OPENMLE: TRAINING LLMS FOR EVOLUTIONARY SEARCH IN MACHINE LEARNING ENGINEERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models have progressed from conversational assistants to reasoners and autonomous agents, and are now entering a stage in which they contribute to discovery itself, including the development of AI: *AI that trains AI*. Frontier labs have begun pursuing automated AI research, yet the open community lacks the training data and testbeds needed to study it: existing machine learning engineering (MLE) resources contain at most a few hundred tasks and are built for evaluation rather than training. We present OPENMLE, an open full-stack framework for AI-train-AI in executable MLE. First, OPENMLE-GYM provides, to our knowledge, the first large-scale AI-train-AI testbed: 5,758 quality-gated executable tasks spanning broad machine learning problems, with isolated execution, task-specific evaluators, and verified evolutionary trajectories. Second, OPENMLE-TRAIN post-trains a single model for the *discovery loop*, where an agent repeatedly drafts, improves, debugs, and recombines solutions under delayed, noisy feedback: execution-verified SFT, then RL on per-task calibrated execution rewards, teaches it to serve as the shared program-revision operators. Third, OPENMLE-EVO deploys the trained model as the variation engine of long-horizon evolutionary search. On MLE-Bench Lite, post-training Qwen3.6-35B-A3B into OPENMLE-35B raises Medal Average from 39.39% to 60.61% with the harness fixed. With the model fixed, OPENMLE-EVO raises it from 53.03% under original AIRA-Evo to 60.61%. Adding benchmark-disjoint experience priors and asynchronous parallel search to OPENMLE-EVO, at the same total sandbox budget, brings the system to 71.21% with 3B active parameters, on par with GPT-5.5 with Codex (68.18%) under the same budget. Results on ten held-out NatureBench tasks show transfer to scientific reproduction tasks.

1 INTRODUCTION

Large language models have progressed from *chat* to *reasoning* (OpenAI, 2024; Guo et al., 2025) and autonomous *agents* (Jimenez et al., 2024; Yang et al., 2024), motivating *innovation* as a next stage, where models contribute to discovery beyond executing specified tasks (Metz, 2024; Morris et al., 2024). AI research offers a concrete setting: systems already write code, run experiments, and search algorithms (Romera-Paredes et al., 2024; Lu et al., 2024; Novikov et al., 2025) to help build subsequent models, enabling *AI-train-AI*.

OpenAI and Anthropic pursue and assess automated AI research (Chan et al., 2024; Starace et al., 2025; OpenAI, 2025; Anthropic, 2024), but open research needs three resources: *training-scale tasks*, beyond the tens to hundreds in evaluation-oriented MLE environments (Huang et al., 2023; Chan et al., 2024; Qiang et al., 2025a; Nathani et al., 2025); *open training recipes* that target iterative improvement rather than single-shot outcomes (Liu et al., 2025b; Li et al., 2025b; Yang et al., 2025a; Cai et al., 2026); and *a reproducible testbed* connecting environment construction, training, and search. Together, these resources would let the broader community study how executable experience improves the models driving discovery.

Machine learning engineering is a direct and measurable instantiation of AI-train-AI: an agent must build a machine learning solution for a real task and iteratively improve it from execution feedback (Huang et al., 2023; Chan et al., 2024). What distinguishes it from standard agentic tasks is that it is a *discovery loop*. There is no reference answer; each iteration costs minutes to hours of compute;

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.

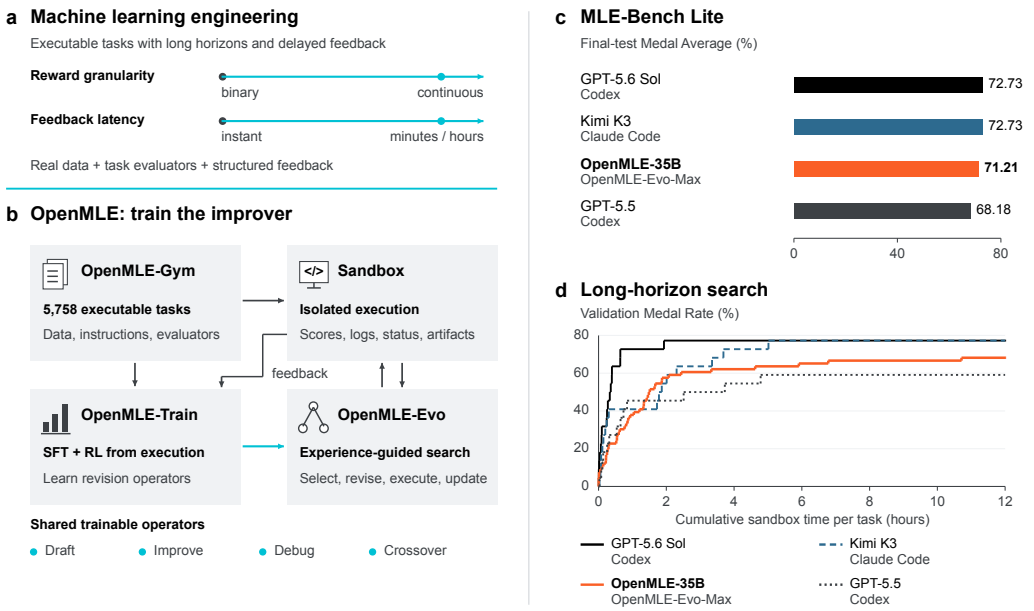

Figure 1: **OpenMLE overview and performance.** **(a)** MLE tasks feature delayed execution feedback. **(b)** OpenMLE connects 5,758 executable tasks, post-training, and evolutionary search through shared operators. **(c)** Final-test Medal Average on 22-task MLE-Bench Lite for OpenMLE-35B with OpenMLE-Evo-Max and the three strongest external systems. **(d)** Best-so-far validation Medal Rate over 12 hours of sandbox execution, using three recorded outcomes per task for ours and one per reference.

feedback is delayed, noisy, and heterogeneous across tasks; and the value of a step lies in whether it improves on the current best solution, not in whether it is "correct." Strong inference-time harnesses exploit this structure through evolutionary or tree search (Jiang et al., 2025; Fang et al., 2025; Nam et al., 2025; Du et al., 2025; Liu et al., 2025a; Toledo et al., 2025; Hambardzumyan et al., 2026), but the model that proposes each revision is rarely trained for that role, and open data and recipes for training it are scarce.

We introduce OpenMLE, an open full-stack MLE framework. OpenMLE-Gym supplies 5,758 quality-gated tasks spanning broad machine learning problems, with isolated execution, structured feedback, task-specific evaluators, and verified evolutionary trajectories. OpenMLE-Train is a post-training recipe that trains shared Draft, Improve, Debug, and Crossover operators with execution-verified SFT followed by RL on task-calibrated execution rewards. OpenMLE-Evo composes these operators through structured experience, non-greedy selection over quality, progress, and novelty, and operator-conditioned memory. The shared interface aligns supervision with deployment: verified evolutionary transitions teach the same program transformations that search later composes over long horizons. Training revisions on executed search experience makes the model the variation engine of a meta-evolutionary loop (Figure 1).

Post-training Qwen3.6-35B-A3B with OpenMLE yields OpenMLE-35B, and we report its MLE-Bench Lite results in three steps. With the OpenMLE-Evo harness fixed, post-training raises Medal Average from 39.39% to 60.61%, placing the model above six other models run under the same harness; OpenMLE-30B reproduces the gain on a second backbone (34.85% to 53.03%). With the model fixed, OpenMLE-Evo raises Medal Average from 53.03% under original AIRA-Evo to 60.61%, and it also improves over Claude Code or Codex on each of the four other models we test. Adding benchmark-disjoint experience priors and asynchronous parallel search to OpenMLE-Evo, at the same total sandbox budget, brings OpenMLE-35B to 71.21% while activating only 3B parameters per token, on par with GPT-5.5 with Codex at 68.18% under the same per-task budget. Ten held-out NatureBench tasks show generalization beyond competition-style MLE.

Our contributions are:

1. **A training-scale testbed.** OPENMLE-GYM exceeds prior public resources compared in Figure 2 by over 10×.
2. **A post-training recipe for discovery loops.** OPENMLE-TRAIN turns verified executions into SFT and RL signal for reusable operators.
3. **A closed training–search loop.** OPENMLE-EVO deploys those operators in experience-guided long-horizon search.
4. **An open model and reproducible artifacts.** Controlled ablations, backbone replication, and scientific-task transfer evaluate the resulting models. Method code accompanies submission; the complete corpus, the SFT trajectories, and both checkpoints are available anonymously (Appendix G.2).

2 OPENMLE-GYM: SCALABLE VERIFIABLE ENVIRONMENTS

Agents that build AI systems through executable experimentation need large, diverse, high-quality task packages, and existing resources are limited in openness, executable diversity, or coverage. A collection of tasks is also not enough by itself: post-training and search both require that a generated program be executed under controlled resources and return feedback reliable enough to act on. Each OPENMLE-GYM task is therefore an environment instance: the state is the specification, public data, hidden evaluator, resource budget, and workspace; the action is an agent-submitted MLE program; the transition is sandbox execution; the observation is a structured execution record; and the reward is the verifiable task-specific score. Appendix A formalizes the evolutionary search that runs on these environments and the post-training objective built on it.

Scalable construction and quality control. Three source-specific paths occupy complementary positions on the quality–scale trade-off: *curated anchors* (156 tasks), manually selected from papers and benchmarks; *Kaggle datasets* (3,362), broadening coverage through an extended MLE-Smith pipeline (Qiang et al., 2025b) with package-level quality control; and *Kaggle competitions* (2,240) in between, where human-authored specifications, metrics, and submission protocols give stronger grounding while the Meta Kaggle catalog supports scale. All three map to one executable package: agent-visible descriptions, training data, test inputs, and a sample submission under a public path, hidden answers under a private one, and a task-specific `metric.py`. Competition packages are built automatically from file schemas, sampled rows, and catalog records, validated by executing their preparation and scoring code, and then passed through an LLM-based semantic quality gate; only tasks passing both gates are retained. MLE-Bench-overlapping competitions are excluded before training. Figure 2 compares task scale with existing works and analyzes the distribution of our curated tasks. Appendix C.1 details our pipeline and the gate, and shows the source hierarchy, curation funnel, and package format (Figure 9).

Sandbox execution at scale. A centralized scheduler records each job, tracks worker availability, and dispatches to CPU/GPU Docker workers by resource requirement. Each worker runs the candidate in a fresh workspace on the public inputs, scores its submission separately, and writes outputs and artifacts to shared storage. The backend returns six feedback modes — successful completion, runtime error, missing code, missing submission, scoring failure, and timeout — each preserving status, score when available, logs, error type, runtime metadata, and artifacts, so the agent can distinguish invalid execution from weak task performance. Appendices C.1 and C.2 give construction and architecture details.

3 OPENMLE-TRAIN: AN EXECUTION-GROUNDED POST-TRAINING RECIPE

Evolutionary AutoResearch is judged by the best executable program found within a finite budget. A controller may invoke operators hundreds or thousands of times, so the model must learn more than one-shot generation: it must repeatedly repair, refine, and recombine programs as the search unfolds. Post-training should therefore expand the set of strong programs reachable under repeated sampling while improving the individual transformations the harness composes. SFT and RL play complementary roles: execution-grounded SFT distils verified complete solutions and successful

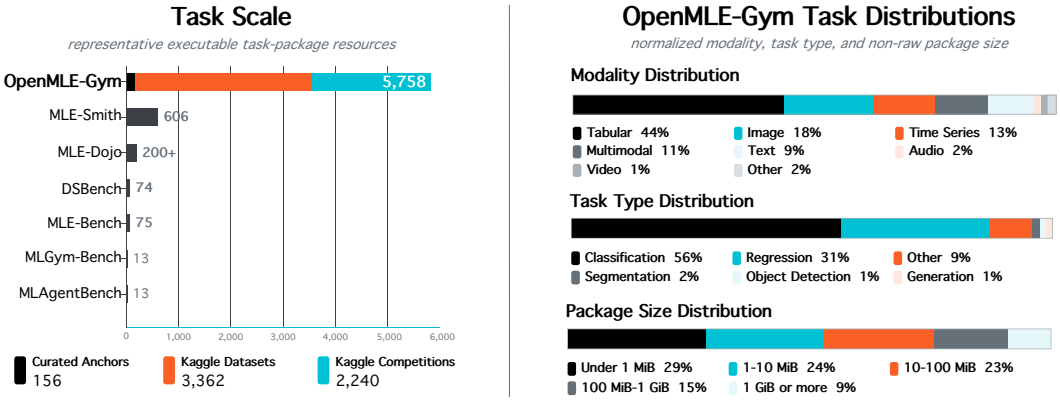

Figure 2: Task scale and distribution of OPENMLE-GYM. **Left:** comparison with executable task-package resources and source breakdown of the 5,758 tasks. **Right:** task distribution analyzed across modality, task type and package size.

local revisions from teacher models, raising the density of strong executable MLE solutions the model can produce before online learning, and RL then moves probability toward the better candidates among those it can already reach (Yue et al., 2025).

Trainable atomic operators. We aim to train the model to apply specified program-transformation operators across search controllers, rather than learn a controller-specific search policy. Accordingly, OPENMLE uses single model calls under four operators, rather than full search trajectories, as its training unit. Specifically, DRAFT generates a program from the task specification; IMPROVE revises a valid parent using its feedback; DEBUG repairs a failed parent using its error output; and CROSSOVER recombines two parents using evidence about which mechanisms to retain or avoid. Each call produces a complete candidate program, which is executed once in the sandbox. Table 6 in the appendix specifies the inputs and intended transitions for all four operators.

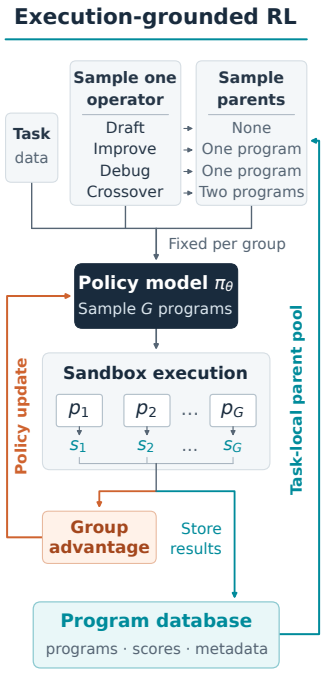

Figure 3: Operator sampling and the RL training process.

