, and invalid executions, such as crashes, receive a task-defined score. For a budget of T executions, with V_t the candidates evaluated in the first t, the objective is the expected best score at budget exhaustion,

$$J_T = \mathbb{E}[F_T], \qquad F_t = \max_{v \in V_t} s_v, \tag{1}$$

where the expectation is taken over generation and evaluation, and F_t is the *incumbent* score.

Algorithm 1 MESH: Co-Evolving Beliefs and Solutions

Require: initial artifact x_0; budget T; low-level search; workers Π; evaluator $\mathcal{E}$; high-level agent $\mathcal{U}$; hyperparameters $K, L, \nu, \rho, N_{\text{retire}}$

Evaluate x_0 as the root v_0; $B \leftarrow K$ empty slots; $t \leftarrow 1$; $k_{\text{open}} \leftarrow \perp$ ▷ slot of the open branch
while $t < T$ **do**
  if $k_{\text{open}} = \perp$ **then**
    Choose $k \in \mathcal{A}_t$ by Thompson sampling ▷ Eq. (3)
    if $k = 0$ **then** $v \leftarrow$ parent chosen by the low-level search; $\delta \leftarrow \emptyset$
    else $k_{\text{open}} \leftarrow k$; $P \leftarrow (\,)$; $F_{\text{open}} \leftarrow F_t$; $v \leftarrow \tau_k$; $\delta \leftarrow \delta_k$ ▷ open a branch
  else
    $v \leftarrow$ last node of P; $\delta \leftarrow \delta_{k_{\text{open}}}$ ▷ current directive of the slot
  end if
  $x_u \sim \Pi(\cdot \mid x_v, c, \delta)$; $(s_u, f_u) \leftarrow \mathcal{E}(x_u)$
  if $k_{\text{open}} = \perp$ **then**
    Update the statistics of action 0 with $r_0 = [s_u - F_t]_+$ ▷ Eq. (2)
  else
    Append u to P
    if (u is valid and $s_u > F_{\text{open}}$) **or** $|P| = L$ **then**
      Credit slot k_{open} with $R(P)$ over $|P|$ executions
      Retire the slot if $S_{k_{\text{open}}} = 0$ after N_{retire} branches; $k_{\text{open}} \leftarrow \perp$
    end if
  end if
  Add u with its provenance to $\mathcal{D}$ as a child of v; $t \leftarrow t + 1$
  if ν divides $t - 1$ **then** $B \leftarrow \mathcal{U}(B, \mathcal{C}_t)$ ▷ joint revision, Eq. (4)
end while
return $\arg\max_{v \in V_T} s_v$

3.2 DISCOVERY AS SEARCH

Each artifact has a single parent, so the history forms a rooted tree $\mathcal{D}_t$ over V_t, as in tree-search agents (Yao et al., 2023; Zhou et al., 2024; Jiang et al., 2025). Node v stores (x_v, s_v, f_v), and the root v_0 is the evaluated initial artifact, so $V_1 = \{v_0\}$. Every node remains expandable, and repeated expansion yields alternative revisions (Inoue et al., 2025). A node's score measures the candidate itself, whereas its value for Eq. (1) lies in its unknown best descendant, so search control must decide both which candidate to develop and what change to attempt.

3.3 SEARCH CONTROL

Two LLM workers $\Pi = (\pi_{\text{sci}}, \pi_{\text{impl}})$ produce candidates, a scientist that proposes a modification and an implementer that realizes it, separating ideation from implementation as in research agents (Yamada et al., 2025). At each step, a *controller* selects a parent $v \in V_t$, optionally specifies an intervention directive δ, and assembles a bounded context c from the task description and history. The execution draws $x_u \sim \Pi(\cdot \mid x_v, c, \delta)$, evaluates $(s_u, f_u) \leftarrow \mathcal{E}(x_u)$, and adds u to $\mathcal{D}_t$ as a child of v. With workers and evaluator fixed, methods differ only in the controller. Score-driven controllers choose v by score and let the workers decide the change (Jiang et al., 2025; Inoue et al., 2025; Toledo et al., 2025; Chen et al., 2026), whereas in MESH explicit hypotheses choose both v and δ.

4 MESH: CO-EVOLVING BELIEFS AND SOLUTIONS

MESH reads every outcome twice, as a score that measures the candidate and as evidence that bears on the beliefs (Figure 1 and Algorithm 1). A high-level agent holds working hypotheses in a global belief state (Section 4.1). A selected belief decides which candidate the workers develop and what change they attempt, and otherwise a score-driven low-level search chooses the candidate (Section 4.2). All outcomes revise the beliefs (Section 4.3), closing a recursive loop in which beliefs and solutions co-evolve (Section 4.4). Appendix A gives the remaining details.

4.1 BELIEFS AS WORKING HYPOTHESES

The belief state is the run's explicit account of what it has learned about its task. Each belief is a working hypothesis $\beta = (\tau, m, \delta, e)$ about how to improve an existing candidate. The target τ references a candidate in $\mathcal{D}_t$, the mechanism m is a natural-language interpretation of recorded observations that cites the candidates supporting it, the directive δ specifies an intervention, and the expected outcome e states the change in score or diagnostic behavior that δ is predicted to produce. In Figure 1, variants that remove an input transformation g have lost validation accuracy, so β_1 targets an earlier candidate v_2 that keeps g (τ), hypothesizes that v_2 overfits (m), proposes to retune regularization while keeping g (δ), and predicts that validation accuracy rises (e).

The pair (τ, δ) determines what the search executes, and (m, e) determines how its outcomes are read. A score shows whether a candidate improved but not why, whereas an outcome compared with a stated expectation bears on the mechanism behind it, so each belief-directed execution tests an explicit prediction. With diagnostic feedback, the test separates three cases that a score conflates. A crash exposes an implementation limitation, which the rest of the branch addresses. An outcome that moves toward e without improving the incumbent calls for refining the implementation within the branch, or δ at the next revision. An outcome whose diagnostics contradict e refutes m, and revision rewrites m, retargets the belief, or replaces it.

