REHEARSE: STEPPING BACK FROM THE CONFIDENCE CLIFF IN AUTORESEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Improving a machine-learning system and maintaining the ability to identify its next improvement are distinct challenges for autoresearch. We isolate pre-execution judgment from proposal generation and execution, and examine how it changes as successful modifications accumulate. We construct a 366-pair benchmark from 39 paper-derived tasks in public AutoSOTA logs (Li et al., 2026; Tsinghua FIB Lab, 2026). Its primary 296 same-baseline pairs each match one modification that produced an accepted improvement with one that did not. With measured outcomes hidden, an LLM judge given candidate rationales but no prior-attempt history becomes more consistent across presentation orders but less accurate as successful changes accumulate. On the full benchmark, accuracy on agreed verdicts (selective accuracy) falls from 82.8% to 56.9%, while the fraction of pairs receiving an agreed verdict (strict-consensus coverage) rises from 76% to 85%. We call this mismatch the *confidence cliff*. **Rehearse** is a lightweight skill that proposes several ideas, retrieves similar past attempts and their outcomes to compare the candidates, runs the most promising, and records its outcome for future judgments. This focused outcome memory raises late selective accuracy to 83.5%. Across 4,000 budgeted training runs over three loops, Rehearse improves the endpoint under the same training-run budget on nanochat, image classification, and time-series forecasting.

1 INTRODUCTION

Autoresearch lets an LLM agent improve a machine-learning codebase by proposing modifications, running training experiments, and keeping changes that improve the metric. Improving the codebase and reliably identifying its next improvement are distinct challenges: each accepted change creates a new baseline for subsequent decisions. An unsuccessful modification still consumes a training run before its patch is reverted.

The public AutoSOTA trajectories show why this distinction matters. The logs (Li et al., 2026; Tsinghua FIB Lab, 2026) show that the fraction of helpful modifications falls from 70% in iterations 1–2 to 43% in iterations 6–10 and 44% thereafter; among helpful modifications, the mean relative gain falls from 3.6% in iterations 1–2 to 0.3% after iteration 10 (Supplementary Figure 4, Appendix H). Useful changes become less common and yield smaller gains when they do occur. Efficient allocation of training runs therefore depends on recognizing what can still improve the current baseline.

Recent work has established that pairwise preferences can filter candidate ML solutions before execution (Zheng et al., 2026). We ask a complementary question: whether this judgment remains reliable as a loop accepts changes and moves through an increasingly history-dependent optimization trajectory. The AutoSOTA logs let us test this depth dependence directly (Li et al., 2026; Tsinghua FIB Lab, 2026). We build an outcome-labeled benchmark from 39 paper-derived AutoSOTA tasks, centered on 296 strictly same-baseline pairs, each pairing one modification that produced an accepted improvement with one that did not, with an auxiliary 70-pair worked-versus-worked ranking probe. With an outcome-blind candidate rationale but no memory, a judge comparing candidates before execution selects much better than random. The live ablation in Figure 3a also shows that explicit selection beats taking the first proposal, so the judgment is actionable.

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.

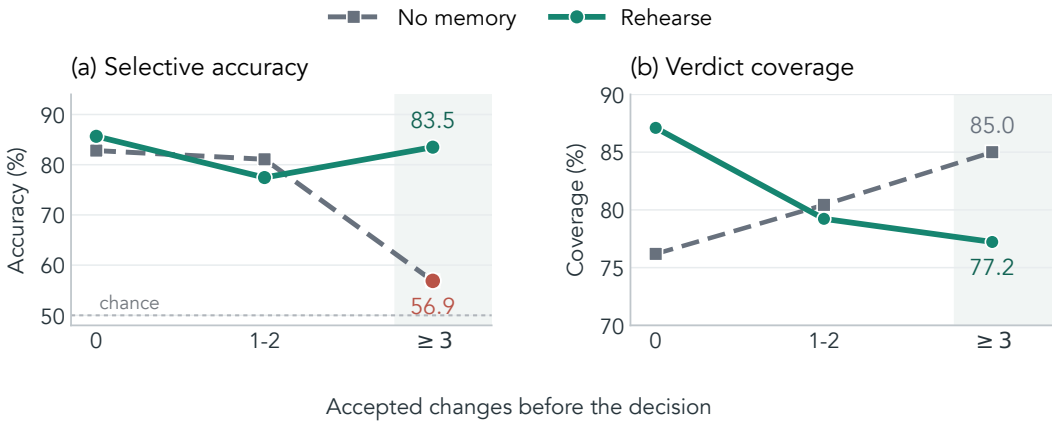

Figure 1: **Focused outcome memory mitigates the late confidence cliff.** Results on the 366-pair benchmark, grouped by the number of successful changes already accumulated (n_{baseline}). **(a)** Without memory, selective accuracy falls from 82.8% to 56.9%; Rehearse retains 83.5% late accuracy. **(b)** The no-memory judge issues more verdicts at depth even as those verdicts become less accurate. Shading marks depth ≥ 3.