Execution-grounded supervised fine-tuning. To build execution-grounded SFT data efficiently, we execute sampled programs, retain examples by validity and task score, and stop each task when its accepted-example quota or execution budget is reached, directing verification compute to tasks with sparse successes. The *parallel path* independently samples and executes complete DRAFT solutions (17,245 full-response examples), whereas the *evolutionary path* conducts multi-step searches with all four operators (DRAFT, IMPROVE, DEBUG, and CROSSOVER) and retains useful local steps from high-quality trajectories (9,014 trajectory-step examples). Figure 12 illustrates both paths. We fine-tune the model on the resulting 26,259-example corpus; corpus statistics are in Appendix D.1, and SFT training settings in Appendix D.3.

Execution-grounded reinforcement learning. After SFT, we train the operators using a GSPO-based objective (Zheng et al., 2025), with task-adaptive reward processing and entropic advantages. As shown in Figure 3, for each rollout group we sample an operator for a given task and select its required parent context from the program database. With the task, operator, parent context, and prompt fixed, we sample G responses, each containing reasoning and a complete candidate program, and execute the programs independently.

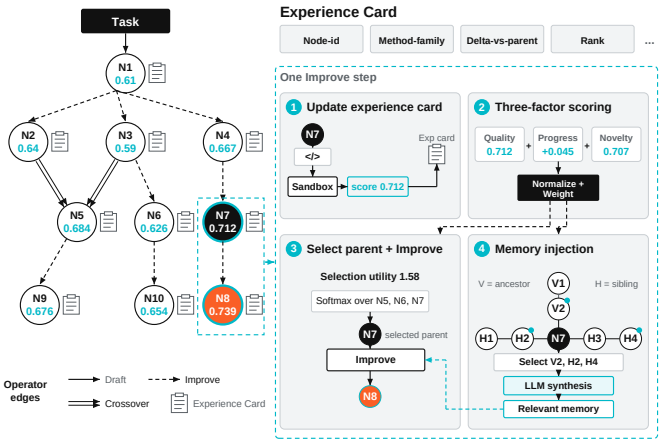

Figure 4: The OPENMLE-EVO search harness. **Left:** the search tree expands candidates through drafting, improvement, and crossover, pairing each evaluated node with a structured experience card. **Right:** card metadata updates the global search state, computes parent-selection weights from quality, progress, and novelty, selects the next parent, and retrieves relevant memories from key ancestors and siblings for the following operation.

The challenge is to turn their execution scores into a useful within-group learning signal: some programs yield no valid score, successful programs are evaluated using task-specific metrics with different ranges, and fixed leaderboard or theoretical bounds may span a much wider range than the scores attained by the current policy. We therefore construct task-adaptive bounds from the scores of successful historical programs and the current group, obtaining processed rewards $r_{\mathrm{proc},i}$ that preserve meaningful differences among candidates. For IMPROVE, DEBUG, and CROSSOVER, the reward r_i also credits verified improvement over the parent (Appendix D.4). Advantages use the entropic weighting of Yuksekgonul et al. (2026), favouring stronger candidates:

$$A_i^{\mathrm{ent}} = \frac{\exp(\beta r_i)}{\frac{1}{G-1}\sum_{j\neq i}\exp(\beta r_j)} - 1, \tag{1}$$

where β is chosen under a fixed entropy budget. The policy update applies each candidate's advantage only to tokens generated by its own call; no credit is propagated along the parent path. Sandbox runs can last minutes to hours and create long-tail delays, so completed rollout groups are consumed asynchronously from a trainer queue. Appendix D.4 gives the parent-selection rule, hyperparameters, and reward formulas; validation reward and medal count rise over roughly 200 RL steps (Figure 15).

4 OPENMLE-EVO: EXPERIENCE-DRIVEN LONG-HORIZON SEARCH

AI systems that improve AI artifacts must use execution outcomes to decide what to try next, not merely sample more candidates; over long horizons this needs a persistent queryable record of prior attempts, compute allocation across high-quality, improving, and novel branches, and bounded context tailored to the transformation applied. OPENMLE-EVO adopts an AIRA-Evo-style population loop (Toledo et al., 2025; Hambardzumyan et al., 2026) to compose OPENMLE's trained operators, but redesigns how that loop uses execution evidence. Where original AIRA-Evo keeps largely free-form memory, OPENMLE-EVO records every evaluated node as structured experience, and two mechanisms read that record: experience-guided parent selection replaces selection by scalar fitness, and on-demand, operator-conditioned memory replaces eager summarization (Figure 4).

Structured experience. After evaluating a candidate, the harness creates a node-level *experience card*. Its metadata is extracted deterministically from the search state and execution result, recording provenance, performance, outcome, and resource use. The cards are aggregated into a task-global experience board that tracks explored method families, the best candidate in each family, underexplored directions, repeated failures, score trends, and the parent graph. The record serves both mechanisms below: it supplies the quality, progress, and novelty signals for parent selection, and the lineage, neighbourhood, failure, and resource evidence from which each operator's context is built.

Experience-guided parent selection. Original AIRA-Evo derives sampling probabilities primarily from normalized fitness, so expansion concentrates on already strong nodes. OPENMLE-EVO instead samples parents from a softmax over three card-derived factors — normalized validation score,

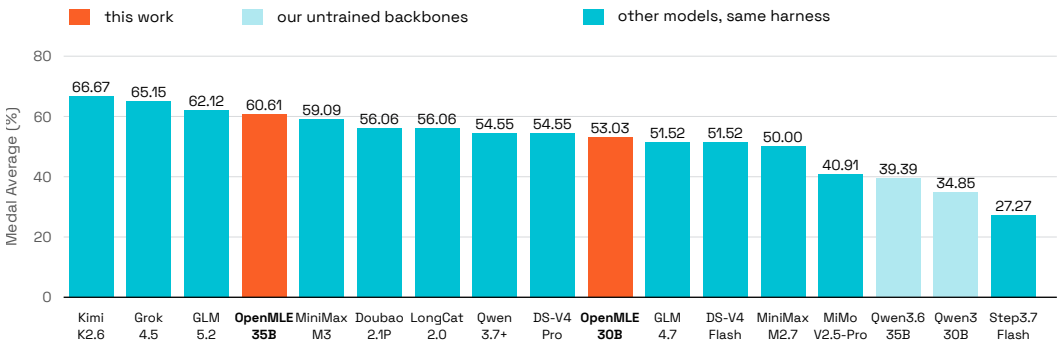

Figure 5: Model performance on MLE-Bench Lite under the common OPENMLE-EVO harness and protocol of Section 5, so differences reflect the model alone. Orange: OPENMLE-35B and OPENMLE-30B; pale: their untrained backbones. OPENMLE-EVO-MAX is excluded.

positive improvement over the strongest parent, and method-family novelty — so that budget also reaches candidates that made real progress or opened an underexplored family; Appendix E.3 gives the utility and the sampling rule.

On-demand memory and operator-conditioned context. Original AIRA-Evo summarizes every evaluated node eagerly, before the operator that will consume the summary is known. OPENMLE-EVO defers synthesis until an IMPROVE, CROSSOVER, or DEBUG call has selected its nodes, then summarizes only those parents and their retrieved ancestors, siblings, or error-matched attempts, and caches the result; each operator therefore receives a small context specific to its role; Appendices E.2 and E.3 give the retrieved state per operator, the schemas, and the prompts.

5 EXPERIMENTS

We evaluate on the official 22-task MLE-Bench Lite split released with MLE-Bench (Chan et al., 2024). Unless noted, each configuration uses three independent runs under a 12-hour per-task budget on a single RTX 4090 capped at 12 GB VRAM. *Medal Average* is the mean fraction of tasks receiving any Kaggle medal and *Human Rank* the mean fraction of human leaderboard participants surpassed, with invalid submissions scored zero. Appendices F.1 and F.2 give the full protocol and the per-configuration statistics.

5.1 POST-TRAINING WITH OPENMLE-GYM AND OPENMLE-TRAIN

Under the identical OPENMLE-EVO harness, post-training lifts OPENMLE-35B from its Qwen3.6-35B-A3B base at 39.39% Medal Average and 0.5828 Human Rank to 60.61% and 0.7647, and reproduces on a second backbone and scale, where OPENMLE-30B improves over Qwen3-30B-A3B-Thinking-2507 from 34.85% to 53.03% (Figure 5). Under this common harness the trained 35B model also places above MiniMax M3, Doubao Seed 2.1 Pro, LongCat 2.0, Qwen3.7 Plus, and both DeepSeek-V4 variants, within 6.06 points of the best model we evaluated.

Both post-training stages contribute. We evaluated the checkpoints before and after RL to separate the contributions of SFT and RL(Table 1). The results show that SFT and RL each contribute to better MLE capabilities for two backbones. In comparison, Qwen3.6-35B-A3B shows greater potential for RL: even after SFT, RL still yields noticeable score improvements.

Scaling executable tasks produces a better MLE agent. Training the same SFT stage for the same number of optimizer steps, on $1/8$, $1/4$, $1/2$, and all of the corpus (Figure 6) shows the supply of executable tasks itself mattering. Across medal average, human rank, and valid rate (Figures 6 and 14), every fraction improves on the untrained backbone, and the full corpus gives the highest Medal Average on both backbones.

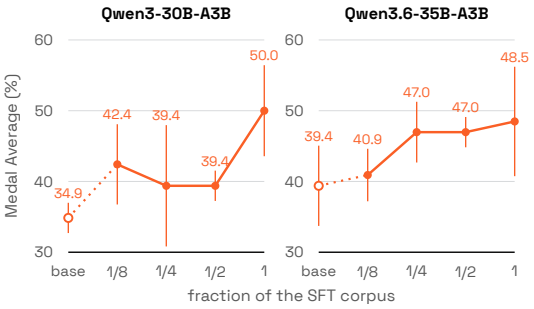

Figure 6: SFT data scaling: SFT-only checkpoints trained for the same number of optimizer steps on $1/8$ to all of the corpus. Open markers are the untrained backbones; bars are ± 1 s.d. over three runs. Appendix D.2 gives Valid Rate and Human Rank.

	Qwen3.6-35B		Qwen3-30B	
Stage	MA	HR	MA	HR
Base	39.39	0.5828	34.85	0.5573
+ SFT	48.48	0.6614	50.00	0.6375
+ RL	**60.61**	**0.7647**	**53.03**	**0.7055**

Table 1: Post-training ladder with backbone, harness, tasks, seeds, and budget fixed. MA: Medal Average (%); HR: Human Rank.

5.2 OpenMLE-Evo with OpenMLE-Evo

With the model fixed, this section isolates the contribution of search; Appendix F.4 shows two search trajectories.

OpenMLE-Evo outperforms previous evolutionary search baseline and general agent harness. Figure 7 swaps the controller under five fixed models. Against original AIRA-Evo, the matched OpenMLE-35B comparison raises Medal Average from 53.03% to 60.61%. Against general-purpose coding-agent scaffolds, OpenMLE-Evo gains over Claude Code and Codex. Medal Average rises in every pair shown.

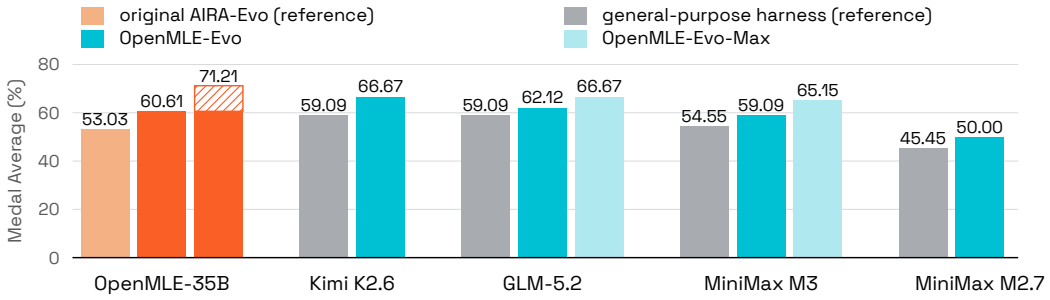

Figure 7: Harness comparison with the model held fixed: same checkpoint, sandbox protocol, and a per-task budget of 12 RTX 4090-hours. References: original AIRA-Evo for OpenMLE-35B, Codex for MiniMax M3, Claude Code otherwise. Hatched cap: OpenMLE-Evo-Max increment over OpenMLE-Evo. Medal Average (%).

Which parts of OpenMLE-Evo carry that gain. Both mechanisms of Section 4 read the same structured experience record, so the ablation keeps that record and enables one mechanism at a time (Table 2). With experience-guided parent selection, Medal Average rises from 53.03% to 56.06%; with operator-conditioned memory, to 57.58%; with both, to 60.61%. Each mechanism helps alone, but only the two together reach the full OpenMLE-Evo result.

Strengthening OpenMLE-Evo further. On top of OpenMLE-Evo, we design OpenMLE-Evo-Max which adds benchmark-disjoint experience priors distilled from public competition artifacts, and asynchronous parallel search at an unchanged total sandbox budget. The priors alone reach a medal average of 62.12%; adding parallel search reaches 71.21% with Valid Rate at 22.00/22.

Table 2: Search ablations with the model held fixed. **Top:** OPENMLE-EVO's own changes to original AIRA-Evo on OPENMLE-35B; both partial rows keep the structured experience record and enable one of the two mechanisms that read it. **Middle:** what the two OPENMLE-EVO-MAX additions contribute on top of OPENMLE-EVO.

Configuration	Valid Rate	Medal Avg. ↑	Medals	Human Rank ↑
What inside OPENMLE-EVO *carries the gain over original AIRA-Evo*				
OPENMLE-35B · original AIRA-Evo	—	53.03%	35/66	—
+ experience + guided selection	21.33 ± 0.47	$56.06\% \pm 7.73\%$	37/66	0.6986 ± 0.0370
+ experience + on-demand memory	22.00 ± 0.00	$57.58\% \pm 7.73\%$	38/66	0.7168 ± 0.0263
+ experience + both (OPENMLE-EVO)	21.67 ± 0.47	$\mathbf{60.61\% \pm 7.73\%}$	**40/66**	$\mathbf{0.7647 \pm 0.0376}$
Strengthening the search beyond OPENMLE-EVO				
OPENMLE-35B · OPENMLE-EVO	21.67 ± 0.47	$60.61\% \pm 7.73\%$	40/66	0.7647 ± 0.0376
+ experience priors	21.67 ± 0.47	$62.12\% \pm 2.14\%$	41/66	0.7283 ± 0.0103
+ priors + asynchronous search	22.00 ± 0.00	$\mathbf{71.21\% \pm 8.57\%}$	**47/66**	$\mathbf{0.8126 \pm 0.0388}$

That places the system in the same range as the frontier model–harness systems we ran under the same sandbox protocol: GPT-5.5 with Codex reaches 68.18%, and GPT-5.6 Sol and Kimi K3 72.73%. Figure 19 in Appendix F.4 tracks validation Medal Rate over the 12-hour budget together with the final test score for OPENMLE-EVO-MAX and the frontier systems.