The global belief state $B_t = (\beta_1, \ldots, \beta_K)$ occupies K persistent slots, with $\beta_k = (\tau_k, m_k, \delta_k, e_k)$ the belief held in slot k. Every node records its provenance, the slot and belief that produced it, so each outcome is read against the prediction it tested. Beliefs are formed from a compact summary of the whole history and can target any candidate, so evidence from one branch can direct executions in another. $\mathcal{D}_t$ records what has been measured, and B_t what the agent concludes from it.

4.2 BELIEF-DIRECTED SEARCH

A selected belief makes both decisions of search control (Section 3.2). When no branch is open, the allocator chooses an action from $\mathcal{A}_t = \{0\} \cup \{k : \text{slot } k \text{ holds a belief}\}$. Action $k \geq 1$ opens a branch for slot k, a sequence of executions starting at τ_k and following δ_k. Action 0 delegates one execution to the low-level search, which picks the parent by score without a directive.

Bounded commitment. Judging an intervention by its first implementation undervalues ideas that need refinement. In Figure 1, the second branch of β_1 yields no gain in its first execution and a new incumbent in its second. Each branch therefore spans up to L consecutive executions. The first develops τ_k, and each later one develops the latest candidate of the branch with its feedback, so the workers repair or refine the same intervention rather than reimplement it. The branch closes once a valid candidate exceeds F_{open}, the incumbent score at opening, and the bound L keeps a failing intervention from consuming the budget. Belief-directed candidates enter the same tree, so the low-level search also builds on them.

Credit. A closed branch $P = (v^{(1)}, \ldots, v^{(\ell)})$ earns its best improvement over F_{open} among its valid members P_{val}, and an execution delegated through action 0 after t executions earns the improvement of its candidate u over F_t,

$$R(P) = \max(\{0\} \cup \{s_v - F_{\text{open}} : v \in P_{\text{val}}\}), \qquad r_0 = [s_u - F_t]_+, \tag{2}$$

where $[y]_+ = \max(y, 0)$. For each action k, the allocator tracks the number N_k of executions consumed, the number S_k of them that improved the incumbent, and the cumulative improvement Σ_k. Since a branch closes at its first improving execution, S_k/N_k is a per-execution success rate for beliefs and delegation alike. The statistics belong to the slot rather than to its current belief, so a refined belief inherits the record of the line of investigation it continues, and the agent cannot reset a poor record by rewording or replacing its hypothesis. Only retirement, triggered by realized outcomes, clears a record. A slot is retired once it completes N_{retire} branches without improvement ($S_k = 0$), and the next revision fills it with a new hypothesis.

Allocation. Budget follows realized improvement per execution. We instantiate the allocator with Thompson sampling over per-execution success rates (Thompson, 1933), scaling each sampled rate

by the mean improvement of an improving execution,

$$k_t = \arg\max_{k \in \mathcal{A}_t} \theta_k \bar{r}_k, \qquad \theta_k \sim \mathrm{Beta}(1 + S_k,\, 1 + N_k - S_k), \qquad \bar{r}_k = \frac{\rho + \Sigma_k}{1 + S_k}, \tag{3}$$

where the prior improvement $\rho > 0$ enters the mean as one pseudo-observation, so $\bar{r}_k = \rho$ before action k first improves the incumbent. The product $\theta_k \bar{r}_k$ is a sampled estimate of the expected improvement per execution, so a branch of ℓ executions competes on equal terms with ℓ delegated executions, and delegation is the reference against which every belief earns its budget. A belief whose branches stop improving the incumbent loses budget to delegation and to the other beliefs, but its chance of selection decays without vanishing, so a refined hypothesis in the same slot can recover, as β_1 does in Figure 1.

4.3 Evidence-Driven Belief Revision

Credit reads the score of an outcome, and revision reads its evidence. Every ν executions, the high-level agent $\mathcal{U}$ revises the belief state,

$$B_t^+ = \mathcal{U}\big(B_t^-, \mathcal{C}_t\big), \tag{4}$$

where B_t^- and B_t^+ are the states before and after revision. The revision context $\mathcal{C}_t$ summarizes the task, the slot statistics, and the search tree, with each node as a compact record of its parent, provenance, score, and diagnostic feedback. Slots start empty, so the first ν executions after the root are delegated and the first revision forms the initial beliefs from their outcomes. In a single pass over all K slots, the agent reads each outcome against the belief that produced it. For each slot, it keeps the belief, refines it by rewriting m, δ, or e, retargets it to another candidate, replaces it with a new hypothesis, or drops it, and it fills empty slots with new hypotheses. Because revision reads the shared history rather than a single slot's record, an outcome produced under one belief can revise several others, including their interpretation of earlier executions and their targets elsewhere in the tree. Every execution, including those that earn no credit, thus enters the evidence for revision.

Figure 1 illustrates the two readings of an outcome. The first branch of β_1 ends without improving the incumbent, so it earns no credit and looks unproductive to a score-only controller. Read against e, it is informative. Validation accuracy does not rise and the training loss stays high, contradicting overfitting and indicating underfitting. Revision keeps the target v_2, which the earlier evidence about g still supports, rewrites the mechanism as underfitting, changes the intervention to widening the network, and predicts that both losses fall. The refined belief then yields a new incumbent.

4.4 The Belief-Guided Recursive Loop

Belief-directed search and evidence-driven revision close a loop. Let B_i be the belief state produced by the i-th revision and $\mathcal{D}_i$ the history at that point, with B_0 the K empty slots and $\mathcal{D}_0 = \{v_0\}$. Writing $\mathcal{G}(B_i, \mathcal{D}_i)$ for the next ν executions, which the allocator distributes between the branches of beliefs in B_i and the low-level search, the history and the beliefs evolve as

$$\mathcal{D}_{i+1} = \mathcal{D}_i \cup \mathcal{G}(B_i, \mathcal{D}_i), \qquad B_{i+1} = \mathcal{U}\big(B_i,\, \mathcal{C}(\mathcal{D}_{i+1})\big), \tag{5}$$

where $\mathcal{C}(\cdot)$ builds the revision context, including the slot records, from the history. Substituting the first equation into the second shows that B_i enters the next revision twice, as the state being revised and through the evidence it chose to collect. Through this recurrence, MESH improves itself within a single discovery run. Sections 5.3 and 5.4 trace the recursion in individual runs and ablate its modules.