As successful changes accumulate, the judge becomes more consistent across presentation orders but less accurate. Under strict consensus, a verdict is issued only when both orders agree. Without history, selective accuracy falls sharply once three or more successes have accumulated, while the fraction of pairs receiving a verdict—coverage—rises (Figure 1). We call this mismatch the *confidence cliff*. The logs and benchmark expose two linked late-stage problems: useful changes become rarer, and pre-run selection becomes less reliable on the remaining candidates. Rehearse targets the second problem.

On the same late decisions, focused outcome memory substantially improves selective accuracy over no history and a full-history dump, while an LLM summary is the closest tested alternative. Rehearse combines an explicit pre-run comparison step with a judge that sees only a focused set of relevant past outcomes. The agent proposes several modifications, compares them before execution, and runs only the most promising. Each executed modification is then recorded as what changed and whether it worked, and the next judgment sees only candidate-relevant records.

Contributions. **(1) We isolate and measure a depth-dependent failure in pre-execution judgment.** An outcome-labeled benchmark of incremental code modifications separates candidate selection from proposal generation and execution. It exposes the confidence cliff: the no-history judge becomes less accurate as accepted changes accumulate, even as strict-consensus coverage rises. **(2) We show that candidate-relevant outcomes improve late-stage judgment.** With candidate pairs and judge models held fixed, focused retrieval substantially improves cliff selective accuracy over no history and a full-history dump; the LLM-summary baseline is the closest tested alternative (80.9% versus 83.5%). **(3) We validate this intervention in live autoresearch and deliver it as a skill.** Rehearse combines proposal, pre-run selection, and focused outcome memory. Across $4{,}000$ budgeted training runs over three loops, Rehearse improves the endpoint under the same training-run budget on nanochat, image classification, and time-series forecasting. We will release the skill and benchmark.

2 Related Work

2.1 Measurable-objective autoresearch and pre-execution selection

We study autoresearch loops that optimize a *measurable* objective: a validation loss or accuracy returned by a full training run. Karpathy's nanochat loop edits `train.py`, runs one training job, reads back `val_bpb`, and keeps the edit only if the score improves (Karpathy, 2026). AIDE casts ML engineering as search over a scored solution tree, drafting, debugging, or improving code from

selected nodes (Jiang et al., 2025). The evolutionary line (FunSearch (Romera-Paredes et al., 2024), AlphaEvolve (Novikov et al., 2025), and the open CodeEvolve (Assumpção et al., 2025)) keeps archives of scored programs and uses LLM generation, mutation, or crossover to propose variants. Aster (Bicker, 2026) and Bilevel Autoresearch (Qu & Lu, 2026) study iteration efficiency, the latter on the same nanochat benchmark we use. MLR-Copilot (Li et al., 2024) plans ideas before running them but grounds candidates with external research artifacts such as prototype code, models, and datasets. ML-engineering benchmarks such as MLE-Bench (Chan et al., 2025) evaluate agents that iterate on machine-learning experiments, and Bayesian-optimization or AutoML systems, including LLM-augmented variants such as LLAMBO (Liu et al., 2024), also choose experiments before paying for evaluation. Those systems usually operate over numerical or structured configuration spaces. A direct comparison on the autoresearch testbed reports that LLM agents struggle to track optimization state across trials (Ferreira et al., 2026).

ForeAgent is the closest prior system on pre-execution selection: it compares fully instantiated ML solution code and uses confidence-gated pairwise filtering to choose which candidates to verify in an AIDE-style search (Zheng et al., 2026). Rehearse instead compares pending incremental proposals before either patch is applied, with each primary benchmark pair sharing the same accepted baseline. The resulting decision is which next modification deserves a training run, and our focus is how that decision changes with loop depth and with retrieval of the same loop's prior outcomes.

2.2 Autonomous scientific research

A second line automates broader scientific discovery and research write-up. The AI Scientist (Lu et al., 2024) and its successor v2 (Yamada et al., 2025), AI-Researcher (Tang et al., 2025), and AutoResearchClaw (Liu et al., 2026a) generate ideas, run experiments, and draft or evaluate papers. The AI Co-Scientist (Gottweis et al., 2026) instead generates, critiques, ranks, and evolves research hypotheses and proposals for scientist-guided validation. These systems primarily evaluate end-to-end research, paper, or hypothesis quality, rather than the reliability of pre-execution candidate selection under a fixed measurable training metric. Persistent-memory research systems such as EvoScientist (Lyu et al., 2026) and AutoResearchClaw (Liu et al., 2026a) distill past ideas, experiments, or mistakes to improve later research runs.

2.3 Memory and reflection in agents

Our store builds on memory, reflection, and test-time refinement work. Generative Agents store and retrieve natural-language experience for believable behavior (Park et al., 2023); MemGPT pages context in and out like an operating system (Packer et al., 2023); Voyager keeps a library of executable skills (Wang et al., 2024a); Reflexion reflects on task feedback across trials (Shinn et al., 2023); and Self-Refine performs test-time self-feedback and refinement (Madaan et al., 2023). Devil's Advocate moves reflection before an action by anticipating failures and remedies (Wang et al., 2024b).