The gain is bought with fewer tokens, not more nodes. On 66 matched task–runs with the same checkpoint, seed, and budget, OPENMLE-EVO evaluates 12.4% fewer nodes than original AIRA-Evo but spends far fewer tokens on each, and the share of IMPROVE calls that set a new best roughly doubles. Operator-conditioned memory keeps the serialized context short, most at the 99th percentile, where eager whole-history summaries grow with the search (Figure 18 and Table 14 in Appendix F.3).

5.3 GENERALIZATION ACROSS MODALITIES AND TASKS

The gain is not tied to one input modality. Partitioning the 22 tasks into image, text, tabular/structured, audio, and multimodal groups, OPENMLE-35B raises mean Human Rank in all five relative to its base under the same harness and never lowers group-level Medal Rate (Figure 8); the 14 additional medals are spread across every group ($+2/+4/+1/+4/+3$). Audio and multimodal hold two tasks each, so the partition supports the direction, not the per-group magnitude.

Transfer to scientific reproduction. NatureBench (Wang et al., 2026a) evaluates whether coding agents can reproduce or improve upon published scientific results. On a ten-task subset that we selected to limit compute cost, changing one component per controlled row, post-training and evolutionary search raise S and M, suggesting transfer beyond competition-style MLE (Table 3). OpenMLE-35B paired with OpenMLE-Evo matches MiniMax M3 and outperforms DeepSeek-V4-Pro, Claude Opus 4.6, and MiniMax-M2.7.

6 RELATED WORK

AutoResearch systems, benchmarks, and executable environments. AutoResearch systems increasingly automate hypothesis formation, experimentation, and artifact production (Lu et al., 2024; Yamada et al., 2025; Tang et al., 2026), and evaluations isolate complementary slices of that loop: open-ended AI R&D and autonomous post-training (Wijk et al., 2025; Rank et al., 2026; Kirgis et al., 2026), research-oriented problem solving (Nathani et al., 2025; Lupidi et al., 2026), reproducing published results (Starace et al., 2025; Wang et al., 2026a), budgeted policy evolution (Wang et al., 2026b), and Kaggle-style competitions (Chan et al., 2024). Unlike AutoML over predefined pipeline spaces (Thornton et al., 2013; Feurer et al., 2015; Erickson et al., 2020), LLM-based MLE agents experiment through code (Huang et al., 2023; Jing et al., 2024; Qiang et al., 2025a;b). OPENMLE-GYM targets what executable tasks leave open: turning experience into reusable model capability.

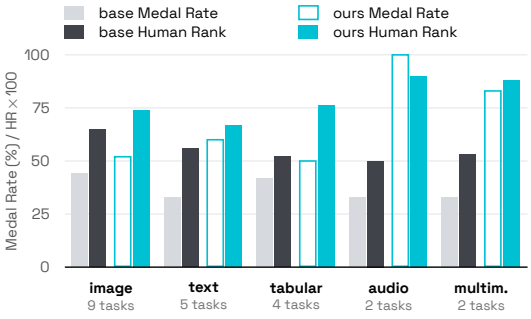

Figure 8: MLE-Bench Lite by input modality, OPENMLE-35B versus its Qwen3.6-35B-A3B base under OPENMLE-EVO. Human Rank is scaled by 100; invalid submissions count as zero.

Model · harness	S↑ /10	M↑ /10
Controlled: one component changed per row		
OPENMLE-35B · OPENMLE-EVO	**3**	**7**
Qwen3.6-35B · OPENMLE-EVO	2	5
Qwen3.6-35B · AIRA-Evo	1	2
Reference agents, reported by NatureBench		
Claude Opus 4.7 · CC	7	10
GLM-5.2 · CC	7	10
GPT-5.5 · Codex	4	10
Kimi K2.6 · CC	3	9
MiniMax M3 · CC	3	7
DeepSeek-V4-Pro · CC	2	6
Claude Opus 4.6 · CC	2	5
MiniMax-M2.7 · CC	0	3

Table 3: NatureBench, ten-task subset: number of tasks on which each agent surpasses (S, $g > 0.1$) or matches (M, $g \geq 0$) the published result, where g is the direction-normalized relative gap to that result (Appendix F.5). CC: Claude Code.

Inference-time scaffolds and evolutionary search. Inference-time scaffolds amplify frozen models through multi-agent decomposition (Trirat et al., 2024; Fang et al., 2025), structured search (Jiang et al., 2025; Chi et al., 2024), targeted refinement or domain knowledge (Liu et al., 2025a; Nam et al., 2025; Du et al., 2025), and evidence accumulated across attempts (Fu et al., 2026; Li et al., 2026). Analyses identify search policy, operator quality, throughput, and ideation diversity as key factors (Toledo et al., 2025; Hambardzumyan et al., 2026), and budget-matched comparisons find no universally best harness (Gupta et al., 2026). Program-evolution systems use language models as mutation operators and executable evaluators as selection (Novikov et al., 2025; Lange et al., 2025), while test-time learning updates the model from execution feedback (Wang et al., 2025; Yuksekgonul et al., 2026). OPENMLE-EVO instead composes operators trained for their roles in search.

Learning MLE agents, and trainable improvers. RLVR drove strong reasoning in mathematics and coding (Guo et al., 2025; Zeng et al., 2025) and now trains long-horizon research and MLE agents (Falck et al., 2026; Yang et al., 2026; Qiang et al., 2026). MLE-specific efforts learn from executable task rewards (Liu et al., 2025b; Li et al., 2025b; Yang et al., 2025a; Cai et al., 2026) but release only part of the end-to-end stack (Appendix G.2). AI-for-AI further optimizes agents, harnesses, and weights (Hu et al., 2024; Lee et al., 2026a; Srikanth et al., 2026; Kim et al., 2026). Closest to our setting, training the proposer inside evolutionary search was shown for program synthesis (Pourcel et al., 2025) and, concurrently, for optimization tasks (Lee et al., 2026b) and coding (MetaEvolve; Wu et al., 2026). MetaEvolve, the nearest, trains a single refinement role by RL on synthesized evolution histories of competitive-programming problems scored by test cases, and runs it in the standard OpenEvolve loop. OPENMLE instead contributes environments, training, and search as one stack for MLE, where scoring a candidate means training a model and feedback is slow and noisy: OPENMLE-GYM supplies 5,758 executable tasks and verified trajectories, OPENMLE-TRAIN trains four operators on them, and OPENMLE-EVO composes them in long-horizon search.

ETHICS STATEMENT

OPENMLE reduces the cost of generating, debugging, and recombining machine learning code, which could equally accelerate unsafe development, amplify a misspecified metric, consume unbounded compute, or produce artifacts of unclear data and licence provenance. Every candidate therefore runs in an isolated sandbox under explicit time and resource limits, is scored by a task-specific evaluator behind a hidden holdout, and leaves complete operation and lineage logs; the scheduler, time limits, and stop policy are set outside the generated program, and scoring runs in a separate process after the

program exits. Scaling workers or search depth should further require stable evaluator performance on adversarial holdouts, complete artifact provenance, and successful replay of sampled improvements, with staged release retaining rollback, evaluator versioning, and the ability to invalidate descendants of a corrupted measurement. The system is not a permissionless self-modifier. The work involves no human subjects and no personally identifying data. Training tasks derive from public competition and dataset sources; we redistribute task packaging and evaluation code rather than the underlying competition data, and each task package records its source and licence.

REPRODUCIBILITY STATEMENT

This statement points to where the material needed to reproduce our results can be found. The unit of work is specified rather than described: Section 3 introduces the atomic operation, and Appendix E.1 defines it and gives the IMPROVE prompt template and two grounded instances; Appendix E.2 lists the fields of the record written after each call. The evaluation protocol — task split, number of runs, per-task budget, and hardware — is stated in Section 5 and expanded in Appendix F.1, with per-configuration means and standard deviations in Appendix F.2. Task construction and acceptance gates are in Appendix C.1; the SFT and RL corpora, training hyperparameters, reward processing, and the entropic advantage estimator are in Appendices D.1, D.3, and D.4; the search-state schema is in Appendix E.2, and parent selection, memory retrieval, and the controller configuration are in Appendix E.3; and the fields that a replay of a recorded result requires are listed in Appendix C.3. An anonymised archive containing the code for our method is submitted as supplementary material, and the complete OPENMLE-GYM corpus, the SFT trajectories, and both post-trained checkpoints are available at an anonymous repository (`https://huggingface.co/datasets/nono314/openmle`), so that the pipeline can be inspected and exercised end to end during review; task data is redistributed under the terms of its original sources. Table 18 records the status of each artifact class at submission, alongside the eighteen prior systems it audits.

AI USE STATEMENT

Generative models appear in this work in two distinct roles, and we separate them here. First, they are components of the system under study: OPENMLE-GYM uses generative models to construct and screen candidate task packages (Appendix C.1), the SFT and RL corpora consist of trajectories produced by models operating in those environments (Appendix D.1), and the reward-hacking check (Appendix D.4) is model-based. These are part of the contribution, are documented in the main text, and are subject to the quality gates described there. The evaluation benchmarks are not model-generated: we use the official MLE-Bench Lite task set and a ten-task subset of NatureBench, with their original hidden evaluators unchanged. Second, generative AI tools were used by the authors in the research process. We account below for every subtask for which the ICLR 2027 AI policy requires disclosure.

- **Used.** *Generate synthetic data sets*: as described above, the OPENMLE-GYM task corpus and the SFT and RL training corpora are produced with generative models; no evaluation data is model-generated. *Clean and reformat dataset*: generative AI tools assisted with data cleaning and reformatting during the construction of the OPENMLE-GYM corpus. *Implement methods*: they assisted with coding across all components of the stack — OPENMLE-GYM, the sandbox and execution infrastructure, OPENMLE-TRAIN, and OPENMLE-EVO. *Assist with translation*: they assisted with translation while preparing the manuscript.
- **Not used.** Help develop theoretical models or conceptual frameworks; formulate mathematical claims; propose or refine hypotheses; design or provide feedback on research methodology or experiments; interpret results.
- **Not applicable.** Provide critical ingredients for proving mathematical claims; assist in the writing of proofs; support qualitative and thematic data analysis.

Among the subtasks for which disclosure is recommended, we used generative AI tools to edit the manuscript for readability — grammar, spelling, and word choice — to produce figure-generation code, and to assist with the literature survey.

We have reviewed all AI-assisted work. Every number reported in this paper originates from a logged execution record rather than from model output; every citation was checked against its source by an author; every figure was verified against the underlying result tables; translated text was checked against the author's intended meaning; and AI-assisted code was read and executed against the recorded test protocol by the authors before any result derived from it entered the paper. We take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Anthropic. Anthropic's responsible scaling policy, version 2.0. https://www.anthropic.com/responsible-scaling-policy, 2024.

Yuzhu Cai, Zexi Liu, Xinyu Zhu, Cheng Wang, and Siheng Chen. Acegrpo: Adaptive curriculum enhanced group relative policy optimization for autonomous machine learning engineering. *arXiv preprint arXiv:2602.07906*, 2026.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. Mle-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, et al. Sela: Tree-search enhanced llm agents for automated machine learning. *arXiv preprint arXiv:2410.17238*, 2024.

Shangheng Du, Xiangchao Yan, Dengyang Jiang, Jiakang Yuan, Yusong Hu, Xin Li, Liang He, Bo Zhang, and Lei Bai. Automlgen: Navigating fine-grained optimization for coding agents. *arXiv preprint arXiv:2510.08511*, 2025.

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. URL https://arxiv.org/abs/2606.06473.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Damon Falck, Samer Sabri, Anja Surina, Thom Foster, Anya Sims, Sam Devlin, Dylan Rogers, Tantum Collins, Kaloyan Aleksiev, Louis Kirsch, and Edward Hughes. Training AI scientists to replicate research. *arXiv preprint arXiv:2608.13331*, 2026.

Haoyang Fang, Boran Han, Nick Erickson, Xiyuan Zhang, Su Zhou, Anirudh Dagar, Jiani Zhang, Ali Caner Turkmen, Cuixiong Hu, Huzefa Rangwala, Ying Nian Wu, Bernie Wang, and George Karypis. Mlzero: A multi-agent system for end-to-end machine learning automation. *arXiv preprint arXiv:2505.13941*, 2025.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, volume 28, 2015.

Shaokang Fu, Yulong Tao, Linbo Jin, Jiarong Zhao, Qiming Shi, Tianjun Pan, Haonan Li, Chengyu Wang, Jia Wu, and Chengfu Huo. Beyond solution-centric search: Adaptive inquiry and knowledge revision for autonomous ML engineering. *arXiv preprint arXiv:2608.02143*, 2026.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Akshat Gupta, Jermaine Lei, Alexander Lu, Gopala Anumanchipalli, and Leshem Choshen. Automated discovery has no universally superior harness. *arXiv preprint arXiv:2607.18235*, 2026.

Karen Hambardzumyan, Nicolas Baldwin, Edan Toledo, Rishi Hazra, Michael Kuchnik, Bassel Al Omari, Thomas Simon Foster, Anton Protopopov, Jean-Christophe Gagnon-Audet, Ishita Mediratta, et al. Aira_2: Overcoming bottlenecks in ai research agents. *arXiv preprint arXiv:2603.26499*, 2026.

Shengran Hu, Cong Lu, and Jeff Clune. Automated design of agentic systems. *arXiv preprint arXiv:2408.08435*, 2024.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*, 2023.

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.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents from becoming data science experts? *arXiv preprint arXiv:2409.07703*, 2024.

Haechan Kim, Yoonho Lee, Gisang Lee, Chelsea Finn, and Kangwook Lee. WHALE: A simple recipe for joint harness-weight optimization. *arXiv preprint arXiv:2609.00196*, 2026.

Peter Kirgis, Sayash Kapoor, Andrew Schwartz, Stephan Rabanser, David Africa, Konstantinos Voudouris, Viet Nguyen, Toby Pilditch, Magda Dubois, Harry Coppock, et al. Can AI agents conduct open-ended AI research? Early evidence from two case studies. *arXiv preprint arXiv:2607.27191*, 2026.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026a.

Young-Jun Lee, Seungone Kim, Minki Kang, Alistair Cheong Liang Chuen, Zerui Chen, Seungho Han, Taehee Jung, and Dongyeop Kang. Evolution fine-tuning: Learning to discover across 371 optimization tasks. *arXiv preprint arXiv:2606.29082*, 2026b.

Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, Shenghao Feng, Mo Zhao, Fengzhi Qiu, Rui Yang, Mengmeng Zhang, Wenyi Zhu, Yingying Sun, Quan Sun, Shunhao Yan, Danyu Liu, Dawei Yin, and Dou Shen. The fm agent. *arXiv preprint arXiv:2510.26144*, 2025a. URL https://arxiv.org/abs/2510.26144.