The recursion stays tied to progress because the agent's judgment decides only which hypotheses to hold, while realized improvement decides how much of the budget each slot receives (Eqs. (2) and (3)), and delegation keeps developing candidates that no belief selects.

5 Experiments

We ask whether MESH outperforms existing search controllers under matched execution budgets (Section 5.2), how the recursion between beliefs and search produces its gains (Section 5.3), and whether each module contributes (Section 5.4).

Table 1: Best-attempt test score on 11 MLGym tasks with 16 executions per run, in each task's native metric. *Baseline* is MLGym's starter program as measured by our adapter.

		Prior work					*Ours*	
Task (metric)	**Baseline**	**Linear**	**MCTS**	**AIRA**	**MLEvolve**	**AB-MCTS-A**	**MESH**	**Human Best**
House Prices (R^2 ↑)	0.880	0.912	0.907	0.910	0.915	0.925	**0.928**	0.990
House Prices-L1 (R^2 ↑)	0.794	0.885	0.879	0.893	0.897	0.888	**0.901**	0.990
CIFAR-10 (acc ↑)	0.498	0.613	0.671	0.659	0.675	0.678	**0.743**	0.994
CIFAR-10-L1 (acc ↑)	0.392	0.463	0.677	0.694	0.721	0.789	**0.831**	0.994
Fashion-MNIST (acc ↑)	0.848	0.883	0.895	0.892	0.913	0.923	**0.924**	0.968
Battle of the Sexes (payoff ↑)	1.018	1.323	1.286	1.197	1.407	**1.440**	1.246	1.667
Prisoner's Dilemma (payoff ↑)	2.370	2.374	2.381	2.403	2.419	2.371	**2.503**	3.000
Colonel Blotto (score ↑)	−0.245	0.024	0.031	0.029	0.034	0.033	**0.042**	0.500
MountainCar (reward ↑)	33.72	42.13	71.24	73.53	78.12	81.94	**85.83**	99.00
Breakout (reward ↑)	48.71	64.28	76.07	74.12	74.46	**81.04**	80.12	100.00
MetaMaze (reward ↑)	15.24	35.62	41.53	43.74	45.49	47.37	**50.13**	52.10

Table 2: Private-test score of the final submission on 8 MLE-bench competitions in each competition's metric. *Human Best* is the Kaggle winner and *Baseline* the starter program.

		Prior work					*Ours*	
Competition	**Baseline**	**Linear**	**MCTS**	**AIRA**	**MLEvolve**	**AB-MCTS-A**	**MESH**	**Human Best**
nomad2018 (RMSLE ↓)	0.120	0.071	0.061	0.063	0.059	0.054	**0.052**	0.051
aptos2019 (QWK ↑)	0.000	0.167	0.181	0.180	0.188	0.192	**0.297**	0.936
jigsaw-toxic (AUC ↑)	0.604	0.738	0.732	0.735	0.745	0.727	**0.747**	0.989
plant-pathology (AUC ↑)	0.537	0.557	0.560	0.563	0.575	0.576	**0.591**	0.984
kuzushiji (F1 ↑)	0.021	0.187	0.202	0.200	0.198	0.185	**0.208**	0.950
vesuvius ($F_{0.5}$ ↑)	0.165	0.205	0.200	0.192	0.195	0.202	**0.215**	0.831
champs (norm. LMAE ↑)	0.000	0.089	0.102	0.108	0.110	0.112	**0.114**	1.000
spaceship-titanic (acc ↑)	0.503	0.803	0.807	0.810	0.815	0.818	**0.819**	0.828

Table 3: Median, interquartile mean (IQM), mean, and optimality gap (Agarwal et al., 2021) of per-task normalized scores, on which the starter program scores 0 and the human best scores 1.

	All (19 tasks)				*MLGym (11)*				*MLE-bench (8)*			
Method	**Median**	**IQM**	**Mean**	**Opt. Gap↓**	**Median**	**IQM**	**Mean**	**Opt. Gap↓**	**Median**	**IQM**	**Mean**	**Opt. Gap↓**
Linear	0.291	0.263	0.303	0.697	0.292	0.296	0.293	0.707	0.179	0.199	0.317	0.683
MCTS	0.370	0.357	0.381	0.619	0.413	0.423	0.410	0.590	0.194	0.206	0.340	0.660
AIRA	0.340	0.349	0.381	0.619	0.368	0.405	0.413	0.587	0.192	0.208	0.338	0.662
MLEvolve	0.374	0.411	0.431	0.569	0.526	0.492	0.486	0.514	0.196	0.217	0.355	0.645
AB-MCTS-A	0.409	0.445	0.457	0.543	**0.625**	0.547	0.527	0.473	0.191	0.203	0.360	0.640
MESH	**0.436**	**0.463**	**0.490**	**0.510**	0.546	**0.548**	**0.559**	**0.441**	**0.259**	**0.253**	**0.395**	**0.605**

5.1 EXPERIMENTAL SETUP

Tasks and models. We evaluate on 11 MLGym tasks (Nathani et al., 2025) spanning regression, vision, game theory, and reinforcement learning, and 8 Kaggle competitions from MLE-bench (Chan et al., 2025). Each task supplies an evaluator and a starter program that roots every search tree. All methods share frozen workers, `gpt-5-mini` (OpenAI, 2025) on MLGym and Claude Sonnet 5 (Anthropic, 2026) on MLE-bench, and an annotator, so only the controller differs.

Baselines. We compare against controllers of each search structure. **Linear** (Karpathy, 2026) hill-climbs along a single trajectory, **MCTS** (Kocsis & Szepesvári, 2006) applies UCT over the observed scores with a fixed branching factor, **AIRA** (Toledo et al., 2025) routes crashed programs to a debugging operator, **MLEvolve** (Du et al., 2026) extends the tree to a graph with cross-branch references and a retrospective memory, and **AB-MCTS-A** (Inoue et al., 2025) Thompson-samples between widening and deepening at every node. MESH delegates executions to AB-MCTS-A and shares its tree, workers, and prompts, so their comparison isolates belief-directed control.