Recent work stores typed records of agent experience as well. Trajectory-Informed Memory (Fang et al., 2026) extracts strategy, recovery, and optimization learnings from execution trajectories and retrieves them on app-and-API tasks. Concurrent work structures failed autoresearch attempts as typed records in shared memory for subsequent research agents (Wang, 2026). EvolveMem uses an autoresearch loop to adapt the retrieval architecture itself (Liu et al., 2026b). Rehearse keeps retrieval fixed and uses candidate-relevant outcomes to choose among pending modifications under a measurable ML-code objective.

2.4 Self-improving loops and self-modifying agents

Self-improving agents can fine-tune on self-generated rationales or self-judged reward data (Zelikman et al., 2022; Yuan et al., 2024), or rewrite their own source code (Zhang et al., 2025). Rehearse instead changes how the loop selects modifications to the target ML codebase, while holding the agent weights and source code fixed.

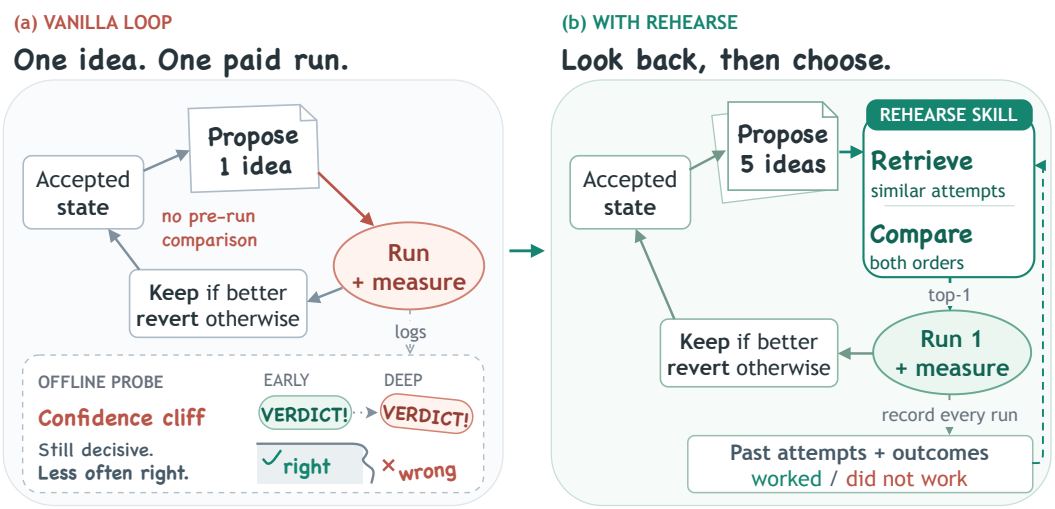

Figure 2: **The confidence cliff motivates a pre-run decision step.** A vanilla loop sends its single proposal directly to training. The dashed inset summarizes the outcome-blind offline probe applied to logged candidate pairs: deep verdicts remain decisive but are less often correct. Rehearse instead proposes five candidates, retrieves relevant prior outcomes for each, ranks all pairs in both orders, and runs only the top-ranked candidate. Each measured outcome is then recorded for future retrieval.

3 REHEARSE

Figure 2 contrasts a standard autoresearch loop with Rehearse. A vanilla loop proposes one modification, runs a full training job, and keeps the change if the metric improves; its pre-run judgment is bundled into proposal rather than exposed as a separate system action. The dashed inset summarizes the offline diagnostic built from logged candidate pairs. Rehearse adds an explicit pre-run comparison step, *predict*, supported by a memory of past attempts and outcomes. The judge ranks candidate modifications using retrieved records of similar attempts and sends only the top k to training (all live experiments here use $k = 1$). We call the resulting loop *Propose–Predict–Execute* (Supplementary Algorithm 1, Appendix A).

3.1 PREDICTING BEFORE EXECUTING

Running a candidate consumes a training run, whereas ranking uses model inference. An exhaustive strict-consensus tournament over N candidates uses $2\binom{N}{2}$ judge calls—20 when $N = 5$, as in our live loops. Rehearse ranks the candidates first and executes only the top k.

We implement the ranking as a pairwise tournament: a judge LLM is shown the shared task context—target paper/codebase summary, target metric, and baseline state—and two candidate modifications, and asked which improves the metric more. Each candidate carries a description, a *hypothesis* (the mechanistic reason the change should help), an implementation field, and a coarse type. This rationale channel lets the judge assess the proposed mechanism rather than only the edit. Removing the hypothesis and implementation produces the largest drop among prompt-component ablations (Section 4). To mitigate position bias, each unordered pair is judged in both presentation orders under *strict consensus*. An agreed verdict gives one vote to the winner; disagreement gives no vote to either candidate. Candidates are ranked by vote count and the top k run. Section 4.2 defines coverage and selective accuracy for this protocol. Pairwise auditing is important here because global judge agreement need not predict within-pool best-of-N selection quality (Landesberg, 2026).