Jin Li, Ahmed Murtadha, Zhiyu Wang, Qiwen Chen, William Chen, Yifei Wu, Guan Wang, Andy L. Siy, Jiayi Yang, Mengsha Huang, Wenhao Li, Yixuan Liu, Shuailin Pan, Mingli Yuan, Sen Song, and Yuhao Sun. Praxist: From experimental artifacts to solution lineages. *arXiv preprint arXiv:2608.25955*, 2026.

Yujiang Li, Zhenyu Hou, Xiaohan Jia, Zihan Luo, Zhilei Bei, Rui Lu, Hong Huang, Jie Tang, and Yuxiao Dong. Mle-rl: Reinforcement learning for self-improvement in machine learning agents. *OpenReview preprint*, 2025b. URL https://openreview.net/forum?id=nElqyHPHAz.

Zexi Liu, Yuzhu Cai, Xinyu Zhu, Yujie Zheng, Runkun Chen, Ying Wen, Yanfeng Wang, Siheng Chen, et al. Ml-master: Towards ai-for-ai via integration of exploration and reasoning. *arXiv preprint arXiv:2506.16499*, 2025a.

Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Bo Zhang, Lei Bai, and Siheng Chen. Ml-agent: Reinforcing llm agents for autonomous machine learning engineering. *arXiv preprint arXiv:2505.23723*, 2025b.

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. doi: 10.48550/arXiv.2408.06292. URL https://arxiv.org/abs/2408.06292.

Alisia Lupidi, Bhavul Gauri, Thomas Simon Foster, Bassel Al Omari, Despoina Magka, Alberto Pepe, Alexis Audran-Reiss, Muna Aghamelu, Nicolas Baldwin, Lucia Cipolina-Kun, et al. Airs-bench: a suite of tasks for frontier ai research science agents. *arXiv preprint arXiv:2602.06855*, 2026.

Rachel Metz. OpenAI scale ranks progress toward 'human-level' problem solving. Bloomberg, https://www.bloomberg.com/news/articles/2024-07-11/openai-sets-levels-to-track-progress-toward-superintelligent-ai, 2024. July 11, 2024.

Meredith Ringel Morris, Jascha Sohl-Dickstein, Noah Fiedel, Tris Warkentin, Allan Dafoe, Aleksandra Faust, Clement Farabet, and Shane Legg. Levels of AGI for operationalizing progress on the path to AGI. In *International Conference on Machine Learning*, 2024.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan O. Arik, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. *arXiv preprint arXiv:2506.15692*, 2025.

Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, et al. Mlgym: A new framework and benchmark for advancing ai research agents. *arXiv preprint arXiv:2502.14499*, 2025.

Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

OpenAI. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

OpenAI. Preparedness framework, version 2. https://cdn.openai.com/pdf/18a02b5d-6b67-4cec-ab64-68cdfbddebcd/preparedness-framework-v2.pdf, 2025. April 15, 2025.

Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolutionary program synthesis: A case study on arc-agi. *arXiv preprint arXiv:2507.14172*, 2025.

Rushi Qiang, Yuchen Zhuang, Yinghao Li, Dingu Sagar V K, Rongzhi Zhang, Changhao Li, Ian Shu-Hei Wong, Sherry Yang, Percy Liang, Chao Zhang, and Bo Dai. Mle-dojo: Interactive environments for empowering llm agents in machine learning engineering. *arXiv preprint arXiv:2505.07782*, 2025a.

Rushi Qiang, Yuchen Zhuang, Anikait Singh, Percy Liang, Chao Zhang, Sherry Yang, and Bo Dai. Mle-smith: Scaling mle tasks with automated multi-agent pipeline. *arXiv preprint arXiv:2510.07307*, 2025b.

Rushi Qiang, Changhao Li, Haotian Sun, Yuchen Zhuang, Chao Zhang, and Bo Dai. Matryoshka Agent: Unfolding sub-agents for long-horizon machine learning engineering. *arXiv preprint arXiv:2607.25090*, 2026.

Ben Rank, Hardik Bhatnagar, Ameya Prabhu, Shira Eisenberg, Karina Nguyen, Matthias Bethge, and Maksym Andriushchenko. Posttrainbench: Can llm agents automate llm post-training? *arXiv preprint arXiv:2603.08640*, 2026.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents. *arXiv preprint arXiv:2609.26457*, 2026.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. Paperbench: Evaluating ai's ability to replicate ai research. *arXiv preprint arXiv:2504.01848*, 2025.

Qiong Tang, Tianxiang Sun, Xiangkun Hu, Xiangyang Liu, Yiran Chen, Yunfan Shao, Bobo Li, Changze Lv, Cheng Xu, Chengsong Huang, et al. FARS: A fully automated research system deployed at scale. *arXiv preprint arXiv:2606.31651*, 2026.

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855, 2013.

Edan Toledo, Karen Hambardzumyan, Martin Josifoski, Rishi Hazra, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, et al. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench. *arXiv preprint arXiv:2507.02554*, 2025.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. Automl-agent: A multi-agent llm framework for full-pipeline automl. *arXiv preprint arXiv:2410.02958*, 2024.

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, Hao Cheng, Pengcheng He, Weizhu Chen, Shuohang Wang, Simon Shaolei Du, and Yelong Shen. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

Yuru Wang, Lejun Cheng, Yuxin Zuo, Sihang Zeng, Bingxiang He, Che Jiang, Junlin Yang, Yuchong Wang, Kaikai Zhao, Weifeng Huang, Kai Tian, Zhenzhao Yuan, Jincheng Zhong, Weizhi Wang, Ning Ding, Bowen Zhou, and Kaiyan Zhang. Naturebench: Can coding agents match the published sota of nature-family papers? *arXiv preprint arXiv:2606.24530*, 2026a.

Zhilin Wang, Han Song, Runzhe Zhan, Jusen Du, Jiacheng Chen, Tianle Li, Qingyu Yin, Yulun Wu, Zhennan Shen, Tong Zhu, Yanshu Li, Guanjie Chen, Derek F. Wong, Yafu Li, Yu Cheng, and Yang Yang. EvoPolicyGym: Evaluating autonomous policy evolution in interactive environments. *arXiv preprint arXiv:2607.02440*, 2026b.

Hjalmar Wijk, Tao Roa Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Joshua M. Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Jun Koba Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 66772–66832. PMLR, 2025. URL https://proceedings.mlr.press/v267/wijk25a.html.

Shujin Wu, Cheng Qian, Xiusi Chen, and Heng Ji. Teaching LLMs to self-evolve: Cultivating core meta-skills with reinforcement learning. *arXiv preprint arXiv:2607.21971*, 2026.

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

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Sherry Yang, Joy He-Yueya, and Percy Liang. Reinforcement learning for machine learning engineering agents. *arXiv preprint arXiv:2509.01684*, 2025a.

Xiyuan Yang, Sheikh Sarwar, Jingru Cheng, Zhan Shi, Duanshun Li, Huiyuan Chen, Haiyang Zhang, Xing Fan, Chenlei Guo, Jingrui He, and Zhenyu Liao. Scaling automatic research agents via world models. *arXiv preprint arXiv:2608.12564*, 2026.

Xu Yang, Xiao Yang, Shikai Fang, Yifei Zhang, Jian Wang, Bowen Xian, Qizheng Li, Jingyuan Li, Minrui Xu, Yuante Li, et al. R&d-agent: An llm-agent framework towards autonomous data science. *arXiv preprint arXiv:2505.14738*, 2025b.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025. URL https://arxiv.org/abs/2504.13837.

Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, et al. Learning to discover at test time. *arXiv preprint arXiv:2601.16175*, 2026.

Zhiyuan Zeng, Hamish Ivison, Yiping Wang, Lifan Yuan, Shuyue Stella Li, Zhuorui Ye, Siting Li, Jacqueline He, Runlong Zhou, Tong Chen, Chenyang Zhao, Yulia Tsvetkov, Simon Shaolei Du, Natasha Jaques, Hao Peng, Pang Wei Koh, and Hannaneh Hajishirzi. Rlve: Scaling up reinforcement learning for language models with adaptive verifiable environments. *arXiv preprint arXiv:2511.07317*, 2025.

Ruiyi Zhang, Peijia Qin, Qi Cao, Li Zhang, and Pengtao Xie. Aibuildai: An ai agent for automatically building ai models. *arXiv preprint arXiv:2604.14455*, 2026. URL https://arxiv.org/abs/2604.14455.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, Jingren Zhou, and Junyang Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Bingyang Zheng, Cheng Wang, Rui Ye, Yuzhi Zhang, Linfeng Zhang, Weinan E, Siheng Chen, and Yanfeng Wang. Toward ultra-long-horizon agentic science: Cognitive accumulation for machine learning engineering. *arXiv preprint arXiv:2601.10402*, 2026.

A Problem Formulation

Evolution is a loop in which a model repeatedly modifies a candidate program according to execution feedback; *meta-evolution* additionally trains that model on executed search experience, so that the proposer of modifications itself improves. OpenMLE studies one round of this step.

Evolutionary search. Following MLE-Bench (Chan et al., 2024), a task τ provides a specification, public data, a submission format, and a hidden evaluator R_τ that scores submissions against held-out answers. At each step the search controller chooses an operator a_t (Draft, Improve, Debug, or Crossover) and a context c_t of previously executed programs and their feedback; the model writes a new program, which is executed in a sandbox, and the run returns the best valid program found within the budget:

$$p_t \sim g_\theta(\cdot \mid \tau, a_t, c_t), \qquad p^\star = \operatorname*{arg\,max}_{p_t \text{ valid, within budget}} v(p_t). \tag{2}$$

Here v is the validation score, normalized so that higher is better. On MLE-Bench Lite, OpenMLE-Evo obtains v from the sandbox, which executes each program on a separate validation package of τ with its own public data and held-out answers; search and final selection use only v, and only $p^\star$ is then executed on the benchmark's test package, whose hidden score $R_\tau(p^\star)$ is reported.

Learning to evolve. On OpenMLE-Gym tasks, which exclude the 75 MLE-Bench competitions (Appendix C.1), the hidden score is available as a training signal. Both post-training stages make programs with strong execution outcomes more likely, schematically

$$\max_\theta \ \mathbb{E}\big[\, w \cdot \log g_\theta(p \mid \tau, a, c) \,\big]. \tag{3}$$

In SFT the examples are execution-grounded teacher outputs with $w = 1$; in RL, p is sampled from the model being trained and w is the entropic advantage of Eq. 1, applied within a clipped policy-gradient objective; in both stages the likelihood covers the full response, including reasoning.

B Limitations and Future Work

OpenMLE trains the model that proposes revisions, while the task evaluators and the search harness that composes its operators are specified by us. This division keeps every learning signal verifiable, and it points to three extensions. First, training rewards are grounded in executed outcomes; also rewarding the choice of which hypotheses and research directions deserve further compute would let the improver plan beyond the next revision. Second, the harness decides which parent to expand and what experience each operator sees; learning these decisions as well would make the evolutionary system itself an object of training. Third, OpenMLE-Gym tasks improve machine learning solutions for given problems; extending the environments to the training of language models themselves would bring the loop closer to AI training AI in its full sense.

C OpenMLE-Gym details

C.1 Task construction and selection

The 5,758-task pool contains 156 curated anchors, 3,362 dataset-derived tasks, and 2,240 competition-derived tasks. The anchors comprise 139 Kaggle competitions, 9 Kaggle datasets, and 8 tasks from other public archives. Every accepted task contains a specification, public inputs, deterministic preparation, a sample submission, a private evaluator, and declared resource requirements. The construction pipeline checks file schemas and target definitions, materializes the package, scores the sample submission with the generated metric, and requires a finite score. Complete packages are released for the 1,415 tasks whose recorded source license permits redistribution; for the remaining 4,343 tasks the release contains the preparation and metric code, which rebuilds each package from source data obtained under the source's own terms.

Execution alone does not establish semantic validity. A second gate checks whether the task uses the supplied data, whether splits and metrics are coherent, whether the target is nontrivial, and whether

the description agrees with the artifacts; its rubric covers degenerate targets, label leakage, insufficient data, superficial use of the source assets, and malformed splits, labels, or metrics. The competition builder skips the 75 MLE-Bench competitions, which include all 22 Lite tasks, by exact slug match, and no task identifier or source URL among the 5,758 tasks matches any of them.

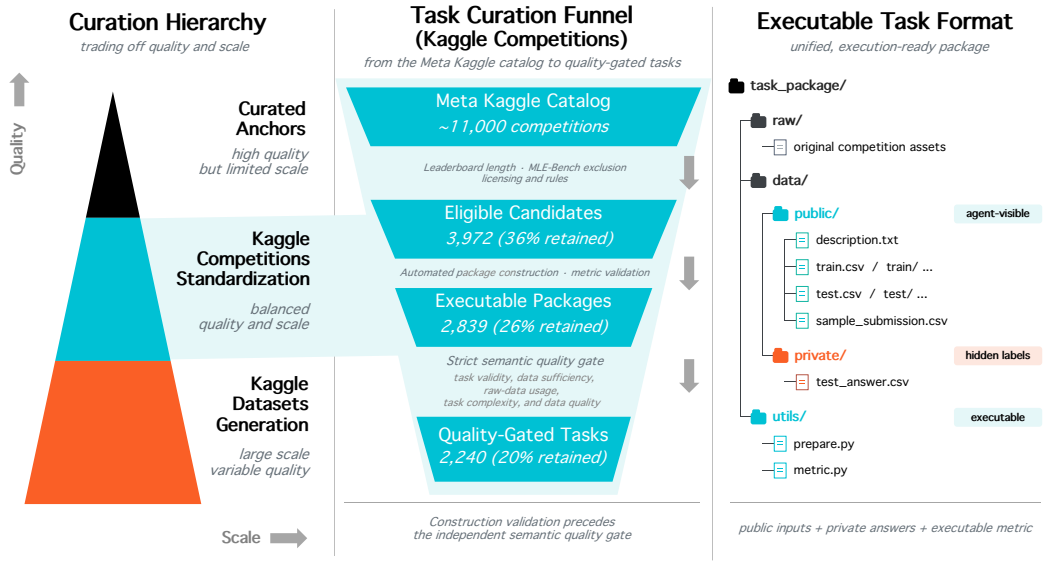

Figure 9: OPENMLE-GYM task curation and executable format. **Left:** source hierarchy. **Middle:** Kaggle-competition filtering from the Meta Kaggle catalog to quality-gated packages. **Right:** public task inputs, private answers, and executable utilities.

Competition-construction state machine. The builder consumes the 3,972 candidate slugs that remain after upstream filtering of the 10,907 competitions in the Meta Kaggle catalog (Figure 9, middle). Independent competitions can be dispatched concurrently, while each competition follows an ordered state machine so that downstream stages run only after their prerequisites succeed. For each slug, the pipeline downloads and unpacks the competition archive, preserves the source assets, and retrieves the competition overview and dataset description. A tool-assisted file-perception stage then enumerates the local tree, probes tabular files for structural summaries and sampled rows, and reads textual documentation. The resulting inventory and competition metadata jointly ground the generated task description, avoiding reliance on either web metadata or filenames alone (Figure 10).