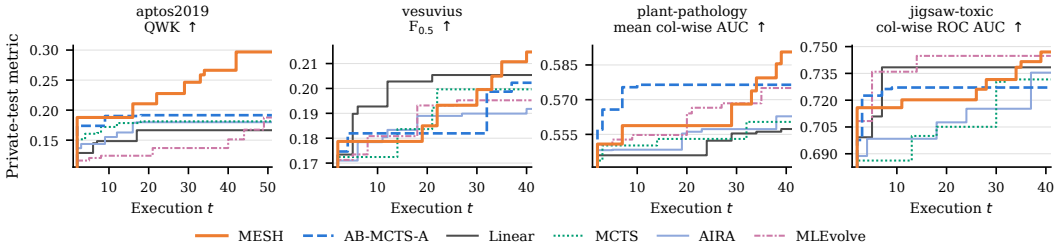


Figure 2: Incumbent score over the budget on four MLE-bench competitions, mean of three seeds.

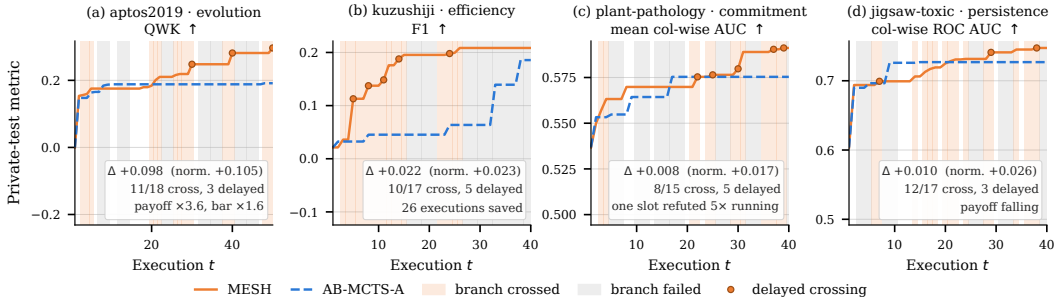


Figure 3: Incumbent score on four MLE-bench runs, with a shared seed for MESH and AB-MCTS-A. Orange and grey bands mark belief-directed branches that do and do not beat F_{open}, gaps mark delegated executions, and circles mark gains after a branch's first step.

Evaluation. Each method receives 16 executions per run on MLGym and 40 to 50 on MLE-bench, and Tables 1 and 2 report means over three seeds. Following each benchmark's protocol, we report the best test score within the budget on MLGym, its Best Attempt metric, and the private-test score of the best-validation candidate on MLE-bench. For aggregation, we normalize each score as $z = \text{clip}_{[0,1]}\left((\tilde{s}-\tilde{s}_0)/(\tilde{s}_{\text{human}}-\tilde{s}_0)\right)$ between the starter program and the human best (the published state of the art on MLGym and the Kaggle winner on MLE-bench), with all scores oriented so that higher is better, and report the median, interquartile mean (IQM), mean, and optimality gap (Agarwal et al., 2021). Orange marks the best automated method in every table and light orange the second best in Tables 1 to 3.

5.2 Main Results

MESH outperforms every search structure. MESH attains the best score on 17 of the 19 tasks, including all eight MLE-bench competitions (Tables 1 and 2), with a mean rank of 1.26 against 2.74 for AB-MCTS-A, and the best aggregate on every metric over all tasks and over MLE-bench (Table 3). Against AB-MCTS-A, belief-directed control raises the IQM from 0.445 to 0.463 overall and from 0.203 to 0.253 on MLE-bench. The two exceptions are Breakout, by 0.9 points, and Battle of the Sexes, whose opponent copies the row player's first move, so scores alone lead AB-MCTS-A to the optimal payoff of 1.440.

Search structure does not move the plateau. The largest gains, on Prisoner's Dilemma, CIFAR-10, and aptos2019, arise where the strongest score-driven controllers stall at a common level. On aptos2019, the five baselines reach a QWK of 0.167 to 0.192 against 0.297 for MESH, and across MLE-bench their median normalized scores span only 0.179 to 0.196 against 0.259. Changing the structure changes which baseline leads, AB-MCTS-A on 13 tasks and MLEvolve on four, but not this level. Nor is MESH bounded by its low-level search. AB-MCTS-A is the weakest baseline on Prisoner's Dilemma, jigsaw-toxic, and kuzushiji, where MESH is the strongest method. The bottleneck is what directs the search, not its structure.

Table 4: Ablation on MLGym. −Representation drops m and e, −Commitment sets $L = 1$, and −Evolution freezes beliefs after the fifth execution. Scores are mapped to $[0, 1]$ by each task's range.

Task	AB-MCTS-A	MESH	−Representation	−Commitment	−Evolution
Battle of the Sexes	0.7197	**0.7222**	0.7203	0.7214	0.7209
Colonel Blotto	0.5183	**0.6253**	0.6018	0.6251	0.5620
CIFAR-10-L1	0.7693	**0.8292**	0.7968	0.8037	0.8098

5.3 How the Recursion Improves the Search

The advantage of MESH builds up over the run. In Figure 2, score-driven controllers make most of their progress early, and AB-MCTS-A stops improving by execution 18 on three competitions. MESH makes 76% of its progress in the second half of the budget, against at most 48% for any baseline, and overtakes the strongest baseline after the midpoint on three competitions.

Beliefs earn most of the budget and succeed three times as often. Figure 3 isolates the high-level agent by comparing MESH with AB-MCTS-A under a shared seed. Belief-directed branches receive 139 of the 166 executions, and beyond the first execution 29.5% of them improve the incumbent, against 9.9% for AB-MCTS-A and 8.7% for the executions MESH delegates, despite a higher incumbent to beat. They produce at least 84% of the incumbent's subsequent rise in every run.

Commitment turns failed first attempts into gains. The first execution of a branch beats F_{open} in 25 of 67 branches. After it fails, later executions still succeed 22% of the time, more than twice the rate of AB-MCTS-A, and yield 16 of the 41 improving branches, so refining a weak first implementation outperforms returning the execution to score-driven search.