3.2 THE FOCUSED STORE

Autoresearch systems retain evaluated history in logs, prompts, or scored search trees (Li et al., 2026; Tsinghua FIB Lab, 2026; Jiang et al., 2025; Karpathy, 2026). Rehearse stores change–outcome records but retrieves them selectively. When a new candidate is judged, retrieval is

restricted to attempts from the same task that were executed before the current step; the store returns only past attempts with cosine similarity at least 0.40 (Supplementary Appendix D). Each retrieved attempt is serialized as one line containing the change and its binary outcome, so the judge receives candidate-relevant precedents rather than a full history.

3.3 DELIVERY

Rehearse is delivered as a skill. Adopting it requires no model fine-tuning or preparatory changes to the target training code. The skill wraps the controller around an existing autoresearch loop, replacing its decision policy between proposal and execution and adding a focused outcome store backed by a few plain files. We isolate the pieces on a controlled benchmark (Section 4) and run the full skill against the vanilla loop on real tasks (Section 5).

4 SINGLE-TURN EVALUATION

We evaluate the predict step on pairs of candidate modifications with known outcomes. This isolates selection from proposal and execution, allowing us to measure how judgment changes with loop depth and with the information supplied to the judge.

4.1 A BENCHMARK OF REAL MODIFICATION OUTCOMES

We build the benchmark from public AutoSOTA logs (Li et al., 2026; Tsinghua FIB Lab, 2026), where each task starts from the repository associated with a SOTA paper and asks the optimizer to improve its measured ML metric. We treat the task/codebase, not the paper text, as the unit of analysis. Each retained candidate has a recorded outcome. The changes span optimizer, schedule, architecture, regularization, augmentation, loss, precision, and batching rather than one narrow family, and each is an incremental proposal evaluated against the task's current accepted baseline.

We use the source loop's keep/revert decision: a modification *worked* when it was accepted under the task's metric and feasibility criteria, and *did not work* otherwise. We retain a successful candidate only if its measured improvement is at least 0.5% of the baseline metric magnitude. Every type-1 candidate has a reported metric delta; none enters the primary benchmark solely as an unevaluated crash, pre-metric timeout, or invalid patch. We use two pair types. The main set is *type-1*: one modification worked and one did not work. We form these pairs only when the two candidates are evaluated from the same accepted baseline state, so both sides share the same task, codebase, metric, starting point, and remaining headroom. Attempts labeled did not work do not update that baseline, so such an attempt before the next successful change can still be paired with that later success without changing the state being judged. This strict same-baseline construction yields 296 pairs containing one worked modification and one that did not work.

We also include a smaller auxiliary *type-2* probe: both modifications worked, and the one with the larger measured improvement is labeled correct. Because successful modifications update the accepted state, a type-2 pair can compare two successful attempts from successive accepted states, where the earlier success may already be part of the later state. We deliberately restrict type-2 to cases where that earlier success is only a small incremental gain, reducing state mismatch for this auxiliary ranking probe. The main selection claim rests on type-1. For every pair, the judgment-time history is cut off before the earlier of the two candidate evaluations, and both candidates in the pair are excluded from retrieval. Together this yields 366 pairs from 39 paper-derived AutoSOTA tasks (296 type-1, 70 type-2).

Candidates are shown as pending proposals with description, hypothesis, implementation, and type. The hypothesis field is generated after log collection by a separate LLM that sees the candidate, task context, implementation, and shared baseline state, but not the measured outcome, metric delta, failure reason, or keep/revert status. The judge never sees either candidate's measured outcome.

The corpus comes from an LLM-driven optimizer, so the benchmark measures how well an agent predicts the outcome of *LLM-proposed* modifications from outcome-blind candidate rationales, approximating the decision Rehearse makes before execution. Every condition receives the same task/codebase context and candidate rationale. The no-memory condition in Table 1 is therefore

Table 1: **How history is shown matters in the late regime.** Pre-run judge performance on the full benchmark and the deep/cliff bucket ($n_{\text{baseline}} \geq 3$). Coverage is the verdict rate under strict consensus; selective accuracy is accuracy conditioned on a verdict.

Memory shown at decision	Overall selective acc. (%)	Cliff coverage (%)	Cliff selective acc. (%)
No memory (full rationale, no history)	77.6	85	56.9
Full-history dump (AIDE-inspired)	82.0	80	70.8
Self-reflection buffer (Reflexion-inspired)	81.0	90	74.1
LLM summary of history	81.9	78	80.9
Rehearse (focused retrieval, ours)	**82.3**	77	**83.5**

the baseline for a pre-run judge without prior attempts; the memory configurations add history on top of the same candidate information.