Executable construction and validation. Conditioned on the generated description, the pipeline produces and executes `prepare.py` to split the labeled raw data deterministically into 80% training and 20% held-out test data (unlabeled original test files are not used), copy associated modality files, expose participant-visible inputs, and isolate held-out answers. Execution success alone is insufficient: the builder checks for the task description, training data, test inputs, sample submission, and private test answers before accepting the package. A failed attempt clears its partial processed outputs and returns the execution error to the next generation attempt, preventing malformed intermediate state from being reused. After preparation succeeds, the pipeline generates the task-specific `metric.py` from the task description together with samples of the submission and private-answer schemas. Metric generation and metric validation are separate stages: the latter dynamically loads the generated evaluator and scores the sample submission against the private answers; execution errors fail this gate before semantic assessment.

Metadata profiling and semantic quality assessment. Post-construction profiling annotates modality and task type from the task description, measures raw and processed data size, classifies the expected CPU/GPU requirement, and records metric-validation output; the package metadata also records the metric direction and, where available, theoretical and leaderboard score bounds.

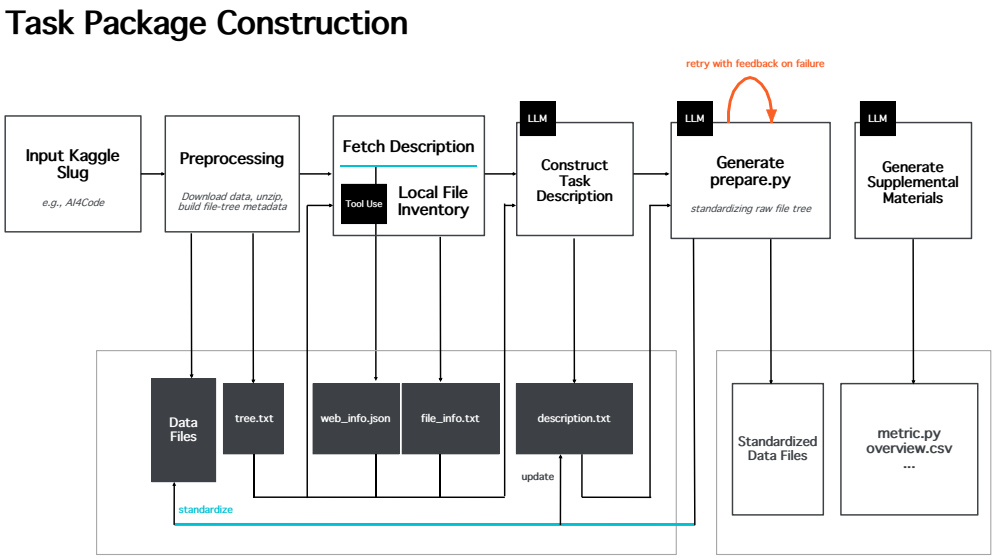

Figure 10: Automated competition task construction. Competition metadata and local-file evidence ground package generation; execution feedback revises failed preparation before metric validation and semantic gating.

The quality evaluator first inspects package structure and executes the metric smoke test; missing or malformed files, misaligned submission rows or identifiers, target columns in the public test inputs, or a failing or non-finite metric deterministically lead to exclusion. For packages that pass these hard gates, the semantic judge receives the description, `prepare.py`, train/test sizes, processed-data size, auxiliary file types, sampled public and held-out rows, available raw inventory, and metric result. It returns scores for task validity, data sufficiency, raw-data usage, task complexity, and data quality, together with one of `recommended` (mean score at least 4 on the 0–5 scale), `conditional`, or `not_recommended`; final aggregation admits only metric-valid tasks with the `recommended` decision.

Composition. Across the three sources the pool covers tabular, text, time-series, image, and other modalities, with classification and regression complemented by more engineering-intensive task types; after canonicalizing labels, 11% of tasks are multimodal and classification and regression together account for 87% (Figure 2). The distribution plots use mutually exclusive normalized groups: tasks containing multiple modalities are assigned to Multimodal, compound task labels to Multitask, and rare labels to Other; package size is measured from constructed task contents.

C.2 Sandbox Execution Infrastructure

A scheduler records each job and dispatches it to an isolated CPU or GPU worker container. The worker runs the candidate in a fresh job workspace with the package's public directory as its data input, and, once the program has exited and written a submission file, scores that file in a separate process in which the task's `metric.py` validates the submission schema and then scores it against the private answers. The scheduler, not the generated program, owns worker leases, time limits, cancellation, and terminal job states. The evidence record distinguishes success, runtime error, timeout, missing code, missing submission, and scoring failure. Logs, outputs, scores, resource use, and generated artifacts are persisted with the program node so that an apparent improvement can be replayed.

Table 4 states the package contract and the evidence recorded for each element.

Table 4: Task-package contract and the corresponding verification evidence.

Contract element	Required property	Recorded evidence
Public inputs	serve as the candidate's data directory	data path, preparation log
Candidate program	completes within explicit resource limits	exit status, runtime, logs, code hash
Submission artifact	exists and satisfies the required schema	artifact path, validity status
Private evaluator	returns a finite direction-aware task score	raw and signed score
Isolation boundary	the candidate runs in its own job workspace and is scored after it exits	worker launch configuration, job timestamps

C.3 Run records and replay manifest

Agent runtimes can appear better by receiving more retries, longer execution, more parallel workers, or a different evaluator, so comparing them requires a record above the final task score. Table 5 lists the fields that each OpenMLE-Evo run records.

Table 5: Replay manifest recorded for each OpenMLE-Evo run.

Layer	Recorded fields	Where recorded
Workload	task list, validation and test package paths, metric direction	task input table, per-task record
Model	model identifier, decoding parameters, response-length limit	resolved run configuration
Controller	operator and parent-selection settings, memory limits, step and time budgets	resolved run configuration
Execution	execution timeout, CPU/GPU resource type, code hash, data path, worker, job status and timestamps	run configuration, sandbox job database
Scheduling	execution mode, requested and resolved worker counts, worker of each node	run manifest, per-node record
Evidence	seed, node and parent IDs, operator, raw model response, execution logs, scores, selected and submitted programs	run configuration, per-node and per-task records
Accounting	prompt/completion tokens, model and sandbox time, status counts	per-node and per-task records

Because every node records its parent identifiers and raw score, and every task record the packages it was scored on, nodes scored by a defective evaluator and their descendants can be located from the record and re-executed from the stored programs, without relying on the model's prose account of what occurred.

D OpenMLE-Train details

D.1 SFT data generation

Figure 11 places the four operators in the training and inference loop.

The parallel path samples complete Draft solutions, executes every candidate, removes duplicate-score copies within a task, and retains at most the four top-scoring valid programs per task. It contributes 17,245 full-response examples. The evolutionary path stores true parent links and extracts local Draft, Improve, or Crossover segments together with their consecutive Debug descendants. A Draft endpoint must reach a positive score, an Improve endpoint must beat its parent, a Crossover endpoint must beat its stronger parent, and every retained endpoint must reach medal level. It contributes 9,014 retained trajectory-step examples, producing 26,259 examples in total. Collection in both paths is budget-adaptive: sampling on a task stops once its accepted-candidate quota is met or its execution budget is spent, so tasks with sparse successes receive more attempts.

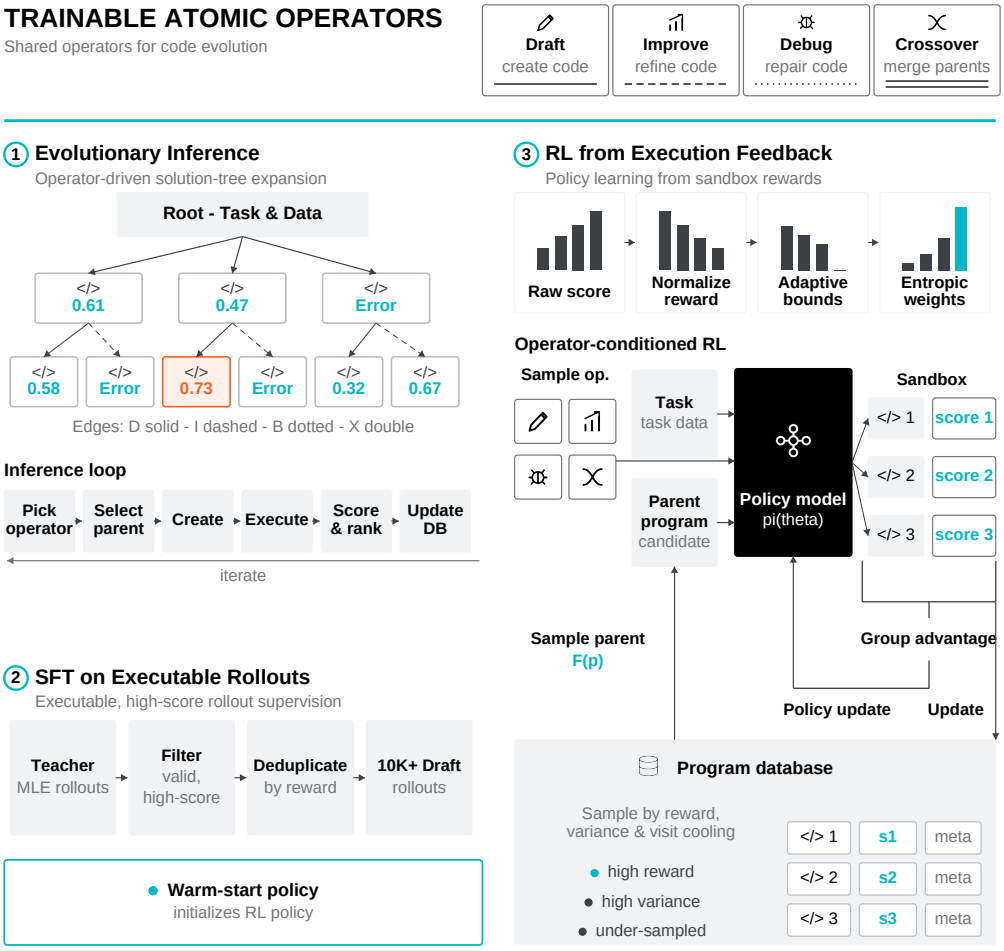

Figure 11: The OpenMLE training and inference workflow. Trainable atomic operators are composed by evolutionary inference, warm-started through executable SFT rollouts, and further optimized with online RL from execution feedback. Panel (2) summarizes the parallel Draft path (17,245 examples); with the evolutionary path of Figure 12, the SFT corpus has 26,259 examples. The same typed operator interface appears in all three stages, so the model or the harness can be swapped while the other is held fixed.

Parallel-path sampling. The teachers first generate multiple independent Draft solutions for each task, and every candidate is executed in the task sandbox. The first collection batch uses GLM-4.7: among candidates with valid scores for the same task, duplicate-score solutions are removed, the remainder are ranked by score, and at most the top four are retained, giving 11,519 examples. The second batch uses GLM-4.7 and Qwen3-30B-A3B-Thinking-2507; after execution and output-length screening, candidates from both teachers are ranked jointly within each task, a GLM-4.7 candidate is retained when it falls in the joint top four, and a Qwen candidate only when it ranks first overall, so each task still contributes at most four examples. This batch contributes 5,726 examples (5,075 GLM-4.7 and 651 Qwen), for 17,245 full-response examples in total.

Evolutionary-path sampling. Complete solutions expose final programs but do not directly supervise how execution feedback should be used to revise an existing solution. GLM-4.7 therefore drives AIRA-Evo searches with all four operators, producing search trees with parent relations, program versions, execution feedback, and task scores. A local segment begins at a Draft, Improve, or

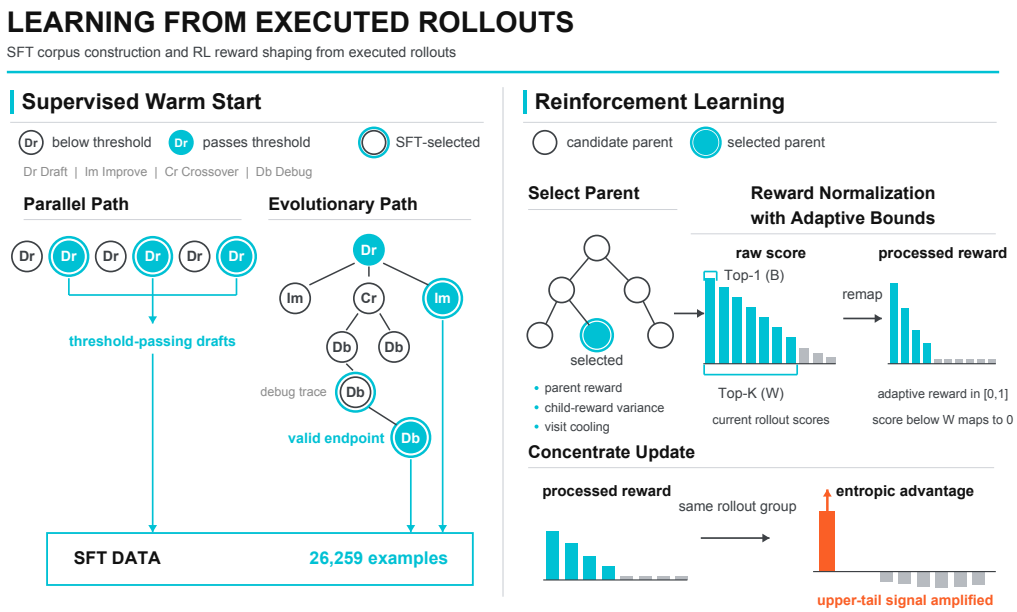

Figure 12: Learning from executed rollouts. Filled nodes pass their path's check (parallel path: a valid score among the task's four best distinct scores; evolutionary path: a valid score), and ringed nodes enter the SFT corpus. In the evolutionary path a valid endpoint can emerge only after repeated DEBUG steps; when that endpoint passes the segment checks, including medal level, the pipeline traces back to the preceding non-debug operator and retains the useful steps of that repair trace. Selected examples from both paths form the 26,259-example released SFT corpus under the budget-adaptive stopping rule of Appendix D.1. For a chosen operator, RL selects a parent using parent reward, child-reward variance, and visit cooling; Top-1/Top-K adaptive bounds map the raw scores of a rollout group to processed rewards, with scores below the resolved lower bound clipped to zero, and entropic advantages then amplify the upper-tail learning signal.