Revision sustains progress after score-driven search stalls. Relative to AB-MCTS-A, belief-directed executions succeed 2.2 times as often in the first half of the budget and 5.4 times as often in the second. Each revision reads the evidence that earlier beliefs chose to collect (Eq. (5)). On aptos2019, the last three improving branches gain 3.6 times as much as the first three against an incumbent 1.6 times higher, and the largest gain follows seven consecutive revisions that each record a refuted hypothesis. Freezing the beliefs forfeits much of this gain (Section 5.4).

5.4 Ablations

Each module matters where the task needs it. In Table 4, the full method scores highest on every task, and removing any module forfeits more than 40% of its gain over AB-MCTS-A on some task. Which module matters depends on how an idea is tested. On Colonel Blotto, where the first implementation of a strategy already tests it, setting $L = 1$ costs 0.2% of the gain, dropping m and e costs 22%, and freezing the beliefs after the fifth execution costs 59%. On CIFAR-10-L1, where pipeline changes need repair and tuning across executions, these ablations cost 43%, 54%, and 32%. Battle of the Sexes, with all arms within 0.003, does not separate the modules.

6 Conclusion

Each execution in a discovery run yields a score for the candidate and evidence about the hypothesis behind it. Score-driven controllers read only the score, whereas MESH reads both and lets candidate solutions co-evolve with explicit beliefs about the task. Beliefs decide which candidates the search develops and how, bounded commitment carries multi-step interventions through, and every outcome, including failed executions, revises the beliefs. With budget tied to measured improvement, this recursion lets the run improve its own search as evidence accumulates. On 19 MLGym and MLE-bench tasks, MESH attains the best score on 17 and moves past the plateau where score-driven controllers of every structure stall. Belief-directed executions succeed three times as often as score-driven ones, and removing mechanism-level hypotheses, commitment, or revision forfeits 43% to 59% of the advantage on the task where each matters most. Belief quality depends on the underlying language model, and expected outcomes are qualitative rather than calibrated. Extending the recursion across runs, tasks, and domains is a natural next step.

AI USE STATEMENT

In this work, we used generative AI tools for research ideation and execution, namely to propose or refine hypotheses, to help develop the conceptual framework of the method and its formalization, to design or provide feedback on the methodology and experiments, to implement the methods and run the experiments, and to analyze and interpret the results. We have not used generative AI tools to generate synthetic datasets or to clean or reformat datasets, and formulating mathematical claims, proving them, translation, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to draft sections of the paper and to edit the paper for readability. We have reviewed all AI-assisted work. The reported numbers were recomputed from the experimental results, and every reference was verified against its original publication. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Lakshya A. Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J. Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, pp. 8479–8565, 2026. URL https://openreview.net/forum?id=RQm2KQTM5r.

Anthropic. System card: Claude Sonnet 5, 2026. URL https://www.anthropic.com/claude-sonnet-5-system-card.

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 *International Conference on Learning Representations*, pp. 50466–50494, 2025. URL https://openreview.net/forum?id=6s5uXNWGIh.

Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. MARS: Modular agent with reflective search for automated AI research. In *International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=qWE13uu99a.

Shangheng Du, Xiangchao Yan, Jinxin Shi, Zongsheng Cao, Shiyang Feng, Zichen Liang, Boyuan Sun, Tianshuo Peng, Yifan Zhou, Xin Li, Jie Zhou, Liang He, Bo Zhang, and Lei Bai. MLEvolve: A self-evolving framework for automated machine learning algorithm discovery. *arXiv preprint arXiv:2606.06473*, 2026. doi: 10.48550/arXiv.2606.06473.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Petar Sirkovic, Artiom Myaskovsky, Grzegorz Glowaty, Felix Weissenberger, Alessio Orlandi, Dan Popovici, Anil Palepu, Keran Rong, Ryutaro Tanno, Khaled Saab, Fan Zhang, Jacob Blum, Andrew Carroll, Kavita Kulkarni, Nenad Tomašev, Dina Zverinski, Ivor Rendulic, Elahe Vedadi, Florian Hasler, Luka Rimanic, Marina Boia, Ivan Budiselic, Ben Feinstein, Mathias Bellaiche, Tom Sheffer, Jan Freyberg, Jeremy Ratcliff, Ottavia Bertolli, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Yuan Guan, Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R. D. Costa, José R. Penadés, Gary Peltz, Yossi Matias, James Manyika, Demis Hassabis, Yunhan Xu, Pushmeet Kohli, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Accelerating scientific discovery with Co-Scientist. *Nature*, 655(8122):487–496, 2026. doi: 10.1038/s41586-026-10644-y.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, pp. 21344–21377, 2025. URL https://openreview.net/forum?id=t9U3LW7JVX.

Yuichi Inoue, Kou Misaki, Yuki Imajuku, So Kuroki, Taishi Nakamura, and Takuya Akiba. Wider or deeper? Scaling LLM inference-time compute with adaptive branching tree search. In *Advances in Neural Information Processing Systems*, volume 38, pp. 35448–35484, 2025. URL https://openreview.net/forum?id=jAsr5GHt3P.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. AIDE: AI-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025. doi: 10.48550/arXiv.2502.13138.

Jiajie Jin, Yuyang Hu, Kai Qiu, Qi Dai, Chong Luo, Guanting Dong, Xiaoxi Li, Tong Zhao, Xiaolong Ma, Gongrui Zhang, Zhirong Wu, Bei Liu, Zhengyuan Yang, Linjie Li, Lijuan Wang, Hongjin Qian, Yutao Zhu, and Zhicheng Dou. Toward generalist autonomous research via hypothesis-tree refinement. *arXiv preprint arXiv:2606.11926*, 2026. doi: 10.48550/arXiv.2606.11926.

Andrej Karpathy. autoresearch: AI agents running research on single-GPU nanochat training automatically. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch.

Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293, Berlin, Heidelberg, 2006. Springer. doi: 10.1007/11871842_29.

Yang Li. Policy guided tree search for enhanced LLM reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 35710–35730. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25bv.html.

Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Qiuyang Mang, Zhifei Li, Ashwin Naren, Ethan Boneh, Audrey Cheng, Alexander Du, Melissa Z. Pan, Kurt Keutzer, Alvin Cheung, Koushik Sen, Alexandros G. Dimakis, Matei Zaharia, and Ion Stoica. EvoX: Meta-evolution for automated discovery. In *Conference on Language Modeling*, 2026. URL https://colm.cc/virtual/2026/poster/1849.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Mikhail Plekhanov, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Nicolaus Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. MLGym: A new framework and benchmark for advancing AI research agents. In *Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=ryTr83DxRq.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025. doi: 10.48550/arXiv.2506.13131.

OpenAI. GPT-5 system card, 2025. URL https://cdn.openai.com/gpt-5-system-card.pdf.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024. doi: 10.1038/s41586-023-06924-6.

Jürgen Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In Ben Goertzel and Cassio Pennachin (eds.), *Artificial General Intelligence*, Cognitive Technologies, pp. 199–226. Springer, Berlin, Heidelberg, 2007. doi: 10.1007/978-3-540-68677-4_7.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933. doi: 10.1093/biomet/25.3-4.285.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, Andrei Lupu, Roberta Raileanu, Tatiana Shavrina, Kelvin Niu, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H. Miller, Abhishek Charnalia, Derek Dunfield, Carole-Jean Wu, Pontus Stenetorp, Nicola Cancedda, Jakob Nicolaus Foerster, and Yoram Bachrach. AI research agents for machine learning: Search, exploration, and generalization in MLE-bench. In *Advances in Neural Information Processing Systems*, volume 38, pp. 35309–35348, 2025. URL https://openreview.net/forum?id=RwfrdKSgCE.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. DeepScientist: Advancing frontier-pushing scientific findings progressively. In *International Conference on Learning Representations*, pp. 47981–48037, 2026. URL https://openreview.net/forum?id=cZFgsLq8Gs.

Yiming Xiong, Shengran Hu, and Jeff Clune. Learning to continually learn via meta-learning agentic memory designs. *arXiv preprint arXiv:2602.07755*, 2026. doi: 10.48550/arXiv.2602.07755.

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. doi: 10.48550/arXiv.2504.08066.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, Chi Wang, Ed H. Chi, Fernando Pereira, Wang-Cheng Kang, Derek Zhiyuan Cheng, and Beidou Wang. PACEvolve: Enabling progress-aware consistent evolution. *arXiv preprint arXiv:2601.10657*, 2026. doi: 10.48550/arXiv.2601.10657.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of Thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 11809–11822, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=46Zgqo4QIU.

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*, pp. 104223–104294, 2026. URL https://openreview.net/forum?id=pUpzQZTvGY.

Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. Monte Carlo tree search for comprehensive exploration in LLM-based automatic heuristic design. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 78338–78373. PMLR, 2025. URL https://proceedings.mlr.press/v267/zheng25o.html.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

Yu Zhuang, Kefei Chen, Yitong Duan, Shuxin Zheng, Jian Li, and Xu-Yao Zhang. AgentRewind: Recoverable execution for long-horizon LLM agents. *arXiv preprint arXiv:2608.14380*, 2026. doi: 10.48550/arXiv.2608.14380.

A Implementation Details

This appendix completes Section 4: the validation applied to revised beliefs, how a belief-directed execution is issued to the low-level search, invalid executions, the provenance each node records, the ablation arms, and the models. Notation is that of Section 4.

A.1 Validation of revised beliefs

A revision returns a proposal for each of the K slots, and each proposal is checked before it is installed. The checks are syntactic and cheap, and a proposal that fails one is discarded rather than repaired, so a slot keeps the belief it already held:

1. **The target must exist.** τ must be the identifier of a node in $\mathcal{D}_t$. A belief naming a candidate that was never evaluated has no restart point.
2. **The hypothesis must be stated.** Both m and δ must be non-empty: a belief has to say what it concludes from the evidence and what it will do about it.
3. **Dropping is explicit.** A proposal may instead mark its slot as dropped, which empties the slot, closes its open branch if it has one, and removes the slot from $\mathcal{A}_t$ until a later revision fills it. Its statistics are untouched, as in Section 4.2.

Installing is in place: the slot's (τ, m, δ, e) are overwritten and (N_k, S_k, Σ_k) are left as they are, so a refined belief continues the record of the line of investigation it inherits. Each revision is also recorded whole, with the proposal, the slots it formed, rewrote, or dropped, and the agent's stated reason, so that the belief state at any execution can be reconstructed from the log alone.

A.2 Issuing a belief-directed execution

Action 0 is the low-level search running unmodified: it selects the parent and receives an empty directive. Action $k \geq 1$ has to expand a node the low-level search did not choose, which MESH does through the search's own interfaces rather than around them. Given the target τ_k, it

1. instantiates the per-node state the search maintains for τ_k, creating it if τ_k has not been expanded before;
2. opens a trial on τ_k labeled with slot k;
3. runs $x_u \sim \Pi(\cdot \mid x_{\tau_k}, c, \delta_k)$ and evaluates it;
4. reports (u, s_u) back to the search, which back-propagates the outcome.

After step 4 the new node is an ordinary member of the search's own tree, so the statistics the low-level search keeps stay valid over the whole run and every candidate, whoever directed it, remains available for the search to develop further. Subsequent executions of an open branch repeat the same procedure from the branch's latest candidate under the slot's current directive, carrying the candidate's diagnostic feedback, so the workers refine one intervention rather than reimplement it. A revision during an open branch therefore takes effect at the branch's next execution.

A.3 Invalid executions and tie-breaking

An invalid execution receives the task-defined score of Section 3.1 and is recorded in $\mathcal{D}_t$ like any other node, so it enters the revision context and the diagnostic feedback it carries is what the rest of the branch addresses. It does not end a branch early: the improvement condition of Section 4.2 requires a *valid* candidate above F_{open}, and an invalid execution contributes nothing to $R(P)$, which is a maximum over the valid members of P. A branch whose executions are all invalid therefore runs to L and is credited zero, the same outcome as a branch that ran and did not improve.

A branch opened near the end of a run is bounded by the executions that remain, so the last branch of a run may be shorter than L.

Ties in Eq. (3) are broken by slot index.