4.2 Strict-consensus judging

LLM pairwise judgments can depend on presentation order (Zheng et al., 2023; Wang et al., 2024c). We mitigate this with *strict consensus*: each pair is judged twice, in both orders, and the verdict counts only when the two orders agree; otherwise the protocol returns no verdict. Following selective-prediction terminology (Geifman & El-Yaniv, 2017), we call the fraction of pairs receiving a verdict *coverage* and the accuracy conditioned on those verdicts *selective accuracy*. We pool the LLM-judge benchmark results across three models.

4.3 Pre-execution selection across outcome types

With candidate rationales but no prior-attempt history, the judge selects the modification that worked in 79.5% of type-1 pairs receiving a strict-consensus verdict; focused outcome memory raises this to 84.2%. Selecting between two improvements is harder: on type-2 pairs, where both worked and the judge must pick the larger gain, selective accuracy rises from 67.4% without memory to 74.1% with it.

Supplementary Appendix I gives illustrative tournament calculations relating this pairwise signal to expected success@1 and Precision@3; Section 5 measures the effect of selection in live loops.

4.4 The confidence cliff

We bucket each decision by n_{baseline}, the number of successful changes already in place when it is made (Figure 1). The depth-0, depth-1–2, and depth-$\geq$ 3 buckets contain 168, 138, and 60 pairs, respectively; the corresponding counts within the primary type-1 subset are 136, 108, and 52 (Supplementary Appendix B). Without memory, selective accuracy is 82.8% at the initial baseline, then falls to 56.9% once three or more successes are in place. Strict-consensus coverage moves in the opposite direction, from 76% on shallow decisions to 85% on deep ones. This is the *confidence cliff*. Restricting the analysis to type-1 pairs leaves the pattern intact: no-memory selective accuracy moves from 85.5% to 83.3% to 56.8% across depths 0, 1–2, and $\geq$ 3, while coverage moves from 80.9% to 83.3% to 84.6%.

Table 1 compares memory strategies on these decisions. Every configuration receives the candidate *rationale* (hypothesis and implementation); only the history supplied at judgment time varies. We adapt a full-history dump inspired by AIDE's scored-tree summaries (Jiang et al., 2025) and a self-reflection buffer inspired by Reflexion (Shinn et al., 2023) to the common judge prompt, alongside an LLM summary. In the deep bucket, these reach 70.8%, 74.1%, and 80.9% selective accuracy, respectively; the self-reflection buffer has 90% coverage. Rehearse retrieves candidate-relevant prior attempts and returns them as raw change-and-outcome text, reaching 83.5% selective accuracy at 77% coverage. Overall averages hide most of the late-stage failure; the cliff bucket is where the strategies separate.

Performance does not increase monotonically with the amount of memory supplied. The full dump and Rehearse draw from the same outcome store, but Rehearse retrieves records for the pending

Table 2: **Ablating focused outcome memory in the late regime.** Single-turn cliff-bucket results ($n_{\text{baseline}} \geq 3$). Coverage is the verdict rate under strict consensus; selective accuracy is accuracy conditioned on a verdict.

Configuration	Coverage (%)	Selective acc. (%)
Rehearse (full)	77	**83.5**
Vary candidate information and retrieval		
− candidate rationale	78	70.2
− retrieved record text (→ retrieval count only)	80	70.8
− focused retrieval (→ full dump)	80	70.8
− extra neighbors (top-1 only)	77	80.5
Remove outcome information		
− outcome label	77	77.5
− did-not-work precedents (worked only)	82	78.9
− worked precedents (did-not-work only)	74	76.1
Add explanatory content		
+ recorded failure reason	77	80.6
+ curated failure reason	81	78.7
No history (full rationale)	85	56.9

candidate. The LLM summary instead compresses the history into a global state. These comparisons favor candidate-relevant records over either an unfiltered or globally summarized history. In our cost audit, the summary baseline uses about $2\times$ as many tokens per decision as Rehearse (Supplementary Appendix J).

Focused retrieval also reaches 84.8% selective accuracy in the deep type-1 bucket. On the common-verdict subset of deep pairs, accuracy is 82.5% with Rehearse versus 63.4% without history, indicating that the gain is not explained solely by differences in verdict coverage. The live-loop example in Section 5.2 illustrates why Rehearse retrieves prior outcomes in relation to each candidate.

4.5 Ablating focused outcome memory

Table 2 tests the information supplied to Rehearse through component removals, replacements, and additions. Removing the candidate rationale, retrieved text, or focused retrieval substantially lowers selective accuracy. Removing individual outcome cues lowers selective accuracy less sharply and, when did-not-work precedents are removed, raises coverage while selective accuracy falls. Adding failure explanations also hurts: recorded and curated reasons lower cliff selective accuracy from 83.5% to 80.6% and 78.7%, respectively. Rehearse therefore keeps only the change and outcome. The ablations show that both candidate-relevant record text and outcome labels improve the judge's late-stage selection. Supplementary Appendix E analyzes the components.