CROSSOVER node and follows its consecutive DEBUG descendants along true parent–child edges until a branch reaches another non-debug node. A DRAFT segment must end with a positive score, an IMPROVE segment must outperform its parent, and a CROSSOVER segment must outperform the better of its two parents; retained endpoints must additionally reach a medal level, i.e. at least the bronze threshold of the task's public Kaggle leaderboard under the Kaggle medal rules used for evaluation (Appendix F.1); an endpoint on a task without a leaderboard has no medal level and is not retained. Single-step segments whose roots are themselves medal-level endpoints are retained directly. For multi-step segments, DeepSeek-V4-Pro reads each complete segment, including task constraints, parent code, endpoint code, adjacent code changes, and stepwise execution feedback, and keeps a step only when its strategy, program structure, or error repair is inherited by later steps and contributes to the final solution. The annotator runs at temperature 0, returns a fixed JSON schema that covers every input step, and discards cosmetic edits and blind retries; its system prompt is released with the SFT code. Every retained segment therefore ends at an execution-verified, medal-level endpoint, and the steps before it are kept by this inheritance criterion. The evolutionary path contributes 9,014 trajectory-step examples.

Figure 13 shows how the corpus splits by supervision type, operator, and length.

The final operator counts are 19,436 Draft, 1,741 Improve, 742 Crossover, and 4,340 Debug examples. Full responses have a median full-message length of 8,407 tokens; trajectory steps have a median of 14,051 because they include executed parents, diffs, and feedback. All samples are rendered through the target chat template, deduplicated on normalized message content, and filtered at a 32,768 token context cutoff.

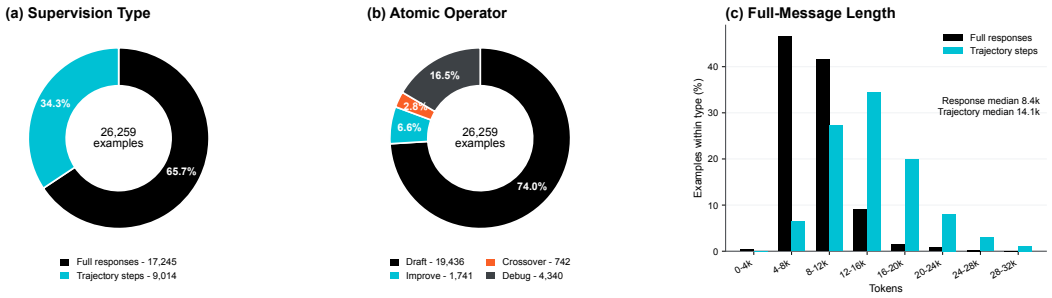

Figure 13: Distribution of the final SFT corpus. (a) Relative proportions of full-response and trajectory-step supervision among the 26,259 examples. (b) Distribution of the four atomic operators. (c) Full-message length distributions for the two supervision types; each bar reports the percentage within its supervision type.

Table 6: Shared program-transformation interface. The core input is the task and parent state each operator receives; a search harness may add further context, which for OPENMLE-EVO is the operator-conditioned memory of Table 11.

Operator	Core input	Intended transition
DRAFT	task specification and public assets	construct a complete executable baseline
IMPROVE	valid parent, score, artifacts, and feedback	increase task quality while preserving validity
DEBUG	failing parent, logs, and error type	repair execution or submission failures
CROSSOVER	two executed parents and their feedback	recombine useful mechanisms without copying full histories

D.2 SFT DATA SCALING

Every configuration behind Figure 6 uses the OPENMLE-EVO harness, the official 22-task split, three runs, and a 12-hour per-task budget on one RTX 4090 capped at 12 GB VRAM. The SFT subsets are drawn by task from the corpus described in Appendix D.1, each checkpoint is trained for the same number of optimizer steps at the same global batch size, so every checkpoint sees the same number of training examples and smaller subsets are revisited more often, and the untrained-backbone reference points are the corresponding rows of Table 13. All points are SFT-only checkpoints; the full-corpus points are the +SFT rows of Table 1, the checkpoints from which RL starts.

Figure 14 plots the other two metrics. Every fraction improves on the untrained backbone in all three metrics, and the full corpus gives the highest Medal Average on both backbones. On Qwen3-30B-A3B, Valid Rate rises at every fraction, from 17.33 to 20.67 tasks out of 22; Medal Average is 42.42% at an eighth of the corpus, 39.39% at a quarter and at half, and 50.00% with the full corpus; and Human Rank moves from 0.5573 through 0.5733, 0.5649, and 0.5806 to 0.6375. On Qwen3.6-35B-A3B, Medal Average goes from 39.39% to 40.91%, 46.97%, 46.97%, and 48.48%, Human Rank from 0.5828 to 0.6068, 0.6497, 0.6654, and 0.6614, and Valid Rate from 19.67 to 21.00–21.67 tasks at every fraction, close to the 22-task ceiling.

D.3 SFT AND RL TRAINING CONFIGURATION

Table 7 lists the SFT and RL settings.

RL validation. The validation curves in Figure 15 are computed every five RL steps on the 176 validation tasks. Each task receives one DRAFT program, sampled at temperature 0.7 and top-p 0.95 with up to 32,768 response tokens. A medal is counted when the task score reaches a medal threshold computed from the task's public leaderboard with the Kaggle rules used by MLE-Bench.

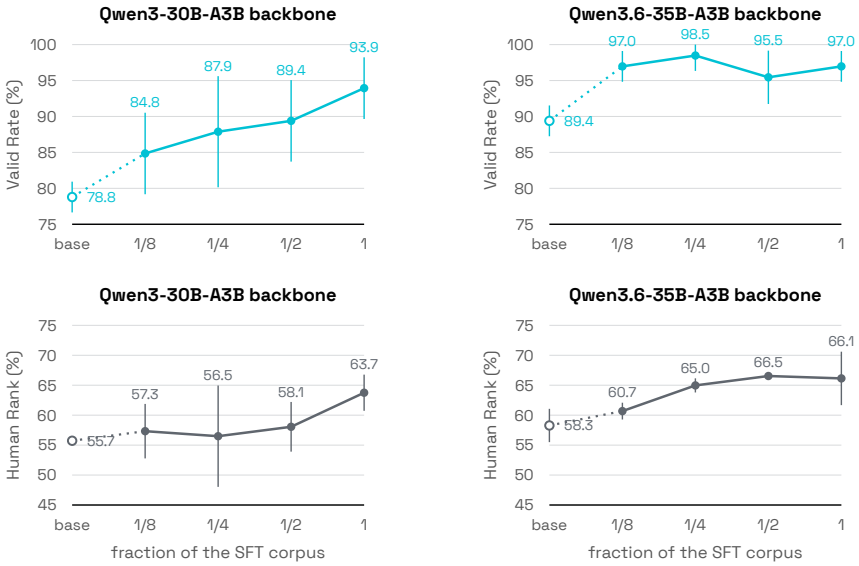

Figure 14: SFT data scaling for the two metrics not plotted in the main text. Rows are Valid Rate (percent of the 22 tasks) and Human Rank ($\times 100$); columns are the two backbones. Open markers are the untrained backbone under the same harness, and error bars are ± 1 standard deviation over three runs. Medal Average is Figure 6.

Table 7: Core SFT and execution-grounded RL configuration, shared by OPENMLE-30B and OPENMLE-35B.

Stage	Setting	Value
SFT	training / precision	full-parameter / bfloat16
SFT	context cutoff / global batch	32,768 tokens / 128
SFT	learning rate / schedule	3×10^{-5} / cosine decay to 0 after 10% warmup
SFT	epochs	3 on the full corpus
RL	objective	GSPO (Zheng et al., 2025), clip $\epsilon = 3.5 \times 10^{-4}$, truncated importance sampling; KL and entropy coefficients 0
RL	advantage	entropic, per rollout group (Appendix D.4)
RL	prompts / samples per prompt	16 / 16
RL	global batch / optimizer steps per rollout	128 / 2
RL	operator probabilities	DRAFT .50, IMPROVE .17, DEBUG .17, CROSSOVER .16
RL	temperature / maximum response	1.0 / 24,576 tokens
RL	sandbox time limit per candidate	1 hour
RL	learning rate	10^{-6}, constant
RL	optimizer	Adam, $\beta_1 = .9$, $\beta_2 = .98$, weight decay .1

D.4 REWARD PROCESSING AND ASYNCHRONOUS ROLLOUT

For a direction-normalized score $\tilde{s}$, the static diagnostic reward is

$$r_{\text{base}}(\tilde{s}) = \text{clip}\left(\frac{\tilde{s} - b_{\text{worst}}}{b_{\text{best}} - b_{\text{worst}}}, 0, 1\right)^{\alpha},$$

where b_{best} and b_{worst} are the best and worst scores on the task's public leaderboard (the metric's theoretical limits when no leaderboard is available) and $\alpha = 1$. Figure 15 reports this static reward as the base reward.

Online training replaces these broad extrema with task-local bounds. Let $x_{(1)} \geq \cdots \geq x_{(K)}$ be the direction-normalized scores of all successful programs stored for the task in the RL program database,

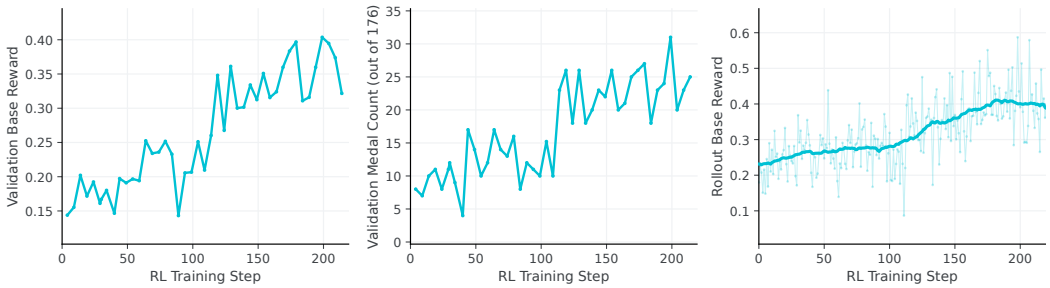

Figure 15: RL training curves for OPENMLE-35B: validation base reward and validation medal count on a held-out set of 176 OPENMLE-GYM tasks (Appendix D.3), and rollout base reward with its running mean, against RL training step.

together with the current group, and let $m = \min(16, K)$. The bounds are

$$B = \min\big(x_{(1)},\, B_{\text{lim}}\big), \qquad W = \max\big(x_{(m)} - 0.25\,(x_{(1)} - x_{(m)}),\, W_{\text{lim}}\big),$$

where B_{lim} and W_{lim} are the outermost theoretical or leaderboard limits of the metric, so the range never leaves the valid score range. The processed reward r_{proc} applies the map above on $[W, B]$ instead of $[b_{\text{worst}}, b_{\text{best}}]$: non-finite candidates and those below W receive zero, and when only one score is available (a task's first success) it receives one. Bounds are resolved once all G candidates of a group have been scored and are shared by the whole group.

For the revision operators IMPROVE, DEBUG, and CROSSOVER, the reward of candidate i adds a progress bonus relative to its parent,

$$r_i = r_{\text{proc},i} + \lambda\, \max\big(r_{\text{proc},i} - r^{\text{par}}_{\text{proc}},\, 0\big), \qquad \lambda = 0.5,$$

where $r^{\text{par}}_{\text{proc}}$ is the processed reward recorded when the parent was scored and, for CROSSOVER, the larger of the two parents' rewards. A candidate with zero processed reward, or whose program text is identical to a parent's, receives $r_i = 0$; for DRAFT, $r_i = r_{\text{proc},i}$. A response without extractable code receives $r_i = -1$, and a candidate flagged by the pre-execution check of Appendix D.5 receives $r_i = -0.5$. All candidates in a group share one parent, so the bonus widens the reward gaps among candidates that score above the parent.

For a rollout group of size G, the advantage follows the entropic objective of TTT-Discover (Yuksekgonul et al., 2026):

$$A^{\text{ent}}_i = \frac{\exp(\beta r_i)}{\frac{1}{G-1}\sum_{j \neq i} \exp(\beta r_j)} - 1,$$

where β is solved for each group by bisection (60 iterations, $\beta \leq 10^6$) so that the weights $q_i \propto \exp(\beta r_i)$ satisfy $\mathrm{KL}(q \,\|\, \mathrm{Unif}_G) = \ln 2$; a group whose rewards are all equal receives zero advantage. Rewards are max-centered before exponentiation for numerical stability, which leaves the ratio unchanged. Adaptive normalization prevents useful differences from collapsing, whereas the entropic transform reallocates gradient weight within the verified group. Neither mechanism can repair an incorrect task evaluator.

Program execution has a heavy-tailed runtime distribution. A synchronous step waits for its slowest sandbox even when other groups are ready. OpenMLE launches generation-and-execution groups independently and places completed groups in a trainer queue while retaining within-group advantages. Two representative asynchronous runs keep task exposure within ± 2 steps of the median, with coefficients of variation of 1.56% and 2.06%.

The implementation records per-task exposure and queue delay.

Parent sampling during RL. RL over evolutionary operators must also choose the state an operator acts on. Rollout groups draw their parents from the program database by a fitness-proportional utility over the parent's processed reward, the variance of its children's rewards, and a visit-based cooling term; the selected parent, operator, and prompt are then fixed for the whole group.

D.5 DETECTING REWARD HACKING DURING RL

Execution grounding removes preference noise but not metric gaming. On difficult tasks, reward can plateau at a low level because the policy discovers a degenerate shortcut rather than a solution; the representative case we observed was a candidate that loads the provided sample submission, shuffles it, and submits it. Sandbox execution scores such a program without complaint, because it is a valid submission.

The mitigation is a pre-execution check applied to every RL candidate. Before a candidate is dispatched to the sandbox, a separate judge model (o3-mini) reads the program and flags code that guesses or returns random or constant predictions, relies on hard-coded rules instead of learning from data, produces empty or trivial predictions, ignores the provided data, or trains no model. A flagged candidate skips execution entirely and receives reward -0.5. This saves the sandbox budget that a degenerate program would otherwise consume and gives the policy an explicit negative signal. Table 16 lists metric gaming together with the other observable failures.

E OPENMLE-EVO INFERENCE DETAILS

E.1 OPERATOR PROMPT TEMPLATE AND ONE ATOMIC OPERATION

This appendix answers, concretely, what a single unit of training and search is: its inputs, the prompt template, the artifact the model returns, the sandbox evidence, and the fields written to the program database.