A.4 PROVENANCE

Every node in $\mathcal{D}_t$ records the action that produced it, the slot if that action was belief-directed, and the belief in force at the time, so an outcome is always read against the prediction it tested. For every execution, each run also logs the action set and the sampled values that produced the choice. For every closed branch, it logs the slot, member nodes, F_{open}, $R(P)$, and stop reason, and for every revision, the full record described above. All analyses in this paper are computed from these logs.

A.5 ABLATION ARMS

The three arms of Section 5.4 are configurations of the same loop, so no arm runs a different code path:

Arm	Component removed	Change
−Representation	mechanism and expected outcome (Section 4.1)	m and e carry no content
−Commitment	bounded commitment (Section 4.2)	$L = 1$
−Evolution	belief revision (Section 4.3)	no revision after execution 5

−Representation fills m and e with fixed placeholders, which pass the validation of Section A.1. The arm therefore differs from MESH in exactly one respect: no mechanism and no expected outcome enter the revision context, while (τ, δ), allocation, bounded commitment, and credit are unchanged.

Each entry of Table 4 is the best of three seeds, mapped to $[0, 1]$ by the range of its task. Battle of the Sexes payoffs are divided by 2, a Colonel Blotto score x becomes $(x + 1)/2$, and accuracy is unchanged.

A.6 WORKERS AND MODELS

$\Pi = (\pi_{\text{sci}}, \pi_{\text{impl}})$, the evaluator $\mathcal{E}$, and the annotator that summarizes feedback are identical across all methods, so only the controller differs. The high-level agent $\mathcal{U}$ that performs revision is an additional call made once every ν executions and is the only language-model call MESH adds. Allocation makes no language-model call: the belief state proposes and Eq. (3) chooses, so a slot's record measures the belief rather than a second model's preference for it.

B BENCHMARKS AND TASKS

The 19 tasks of Section 5.1 come from two benchmarks that pose the same problem in different forms. Both supply an executable artifact space and an evaluator, and every method is scored by the benchmark's own evaluator.

B.1 MLGYM

MLGym (Nathani et al., 2025) packages machine-learning research tasks as self-contained directories. Each configuration supplies a dataset, an evaluation script that the agent may read but not modify, and a published baseline score to beat, and a starter program is the root of the search tree. The agent edits that program, and the evaluator then runs it and returns the task's own metric together with diagnostic output. Scores are returned during the run, so the reported quantity is the best score reached within the budget, the Best Attempt metric of Nathani et al. (2025) (Section 5.1).

The 11 configurations cover regression, vision, game theory, and reinforcement learning. We group them by what the agent hands back, which determines the form a candidate takes:

- **Prediction tasks** hand back a submission file of predictions on the test split. Metrics are the usual supervised ones.
- **Game-theory tasks** hand back a Python strategy that is played repeatedly against a fixed opponent. The score is the mean payoff over the match, so the object being optimized is a policy rather than a model.

- **Reinforcement-learning tasks** hand back trained model weights for a `gymnax` environment. The evaluator rolls the policy out and reports mean episode reward, so training cost is part of the task.

Table 5: The 11 MLGym configurations, in the order of Table 1.

Task (Table 1)	MLGym configuration	Family	Metric
House Prices	`regressionKaggleHousePrice`	prediction	$R^2 \uparrow$
House Prices-L1	`regressionKaggleHousePriceL1`	prediction	$R^2 \uparrow$
CIFAR-10	`imageClassificationCifar10`	prediction	accuracy ↑
CIFAR-10-L1	`imageClassificationCifar10L1`	prediction	accuracy ↑
Fashion-MNIST	`imageClassificationFMnist`	prediction	accuracy ↑
Battle of the Sexes	`battleOfSexes`	game	mean payoff ↑
Prisoner's Dilemma	`prisonersDilemma`	game	mean payoff ↑
Colonel Blotto	`blotto`	game	score ↑
MountainCar	`rlMountainCarContinuous`	RL	mean reward ↑
Breakout	`rlBreakoutMinAtar`	RL	mean reward ↑
MetaMaze	`rlMetaMazeMisc`	RL	mean reward ↑

Prediction tasks. **House Prices** is Kaggle's housing-price regression over tabular property records, scored by R^2 on the test split. **CIFAR-10** and **Fashion-MNIST** are ten-class image classification, scored by accuracy, on natural color images and on grayscale clothing images respectively. The three differ in what a candidate can usefully change: feature engineering and model choice dominate on House Prices, architecture and training schedule on the two image tasks.

Game-theory tasks. In **Battle of the Sexes** and **Prisoner's Dilemma** the agent writes the row player's strategy for a repeated two-action game and is scored by its mean payoff against a fixed opponent. In **Colonel Blotto** the agent allocates a fixed number of soldiers across several battlefields each round. A battlefield goes to whoever commits more soldiers to it, and the score is the mean per-round outcome. These tasks admit no gradient signal at all: a candidate is a decision rule, and the only evidence about it is the payoff it earned and the opponent behavior visible in the match log.

Reinforcement-learning tasks. **MountainCar** is the continuous-action Mountain Car control problem: an underpowered car must build momentum to reach a hilltop, and action magnitude is penalized. **Breakout** is the MinAtar reduction of the Atari game, a small discrete grid with the original's reward structure. **MetaMaze** is navigation in a fixed grid-world maze whose goal location changes between episodes, so the policy has to find the goal and return to it within each episode rather than memorize one route. All three return mean episode reward over the evaluator's rollouts.

The L1 configurations. The two L1 configurations share the dataset, the evaluator, and the published baseline score of House Prices and CIFAR-10, and their search starts from a weaker program, whose score is the Baseline of Table 1. Including both variants lets the same dataset test a run from two starting points.

B.2 MLE-BENCH

MLE-bench (Chan et al., 2025) turns Kaggle competitions into agent tasks. Each competition supplies the original data with a private test split, a constant sample submission, the competition's own metric, and the full leaderboard. As on MLGym, a starter program is the root of the search tree, and its score is the Baseline of Table 2. The agent produces a submission file, which the benchmark's official grader scores on the private test split. The search itself never sees that split: it selects among candidates using validation scores only, and the candidate with the best validation score is the one submitted (Section 5.1).

MLE-bench sorts its competitions into three complexity tiers by how much work a solution takes. The seven tiered competitions used here span all three, and spaceship-titanic comes from MLE-bench's development split.