5 Multi-turn evaluation

We next test whether pre-run selection and focused outcome memory improve live autoresearch under a fixed training-run budget.

5.1 Setup

We run Karpathy's nanochat pretraining loop (Karpathy, 2026), scored by validation bits-per-byte (`val_bpb`, lower is better) under a fixed 30-minute-per-experiment budget, for 100 experiments. To separate the effects of selection and memory, we run four configurations that share one idea-generating model and differ only in how the step to run is chosen: *vanilla* proposes one idea and runs it; *propose-many* proposes several and runs the first; *select* proposes several and runs the predict step's pick; *Rehearse* adds focused outcome memory to select. We report five seeds each, as mean relative improvement over each run's own baseline. We then compare *vanilla* and *Rehearse* on two further loops in Section 5.3. In total, the multi-turn evaluation runs 4,000 budgeted training runs:

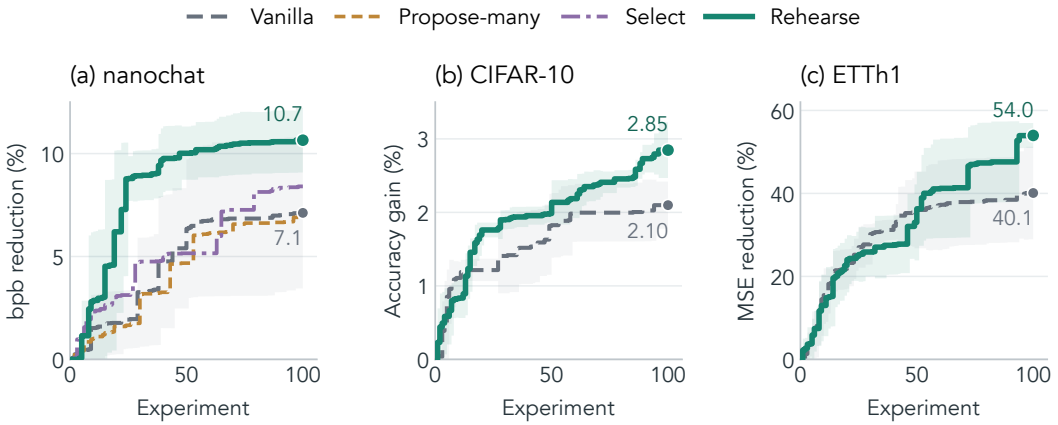

Figure 3: **Pre-run selection improves live-loop search efficiency.** Unsmoothed best-so-far improvement over baseline; lines and bands show the mean and ± one standard deviation over five seeds. **(a)** On nanochat, propose-many tracks vanilla, selection front-loads progress, and Rehearse reaches the highest endpoint. **(b–c)** On CIFAR-10 and ETTh1, Rehearse reaches a better endpoint under the same 100-experiment training-run budget.

4 nanochat configurations × 5 seeds, plus 2 configurations × 5 seeds on each of the two additional loops, all for 100 experiments.

5.2 End-to-end search efficiency

On nanochat (Figure 3a), proposing several ideas and running the first, without selecting, ends close to vanilla (6.9% vs 7.1% mean improvement). The predict step makes the larger pool useful: selecting rather than taking the first lifts the final improvement to 8.4%. The store adds further improvement, to 10.7% (+2.3 pp over select), with the store ahead on every seed.

Selection and memory affect different parts of the trajectory. Selection accelerates early search, while the store contributes additional gains later in the run. On nanochat, Rehearse has cut `val_bpb` by 8.9% at experiment 30, against vanilla's 3.3%.

A related mismatch between tournament scores and actual outcomes appears in the retained nanochat traces from three runs per method (Supplementary Appendix G). We pool executed steps within each depth group; n denotes their count. For *select* without memory, the keep rate falls from 27.3% ($n = 33$) before three accepted changes to 15.7% ($n = 254$) at greater depth. The mean normalized winner score—the fraction of other candidates beaten under strict consensus—rises from 0.70 to 0.81. With Rehearse, the keep rates in the same depth groups are 18.8% ($n = 48$) and 21.5% ($n = 247$), while the mean score remains 0.81–0.82.

Similar modifications can have different outcomes. In the live logs, lowering weight decay (0.2 to 0.15) was reverted, whereas raising it was kept across three steps (0.2 to 0.25 to 0.30 to 0.35). This example motivates interpreting prior outcomes in relation to the pending candidate: blacklisting weight-decay changes after one failed attempt would also discard the successful alternatives.

5.3 Generalization to two additional testbeds

We repeat the endpoint comparison on two additional loops: CIFAR-10 classification (Krizhevsky, 2009) and ETTh1 forecasting (Zhou et al., 2021). Each loop uses a different serving model (Supplementary Appendix B) and runs 100 experiments, comparing *vanilla* against *Rehearse* over five seeds (Figure 3b–c).