Definition. An atomic operation is one model call that emits one complete replacement program, followed by exactly one sandboxed execution of that program. It is not a token-level action, not a file-level diff, and not a multi-step trajectory. Formally, with task τ, operator a, and operator context c built from already-executed parents, the call produces $p \sim g_\theta(\cdot \mid \tau, a, c)$; the sandbox executes p and returns a score s and an execution record z (status, log, runtime). In search on MLE-Bench Lite, s is the validation score v of Appendix A: the task evaluator scores the submission of p on the validation version of τ, and v is read from the sandbox result rather than from the program's printed output. Only if the sandbox returns neither a score nor a failure status does one analysis call to the same model read the log and return a bug flag and a metric. In RL on OPENMLE-GYM tasks, s is the task evaluator's score from which the reward is computed. Table 6 lists what c contains for each operator.

The IMPROVE template. The four operators share the execution-and-storage contract and differ in parent structure, guidance text, and retrieved memory (Table 11). We reproduce IMPROVE in full; double-braced fields are substituted at runtime, the two `public_*` fields carry the task's own system instructions and output instructions (data location and submission format) from the benchmark input, and the empty-memory fallback is shown so the template does not depend on one particular trajectory. The system message is adapted from AIRA-dojo (Toledo et al., 2025). The DRAFT, DEBUG, and CROSSOVER templates and the memory-synthesis and analysis prompts are included in the supplementary code.

```
[SYSTEM MESSAGE]
You are a Kaggle Grandmaster attending a high-stakes competition. In order to win this
competition, you need to come up with an excellent and creative plan for a solution and
then implement this solution in Python, that improves upon an existing solution to the
task.
{{public_system_prompt}}

[USER MESSAGE TEMPLATE]
We will now provide a description of the task.
Task Description: {{task_description}}
Data Description:  {{data_description}}

A previous solution was attempted with the following results:
Code:     ```python {{previous_code}} ```
Feedback: ``` {{previous_terminal_output_with_official_score_redacted}} ```

Improve Guidance:
Based on the previous attempt, its score, and execution output, please analyze what
worked well and what didn't, then improve the solution. Propose exactly one improvement
idea that is different from the previously explored improvement ideas. Keep the
```

```
evaluation method consistent across iterations.

Search Context:
Targeted memory for this improvement: none available.

Execution Budget:
Be aware of the running time of the code, it should complete within
{{execution_timeout}}. Your code will be executed in a sandbox after generation. A
single sandbox run must finish within {{execution_timeout}}. Always choose a
validation/training strategy that leaves enough time to write ./submission.csv.

Submission Check:
Before finishing, verify that ./submission.csv exists, has the same row count and
required columns/order as sample_submission.csv when available, preserves the sample
id/order, and contains no NaN/inf values. When building features, align train/test
columns explicitly and handle missing values before model fitting or prediction.
{{public_user_prompt}}
```

When memory is available, the *Search Context* block instead contains the rendered memory followed by four instructions: preserve what already works in the selected parent, use the ancestors to see which recent changes helped or hurt, use the siblings to avoid nearby failed alternatives and to borrow useful details, and treat board statistics only as weak global guidance.

What the framework does with the response. The framework extracts the Python program from the response and stores it as a new candidate. The sandbox runs the candidate on the public inputs of the task package (for OPENMLE-EVO on MLE-Bench Lite, the task's validation package; Appendix A) under the declared resource limits, scores the submission against that package's private answers, and returns execution status, signed score, evaluator result, and execution log. The framework then inserts one row containing the child code, score, processed reward or fitness, feedback, operator type, parent identifier, raw model response, and execution metadata. Table 8 gives the full field list. The execution log shown to the model has the evaluator's score lines removed; each node's validation score v reaches the model only through the rendered memory, and on MLE-Bench Lite the benchmark's test answers are never used during search.

Two grounded instances. On *nomad2018-predict-transparent-conductors*, a CROSSOVER call at search step 81 received two executed parents — one contributing atomic properties, dynamic covalent edges, and unit-cell volume at validation RMSE 0.06309, the other a robust parser for irregular `.xyz` geometries at validation RMSE 0.06573 — together with bounded negative evidence marking an RDF-cache `TypeError` and a 3328×94 feature mismatch observed in nearby siblings. The single emitted child combined the physics-informed model with the robust parser and reached validation RMSE 0.06087 and held-out RMSE 0.05410. On *leaf-classification*, a late IMPROVE call received one valid multimodal parent and produced the largest single jump of that run by upgrading the fused representation to ConvNeXt-Tiny. Both are search-time calls on evaluation tasks with the same interface as the SFT and RL examples: one operator call returns one complete program, which is executed once and recorded as one node.

Why one controller call is also one RL sample. Because an operation is complete and independently executable, its sandbox result can be attached to exactly the tokens that produced it. A rollout group fixes (τ, a, c) and samples G candidates; each is executed independently; the resulting advantage updates only that call's generated tokens. Section 3 states what this design gives up.

E.2 STRUCTURED EXPERIENCE RECORDS

The controller stores deterministic records rather than relying on a narrative transcript. Every record is bound to a schema version so that later code can distinguish missing historical fields from explicit null values. Large programs and artifacts may live in content-addressed storage, but their hashes and logical roles remain in the node record.

The card states what one evaluated node achieved; the board states where that node sits in the task population. Cached semantic summaries are stored in a separate field and never overwrite the underlying deterministic fields: every score, parent delta, and runtime shown to the model is rendered from the deterministic record, and the summary is appended as free text.

Table 8: Node-level experience-card schema.

Component	Representative fields	Search or audit role
Identity and lineage	schema version, node ID, step, operator, parent IDs, generation	reconstruct incoming search edges
Observed outcome	score, reward, status, error signature, submission validity	separate quality from execution failure
Resource accounting	sandbox/model time, prompt/completion tokens, cost	expose resource use of the attempted method
Method characterization	imports, detected family, prior family count	estimate novelty and repeated directions
Derived signals	parent delta, novelty, rank, incumbent flag, selection utility	preserve the inputs to parent selection
Semantic evidence	plan, execution analysis, cached grounded summary	render bounded reusable context
Artifact references	program, output, log and submission hashes	permit replay and provenance checks

Table 9: Task-global experience-board schema.

Component	Aggregated state	Controller-visible purpose
Global incumbent	node count, best valid node and score	identify current solution frontier
Method coverage	family counts, best nodes, failure rates, underexplored families	balance exploitation and exploration
Progress and failures	score history, deltas, repeated errors, status/operator counts	detect stagnation and recurring faults
Topology	parent graph, ranks, novelty and incumbent flags	retrieve ancestors and siblings
Resources	runtime totals, token counts, parent-selection trace	audit budget and selection behavior

E.3 Parent selection and memory synthesis

Every sandbox evaluation creates an immutable program node containing parent identifiers, operation, complete code, status, score, parent-relative delta, runtime, token counts, artifacts, and a compact experience card. The task-level experience board aggregates the incumbent, method-family coverage, recent progress, recurring failures, topology, and resource use. The database remains the durable record; rendered model context is only a bounded view. Tables 8 and 9 give the two schemas, and Figure 4 shows how a card feeds the next operation.

Experience-guided parent selection. Original AIRA-Evo derives sampling probabilities primarily from normalized fitness, so expansion concentrates on already strong nodes and ignores how much a node improves on its ancestry or whether it opens an underexplored family. OpenMLE-Evo turns the card metadata into three factors and samples a parent from the candidate set $\mathcal{I}$, the valid nodes held in the population (Table 10), with

$$U_i = \lambda_v \tilde{v}_i + \lambda_\Delta \widetilde{\Delta}_i + \lambda_n \nu_i, \qquad P(i \mid \mathcal{I}) = \frac{\exp(U_i/T)}{\sum_{j\in\mathcal{I}} \exp(U_j/T)}, \tag{4}$$

where $(\lambda_v, \lambda_\Delta, \lambda_n) = (1.0, 0.4, 0.25)$ and $T = 1$. The quality factor $\tilde{v}_i$ is v_i min–max normalized over $\mathcal{I}$ in the direction of the task metric, or 0.5 when all scores are equal. The progress factor is $\widetilde{\Delta}_i = \Delta_i / \max_{j\in\mathcal{I}} \Delta_j$ (0 if no node improved), where Δ_i is the positive part of the improvement of v_i over its strongest parent and is 0 when no parent has a valid score, as for Draft nodes. The novelty factor is $\nu_i = 1/\sqrt{1 + N_{f(i)}}$, where $N_{f(i)}$ counts the other nodes of $\mathcal{I}$ in the automatically detected method family of i. Quality carries the largest weight; progress and novelty shift probability toward nodes that improved on their parents or belong to less-explored families. Improve samples one parent and Crossover two parents without replacement from $\mathcal{I}$. A child that fails to execute is

passed to a chain of DEBUG calls whose memory retrieves attempts with the same error signature. The final submission is the valid node with the highest v (Eq. 2); it is never sampled from the softmax.

Table 10: OPENMLE-EVO controller configuration.

Setting	Value
Population	one island of at most 500 valid nodes; after the first generation, a valid child joins if its v is at least the island mean
Generations	5 individuals each; the first generation consists of DRAFT calls; later individuals use CROSSOVER with probability 0.5 from the third generation on and IMPROVE otherwise
Repair	a failing child receives up to 10 consecutive DEBUG calls within 6 h of execution time; an unrepaired child is discarded
Parent selection	Eq. 4 with $(\lambda_v, \lambda_\Delta, \lambda_n) = (1.0, 0.4, 0.25)$ and $T = 1$
Retrieved memory	IMPROVE: 3 ancestors and 3 siblings; CROSSOVER: 2 ancestors and 2 siblings per parent
Limits	7,200 s per sandbox run; search stops at 800 steps, 100 generations, or the 12-hour per-task budget

Table 11: Operation-conditioned memory retrieval.

Operation	Retrieved state	Role
Draft	no inherited node memory	start an independent branch
Improve	parent, recent ancestors, top siblings, board statistics	preserve working mechanisms and contrast nearby alternatives
Crossover	two parents and bounded evidence around both branches	identify complementary mechanisms and conflicts
Debug	current failure, matching error signatures, recent repairs	reuse concrete fixes without replaying the full tree

On-demand memory and operator-conditioned context. In original AIRA-Evo, an analysis call reads the execution output of every evaluated node and writes the summary that later prompts include, spending budget on nodes no later operator selects and producing a summary before the decision context that should shape it exists. OPENMLE-EVO makes no per-node analysis call when the sandbox returns a score or a failure status, and defers synthesis until an IMPROVE, CROSSOVER, or DEBUG call has selected its nodes; it then calls the model only for those parents and their retrieved ancestors, siblings, or error-related attempts, and caches the result. All operator, analysis, and memory-synthesis calls of a run go to the same model, the one being evaluated. The resulting context is small and operator-specific: IMPROVE receives the node's deterministic record, a vertical trace of recent ancestors, and direct siblings ranked by the same utility used for parent selection, so it can contrast the chosen trajectory with nearby alternatives; CROSSOVER applies this to both parents and adds a complementarity cue; DEBUG retrieves prior attempts with the same error signature. Table 11 lists what each operator retrieves. Rich summaries are generated lazily only for retrieved nodes and are cached. The synthesis call receives the plans, code, execution output, score, status, and runtime of the node and its parent, is instructed to use only this evidence, and returns a method overview and a parent comparison stating what changed, whether score, status, and runtime improved, and what to reuse or avoid. Summaries are therefore produced only for nodes that enter a later operation.

F SUPPLEMENTARY EXPERIMENTS

F.1 EXPERIMENTAL SETUP

We evaluate on the official 22-task MLE-Bench Lite split released with MLE-Bench (Chan et al., 2024) unless otherwise stated. Each OPENMLE-EVO configuration is evaluated with three independent runs under a fixed per-task budget of 12 hours on a single RTX 4090 with 12 GB VRAM. For OPENMLE-EVO, this budget counts the cumulative sandbox execution time of the candidate programs evaluated during search, each execution is limited to 2 hours, and no new operator is started once the 12 hours are used. Because GPU performance in the shared evaluation environment fluctuates, the selected

program is executed three times on the test split and the best result is kept, so that the reported score is stable.

We report three aggregate metrics. *Valid Rate* is the mean number of the 22 tasks for which a run produces a valid submission, written $x/22$. *Medal Average* is the mean fraction of tasks receiving any Kaggle medal. *Human Rank* is the fraction of human leaderboard participants whose score is surpassed by the submitted solution, averaged across tasks and runs, so higher is better. Invalid final submissions receive no medal and zero Human Rank.

OpenMLE-35B is the primary model and carries the headline model, system, trajectory, and transfer analyses; OpenMLE-30B is a companion model used to test whether the post-training gain reproduces on a second backbone and scale. OpenMLE-Evo-Max extends the OpenMLE-Evo configuration in two ways: it distils reusable cross-task priors from public competition artifacts, with all MLE-Bench-related sources excluded before distillation, and it runs asynchronous parallel search on several RTX 4090s while keeping the total sandbox time per task at 12 GPU-hours, equivalent to one RTX 4090 for 12 hours.

F.2 Complete model and harness comparisons

All OpenMLE rows below aggregate three independent evaluation runs per task. Each run uses a per-task sandbox budget equivalent to 12 hours on one RTX 4090 with 12 GB VRAM. Values after $\pm$ are population standard deviations (divisor $n = 3$) across the three runs, not confidence intervals. The general-purpose Codex, Claude Code, and Gemini CLI reference points in Figure 16 were run by us under the same sandbox protocol, the same 22-task split, 12-hour per-task budget on one RTX 4090 at 12 GB VRAM, and scoring convention; because of cost, each was run once and carries no dispersion estimate.

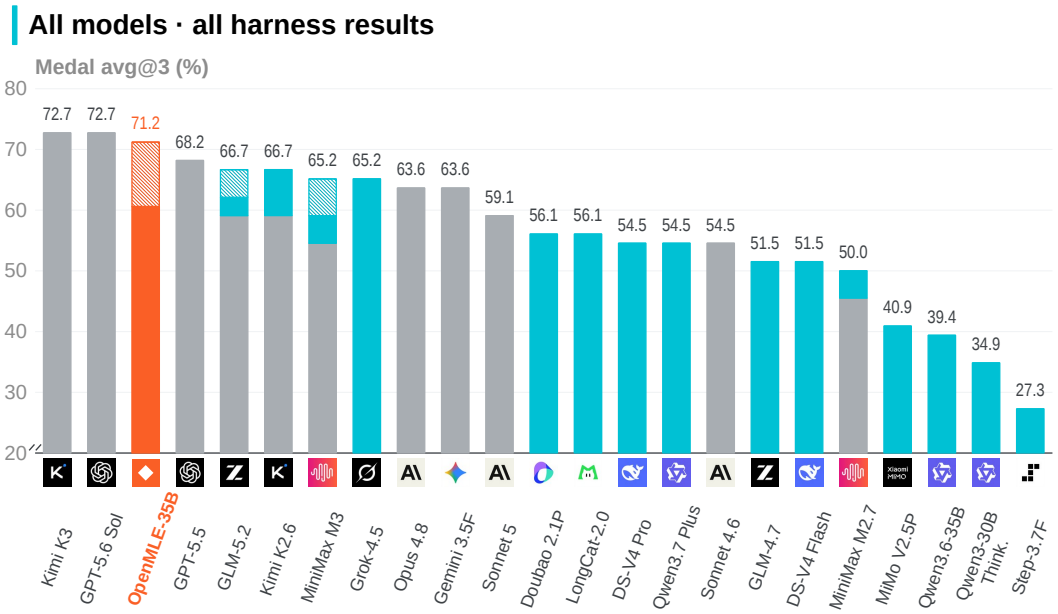

Figure 16: MLE-Bench Lite Medal Average for the model–harness pairs of Table 12 other than the two OpenMLE-30B rows. Gray marks a general-purpose coding-agent harness (Codex, Claude Code, or Gemini CLI), cyan OpenMLE-Evo, and hatching OpenMLE-Evo-Max; orange marks OpenMLE-35B. A model run with several harnesses has one bar divided at each harness's result, so each segment ends at the Medal Average of the harness it marks. All bars use the same per-task sandbox budget, equivalent to 12 hours on one RTX 4090 at 12 GB VRAM (OpenMLE-Evo-Max spreads it over parallel GPUs); OpenMLE-Evo and OpenMLE-Evo-Max bars average three runs and gray bars are single runs.