Table 6: The 8 MLE-bench competitions, in the order of Table 2. Tier is MLE-bench's complexity split (dev marks its development split), and Teams the size of the competition's leaderboard. Table 2 reports champs in LMAE normalized so that the starter program scores 0 and the Kaggle winner 1.

Competition (Table 2)	Kaggle slug	Tier	Teams	Metric
nomad2018	`nomad2018-predict-transparent-conductors`	low	879	RMSLE ↓
aptos2019	`aptos2019-blindness-detection`	low	2929	QWK ↑
jigsaw-toxic	`jigsaw-toxic-comment-classification-challenge`	low	4539	ROC AUC ↑
plant-pathology	`plant-pathology-2020-fgvc7`	low	1318	ROC AUC ↑
kuzushiji	`kuzushiji-recognition`	medium	294	F1 ↑
vesuvius	`vesuvius-challenge-ink-detection`	high	1250	$F_{0.5}$ ↑
champs	`champs-scalar-coupling`	medium	2739	LMAE ↓
spaceship-titanic	`spaceship-titanic`	dev	2288	accuracy ↑

Tabular and molecular prediction. **nomad2018** asks for the formation energy and bandgap of transparent conductor candidates from their crystal structure, scored by the mean column-wise root-mean-squared logarithmic error. **champs** asks for the scalar coupling constant between listed pairs of atoms in a molecule, over 4.19M training pairs and 467,813 test pairs, scored by the competition's log mean absolute error. **spaceship-titanic** is binary classification on 7,823 tabular training rows and 870 test rows, scored by accuracy.

Text. **jigsaw-toxic** is multi-label toxicity classification over 159,571 training and 153,164 test comments, scored by the mean of the per-label ROC AUC. It is the only text task in the set, and its leaderboard is the largest of the eight.

Images. **aptos2019** grades diabetic retinopathy over 3,295 training and 367 test retinal photographs, scored by quadratic-weighted Cohen's kappa. **plant-pathology** classifies apple-leaf disease over 1,638 training and 183 test images, scored by the mean column-wise ROC AUC.

Detection and segmentation. **kuzushiji** requires finding *and* classifying every cursive Japanese character on a scanned book page: 3,244 training pages and 361 test pages at roughly 2404×3874 pixels, with 613,505 labeled character boxes over 4,113 classes. Its modified F1 counts a detection as correct only when the predicted point falls inside the ground-truth box *and* the label matches exactly, so localization and classification must both succeed. **vesuvius** is per-pixel ink segmentation on carbonized Herculaneum papyrus: each fragment is a three-dimensional X-ray surface volume of 65 grayscale slices, with two labeled fragments (up to 9506×14830 pixels) and one held out, and the output is a run-length-encoded binary ink mask scored by $F_{0.5}$. In both, a submission is the output of a multi-stage pipeline, so every stage has to run before any score exists.

C Worked Examples

The examples below are taken from MESH runs, and each shows one mechanism of Section 4. In the first, a crash is addressed by the rest of its branch (Section 4.1). In the second, a failed first execution is refined within its branch into a new incumbent (Section 4.2). In the third, one revision settles two slots from the shared history (Section 4.3). Quotations are verbatim from the run's revision records and candidate code, node identifiers are the ones written in the run, and slot `b`k is slot k of Section 4.1.

C.1 A crash, addressed by the rest of the branch

CIFAR-10. Slot `b1` opened a branch at the incumbent $F_{\text{open}} = 0.5412$. Its first execution, `n000005`, imported a vision library to obtain a pretrained backbone, and the candidate's own comment recorded the assumption it was making:

```
# Import torchvision here (part of PyTorch ecosystem
available in the environment)
```

The execution exited with a non-zero status and was therefore invalid. Under $L = 1$ the branch would have closed here, the slot would have been charged one unpaid execution, and the intervention

would have been indexed as a failure. Instead the branch remained open, and the revision that followed identified the cause:

> n000005's import bug and n000002/n000003 timeouts showed torchvision and heavy schedules are unreliable; n000001 is the best valid restart point.

The branch's second execution, n000006, ran from n000005 under the slot's revised directive and replaced the dependency:

```
# Note:  removed dependency on torchvision.models to use a
lightweight in-file ResNet
```

It scored 0.7276, crossing F_{open} by $R(P) = 0.1864$, the largest single improvement of that run. The hypothesis was never at fault. Only its first implementation was, and bounded commitment is what kept the distinction available.

C.2 A FAILED FIRST EXECUTION, REFINED WITHIN THE BRANCH

kuzushiji. Slot b1 held a detection-side hypothesis and opened a branch at $F_{\text{open}} = 0.0379$. Its first execution, n000003, scored 0.0158, well below F_{open}, and the revision that read it narrowed the hypothesis rather than abandoning it:

> n000003 gives new evidence that detection-side adaptive-threshold changes hurt, refining b1 away from a repeat of that failure.

The branch's second execution ran under that refined directive from n000003 itself and produced n000004 at 0.1188, crossing F_{open} by $R(P) = 0.0809$ and more than tripling the incumbent. The next revision recorded the relation between the two executions explicitly:

> n000003 is a confirmed dead-end regression already superseded by its own fix n000004.

and the one after it read the result back against the hypothesis that had motivated the branch:

> b1's delayed_success on n000004 confirms detection fix is real and dominant.

The losing execution was not noise to be discarded: it located the part of the intervention that was wrong, and the same branch supplied the fix.

C.3 JOINT REVISION: EVIDENCE CROSSING SLOTS

aptos2019. n000009 was produced by delegation: the low-level search chose the parent and no belief directed it. The revision that read it used it to settle a question about a different slot, one whose directive had not yet been executed at all:

> n000009 effectively refutes b1's untested finer-grid hypothesis by already testing it (near-duplicate) with only $+0.0014$ gain; three consecutive model-tuning failures (n000006–n000008) confirm b2 must pivot fully to feature selection, not model config.

One revision settles two slots from one shared history. It refutes b1 with an execution that no belief directed, before b1 had spent any budget of its own, so the evidence lies outside b1's own record, and it confirms b2 on a course that b2's own failures had set.