Rehearse reaches a higher final mean on both loops. On CIFAR-10 it improves accuracy by 2.85% over baseline, versus 2.10% for vanilla; on ETTh1 it reduces MSE by 54.0%, versus 40.1%. Rehearse reaches vanilla's final mean after 54 experiments on CIFAR-10 and 63 on ETTh1. Per seed, it ends ahead on 9 of 10 runs (Supplementary Table 4, Appendix G).

6 DISCUSSION

Judgment along the optimization trajectory. An autoresearch agent chooses modifications for a baseline changed by its own earlier successes. Evaluation at the initial state alone therefore leaves an important question unanswered: does the judge remain useful after the loop has already improved the system? The confidence cliff shows that agreement across presentation orders can persist even as selection deteriorates (Figure 1). Strict consensus filters out order-sensitive verdicts, but errors shared by both orders remain. In the observed trajectories, initial-state accuracy and subsequent agreement therefore give an incomplete picture of the judge's ability to identify useful changes.

Memory as evidence for the next decision. With candidate pairs and judge models held fixed, focused retrieval recovers much of the lost late-stage accuracy (Table 1). This provides an intervention at decision time: changing the context can improve selection without replacing the proposals or retraining the judge. The full-history and summary comparisons show that the representation of experience matters, not just its availability. Rehearse exposes whether similar modifications worked, letting the judge interpret those outcomes relative to each pending candidate. The relevant evidence includes both successful and unsuccessful attempts, as the outcome-removal ablations show (Table 2).

From better judgment to better search. The nanochat ablation separates having more proposals from choosing among them: running the first of several candidates ends near vanilla, whereas selection and then focused memory improve the endpoint (Figure 3a). The fixed-pair benchmark measures judgment with candidates held fixed; the live loops evaluate the resulting policy as accepted changes alter later proposals and baselines. Together, they connect improved pre-run judgment to additional search progress under the same training-run budget.

7 LIMITATIONS

Evidence and scope. The single-turn benchmark isolates pre-execution judgment, whereas live-loop outcomes also reflect proposal quality, path dependence, implementation, and metric noise. The live evidence covers three measurable-objective loops with five seeds per configuration, so broader task families and larger seed studies remain to be tested.

Candidate pools and cost. Rehearse can only select among candidates produced by its proposer. The live study uses $N = 5$ and $k = 1$; other pool sizes are untested. Reported budgets count training runs rather than model inference, and each Rehearse step uses one multi-candidate proposal call and 20 judge calls (Supplementary Appendix B).

Cliff measurement. The depth variable n_{baseline} is observational: later decisions may differ in headroom, modification family, and difficulty. Primary type-1 pairs share an accepted baseline, while memory comparisons hold candidate pairs and judge models fixed. The controlled intervention is therefore the history shown to the judge, rather than optimization depth.

8 CONCLUSION

Autoresearch efficiency depends on pre-run selection as well as idea generation. We measure this judgment across loop depth and identify a late regime where selective accuracy falls while coverage stays high. Rehearse addresses this failure with focused records of similar attempts and outcomes, which support an explicit comparison of candidates before execution. We implement this Propose–Predict–Execute loop as a skill. Across $4{,}000$ budgeted training runs on three loops, it improves the endpoint under the same training-run budget.

AI USE STATEMENT

Generative AI tools assisted with language polishing. The authors reviewed all AI-assisted edits and take responsibility for the final manuscript.

REFERENCES

Henrique Assumpção, Diego Ferreira, Leandro Campos, and Fabricio Murai. CodeEvolve: An open source evolutionary coding agent for algorithmic discovery and optimization. *arXiv preprint arXiv:2510.14150*, 2025. URL https://arxiv.org/abs/2510.14150.

Emmett Bicker. Aster: Autonomous scientific discovery over 20x faster than existing methods. *arXiv preprint arXiv:2602.07040*, 2026. URL https://arxiv.org/abs/2602.07040.

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 *The Thirteenth International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7e3767db483c942b883eb4f8cfb74e31-Abstract-Conference.html.

Gaodan Fang, Vatche Isahagian, K. R. Jayaram, Ritesh Kumar, Vinod Muthusamy, Punleuk Oum, and Gegi Thomas. Trajectory-informed memory generation for self-improving agent systems. *arXiv preprint arXiv:2603.10600*, 2026. URL https://arxiv.org/abs/2603.10600.

Fabio Ferreira, Lucca Wobbe, Arjun Krishnakumar, Frank Hutter, and Arber Zela. Can LLMs beat classical hyperparameter optimization algorithms? a study on autoresearch. *arXiv preprint arXiv:2603.24647*, 2026. URL https://arxiv.org/abs/2603.24647.

Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/4a8423d5e91fda00bb7e46540e2b0cf1-Abstract.html.

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*, 655:487–496, 2026. doi: 10.1038/s41586-026-10644-y. URL https://doi.org/10.1038/s41586-026-10644-y.

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. URL https://arxiv.org/abs/2502.13138.

Andrej Karpathy. autoresearch: A minimal self-improving LLM research loop. GitHub repository, 2026. URL https://github.com/karpathy/autoresearch/commit/228791fb499afffb54b46200aca536f79142f117. Commit 228791fb499afffb54b46200aca536f79142f117; accessed 2026-09-15.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Eddie Landesberg. When LLM judge scores look good but best-of-N decisions fail. *arXiv preprint arXiv:2603.12520*, 2026. URL https://arxiv.org/abs/2603.12520.

Ruochen Li, Teerth Patel, Qingyun Wang, and Xinya Du. MLR-Copilot: Autonomous machine learning research based on large language models agents. *arXiv preprint arXiv:2408.14033*, 2024. URL https://arxiv.org/abs/2408.14033.

Yu Li, Chenyang Shao, Xinyang Liu, Ruotong Zhao, Peijie Liu, Hongyuan Su, Zhibin Chen, Qinglong Yang, Anjie Xu, Yi Fang, Qingbin Zeng, Tianxing Li, Jingbo Xu, Fengli Xu, Yong Li, and Tie-Yan Liu. AutoSOTA: An end-to-end automated research system for state-of-the-art AI model discovery. *arXiv preprint arXiv:2604.05550*, 2026. URL https://arxiv.org/abs/2604.05550.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Meng Chen, Congyu Zhang, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, Lu Feng, Xujiang Zhao, Haifeng Chen, Jiawei Zhou, Xiao Wang, Weitong Zhang, Hongtu Zhu, Yun Li,

Jieru Mei, Hongliang Fei, Jiaheng Zhang, Linjie Li, Linjun Zhang, Yuyin Zhou, Sheng Wang, Caiming Xiong, James Zou, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. AutoResearchClaw: Self-reinforcing autonomous research with human-AI collaboration. *arXiv preprint arXiv:2605.20025*, 2026a. URL https://arxiv.org/abs/2605.20025.

Jiaqi Liu, Xinyu Ye, Peng Xia, Zeyu Zheng, Cihang Xie, Mingyu Ding, and Huaxiu Yao. EvolveMem: Self-evolving memory architecture via AutoResearch for LLM agents. *arXiv preprint arXiv:2605.13941*, 2026b. URL https://arxiv.org/abs/2605.13941.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/84b8d9fcb4e262fcd429544697e1e720-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2408.06292.

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*, 2026. URL https://arxiv.org/abs/2603.08127.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

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. URL https://arxiv.org/abs/2506.13131.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023. URL https://arxiv.org/abs/2310.08560.

Joon Sung Park, Joseph O'Brien, Carrie Jun Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pp. 1–22. ACM, 2023. doi: 10.1145/3586183.3606763. URL https://doi.org/10.1145/3586183.3606763.

Yaonan Qu and Meng Lu. Bilevel autoresearch: Meta-autoresearching itself. *arXiv preprint arXiv:2603.23420*, 2026. URL https://arxiv.org/abs/2603.23420.

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

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, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. AI-researcher: Autonomous scientific innovation. *arXiv preprint arXiv:2505.18705*, 2025. URL https://arxiv.org/abs/2505.18705.

Tsinghua FIB Lab. AutoSOTA public optimization logs and code. GitHub repository, 2026. URL https://github.com/tsinghua-fib-lab/AutoSOTA/commit/419fe0139e0bcc0bf5988da8cf7bf14d7cf71eef. Commit 419fe0139e0bcc0bf5988da8cf7bf14d7cf71eef; accessed 2026-09-15.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2024a. URL https://openreview.net/forum?id=ehfRiF0R3a.

Hanchun Wang. Negative knowledge as failure-aware shared memory for AutoResearch. *arXiv preprint arXiv:2606.21024*, 2026. URL https://arxiv.org/abs/2606.21024.

Haoyu Wang, Tao Li, Zhiwei Deng, Dan Roth, and Yang Li. Devil's advocate: Anticipatory reflection for LLM agents. *arXiv preprint arXiv:2405.16334*, 2024b. URL https://arxiv.org/abs/2405.16334.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9440–9450, Bangkok, Thailand, August 2024c. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.511. URL https://aclanthology.org/2024.acl-long.511/.

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. URL https://arxiv.org/abs/2504.08066.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E. Weston. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 57905–57923. PMLR, 2024. URL https://proceedings.mlr.press/v235/yuan24d.html.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/639a9a172c044fbb64175b5fad42e9a5-Abstract-Conference.html.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel Machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*, 2025. URL https://arxiv.org/abs/2505.22954.

Jingsheng Zheng, Jintian Zhang, Yujie Luo, Yuren Mao, Yunjun Gao, Lun Du, Huajun Chen, and Ningyu Zhang. Can we predict before executing machine learning agents? In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3941–3974, San Diego, California, United States, July 2026. Association for Computational Linguistics. doi: 10.18653/v1/2026.acl-long.182. URL https://aclanthology.org/2026.acl-long.182/.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(12):11106–11115, 2021. doi: 10.1609/aaai.v35i12.17325. URL https://ojs.aaai.org/index.php/AAAI/article/view/17325.