Table 12 is the full single-value landscape: Panel A holds the controlled comparisons, Panel B places OpenMLE-Evo results for further backbones alongside them, and Panel C lists general-purpose coding-agent references. Every general-purpose harness row, in Panel A or C, is a single run under the same sandbox protocol. For the three-run configurations with repeat-level records, Table 13 reports the mean and standard deviation over runs.

Table 12: MLE-Bench Lite results. Panel A presents the primary OpenMLE-35B comparison first, followed by the companion OpenMLE-30B replication and matched harness comparisons; Panels B–C provide broader system context. OpenMLE-Evo and OpenMLE-Evo-Max rows average three runs; rows with a general-purpose harness (Codex, Claude Code, or Gemini CLI) are single runs.

Model / system	Framework	Valid Rate ↑	Medal Average ↑	Human Rank ↑
A. Controlled comparisons				
OpenMLE-35B				
Qwen3.6-35B-A3B	OpenMLE-Evo	19.67/22	39.39%	0.5828
OpenMLE-35B	OpenMLE-Evo	21.67/22	60.61%	0.7647
OpenMLE-35B	**OpenMLE-Evo-Max**	22.00/22	**71.21%**	**0.8126**
OpenMLE-30B				
Qwen3-30B-A3B-Thinking-2507	OpenMLE-Evo	17.33/22	34.85%	0.5573
OpenMLE-30B	OpenMLE-Evo	21.67/22	53.03%	0.7055
OpenMLE-30B	**OpenMLE-Evo-Max**	22.00/22	66.67%	0.8053
Matched harness comparison · GLM-5.2				
GLM-5.2	Claude Code	21.00/22	59.09%	0.7948
GLM-5.2	OpenMLE-Evo	19.67/22	62.12%	0.7069
GLM-5.2	**OpenMLE-Evo-Max**	22.00/22	**66.67%**	**0.8164**
Matched harness comparison · MiniMax M3				
MiniMax M3	Codex	22.00/22	54.55%	0.7099
MiniMax M3	OpenMLE-Evo	22.00/22	59.09%	0.7994
MiniMax M3	**OpenMLE-Evo-Max**	22.00/22	**65.15%**	**0.8007**
Matched harness comparison · Kimi K2.6				
Kimi K2.6	Claude Code	18.00/22	59.09%	0.7062
Kimi K2.6	OpenMLE-Evo	21.67/22	**66.67%**	**0.7859**
Matched harness comparison · MiniMax M2.7				
MiniMax M2.7	Claude Code	18.00/22	45.45%	0.5547
MiniMax M2.7	OpenMLE-Evo	22.00/22	**50.00%**	**0.7039**
B. Broader OpenMLE-Evo system context				
Grok-4.5	OpenMLE-Evo	22.00/22	65.15%	0.8052
LongCat-2.0	OpenMLE-Evo	21.00/22	56.06%	0.7343
Doubao Seed 2.1 Pro	OpenMLE-Evo	20.33/22	56.06%	0.7170
Qwen3.7 Plus	OpenMLE-Evo	21.67/22	54.55%	0.7234
DeepSeek-V4-Pro	OpenMLE-Evo	21.67/22	54.55%	0.6849
DeepSeek-V4-Flash	OpenMLE-Evo	21.33/22	51.52%	0.6957
GLM-4.7	OpenMLE-Evo	21.33/22	51.52%	0.6521
MiMo-V2.5-Pro	OpenMLE-Evo	17.00/22	40.91%	0.5213
Step-3.7 Flash	OpenMLE-Evo	19.00/22	27.27%	0.4953
C. General-purpose coding-agent references				
GPT-5.6 Sol	Codex	22.00/22	72.73%	0.8891

Continued on next page

Table 12 — continued from previous page

Model / system	Framework	Valid Rate ↑	Medal Average ↑	Human Rank ↑
Kimi K3	Claude Code	22.00/22	72.73%	0.8574
GPT-5.5	Codex	21.00/22	68.18%	0.7833
Claude Opus 4.8	Claude Code	22.00/22	63.64%	0.8219
Gemini 3.5 Flash	Gemini CLI	20.00/22	63.64%	0.7499
Claude Sonnet 5	Claude Code	22.00/22	59.09%	0.7730
Claude Sonnet 4.6	Claude Code	22.00/22	54.55%	0.7670

Table 13: Repeated-evaluation statistics on MLE-Bench Lite: mean ± population standard deviation over three runs. Valid Rate is the mean number of tasks producing valid submissions out of 22.

Model	Harness	Valid Rate	Medal Average ↑	Human Rank ↑
Qwen3-30B-A3B	OpenMLE-Evo	17.33 ± 0.47	$34.85\% \pm 2.14\%$	0.5573 ± 0.0074
OpenMLE-30B	OpenMLE-Evo	21.67 ± 0.47	$53.03\% \pm 4.29\%$	0.7055 ± 0.0505
OpenMLE-30B	OpenMLE-Evo-Max	22.00 ± 0.00	$66.67\% \pm 5.67\%$	0.8053 ± 0.0236
Qwen3.6-35B-A3B	OpenMLE-Evo	19.67 ± 0.47	$39.39\% \pm 5.67\%$	0.5828 ± 0.0278
OpenMLE-35B	OpenMLE-Evo	21.67 ± 0.47	$60.61\% \pm 7.73\%$	0.7647 ± 0.0376
OpenMLE-35B	OpenMLE-Evo-Max	22.00 ± 0.00	$\mathbf{71.21\% \pm 8.57\%}$	0.8126 ± 0.0388
GLM-5.2	OpenMLE-Evo-Max	22.00 ± 0.00	$66.67\% \pm 8.57\%$	$\mathbf{0.8164 \pm 0.0233}$
MiniMax M3	OpenMLE-Evo	22.00 ± 0.00	$59.09\% \pm 0.00\%$	0.7994 ± 0.0225
MiniMax M3	OpenMLE-Evo-Max	22.00 ± 0.00	$65.15\% \pm 2.14\%$	0.8007 ± 0.0089
Kimi K2.6	OpenMLE-Evo	21.67 ± 0.47	$66.67\% \pm 5.67\%$	0.7859 ± 0.0285
Grok-4.5	OpenMLE-Evo	22.00 ± 0.00	$65.15\% \pm 2.14\%$	0.8052 ± 0.0170
LongCat-2.0	OpenMLE-Evo	21.00 ± 0.82	$56.06\% \pm 5.67\%$	0.7343 ± 0.0150
Doubao Seed 2.1 Pro	OpenMLE-Evo	20.33 ± 0.47	$56.06\% \pm 2.14\%$	0.7170 ± 0.0397
Qwen3.7 Plus	OpenMLE-Evo	21.67 ± 0.47	$54.55\% \pm 6.43\%$	0.7234 ± 0.0408
DeepSeek-V4-Pro	OpenMLE-Evo	21.67 ± 0.47	$54.55\% \pm 3.71\%$	0.6849 ± 0.0258
DeepSeek-V4-Flash	OpenMLE-Evo	21.33 ± 0.47	$51.52\% \pm 5.67\%$	0.6957 ± 0.0200
GLM-4.7	OpenMLE-Evo	21.33 ± 0.47	$51.52\% \pm 7.73\%$	0.6521 ± 0.0543
MiniMax M2.7	OpenMLE-Evo	22.00 ± 0.00	$50.00\% \pm 7.42\%$	0.7039 ± 0.0298
MiMo-V2.5-Pro	OpenMLE-Evo	17.00 ± 1.63	$40.91\% \pm 3.71\%$	0.5213 ± 0.0422
Step-3.7 Flash	OpenMLE-Evo	19.00 ± 0.00	$27.27\% \pm 6.43\%$	0.4953 ± 0.0385

Under a fixed controller, both trained checkpoints score above their base models on all three metrics. With the backbone fixed, OpenMLE-Evo raises Medal Average over the general-purpose harness in each of the four matched pairs of Panel A and over original AIRA-Evo for OpenMLE-35B (Table 2), and OpenMLE-Evo-Max raises it further for GLM-5.2 and MiniMax M3. The 71.21% result combines both interventions and is not reported as a model-only result.

Figure 17 plots each model's best result against total model size.

F.3 Search productivity

The matched search study runs both controllers with the same OpenMLE-35B checkpoint, seed, single search worker, and 12-hour task budget over 66 task-runs. OpenMLE-Evo evaluates 3,004 nodes versus 3,430 for the reference AIRA-Evo controller. Although it expands 12.4% fewer nodes, it records 246 strict new-best updates versus 229. Total model tokens fall from 129.3M to 75.3M and prompt tokens from 83.5M to 41.5M. The share of Improve calls that set a new best rises from 4.7% (44 of 931) to 9.4% (72 of 769).

Table 14: Matched search productivity.

Harness	Nodes	Total tokens	Prompt tokens	New-best/M tokens
Original AIRA-Evo	3,430	129.3M	83.5M	1.77
OpenMLE-Evo	3,004	75.3M	41.5M	3.27
Relative change	−12.4%	−41.7%	−50.3%	+84.3%

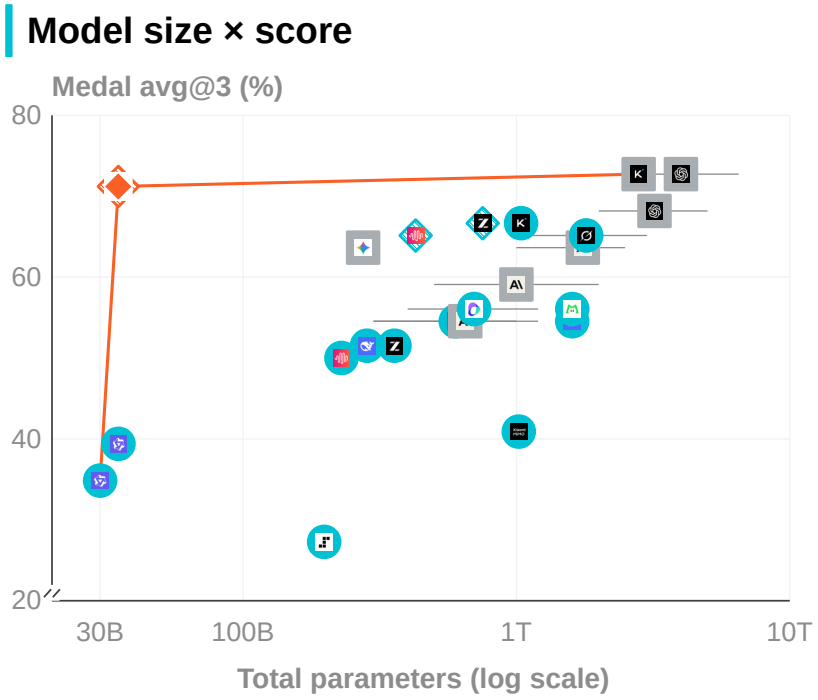

Figure 17: MLE-Bench Lite Medal Average against total model size, with one marker per model of Figure 16 at its best harness (orange: OPENMLE-35B with OPENMLE-EVO-MAX). Marker shape gives the harness: square for a general-purpose coding agent, circle for OPENMLE-EVO, diamond for OPENMLE-EVO-MAX. Sizes are total rather than active parameters, on a logarithmic axis, and are publicly reported counts where available. For GPT-5.5, GPT-5.6 Sol, Claude Opus 4.8, Claude Sonnet 5, Claude Sonnet 4.6, Gemini 3.5 Flash, Grok-4.5, Doubao Seed 2.1 Pro, and Qwen3.7 Plus, whose sizes are undisclosed, the marker sits at the midpoint of an estimated range and the horizontal whisker spans that range. The orange line joins the models that score above every smaller model.

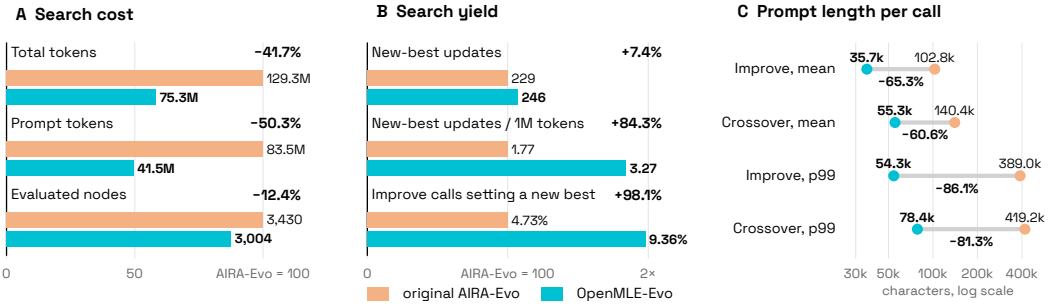

Figure 18: OPENMLE-EVO versus original AIRA-Evo over 66 matched task–runs with OPENMLE-35B: (A) search cost and (B) search yield, both normalized to original AIRA-Evo = 100; (C) serialized prompt length per call (log scale).

For Improve, mean serialized user context falls from 102.8K to 35.7K Unicode characters and the 99th percentile falls from 389.0K to 54.3K. For Crossover, the mean falls from 140.4K to 55.3K and the 99th percentile from 419.2K to 78.4K. The largest reduction occurs in the tail, consistent with replacing full-history concatenation by bounded operation-conditioned memory.

These are model-token and validation-productivity measures. They exclude CPU execution, accelerator idle time, data movement, retry cost, and embodied infrastructure cost, so they must not be interpreted as a complete energy or monetary accounting.

F.4 Long-horizon improvement and case studies

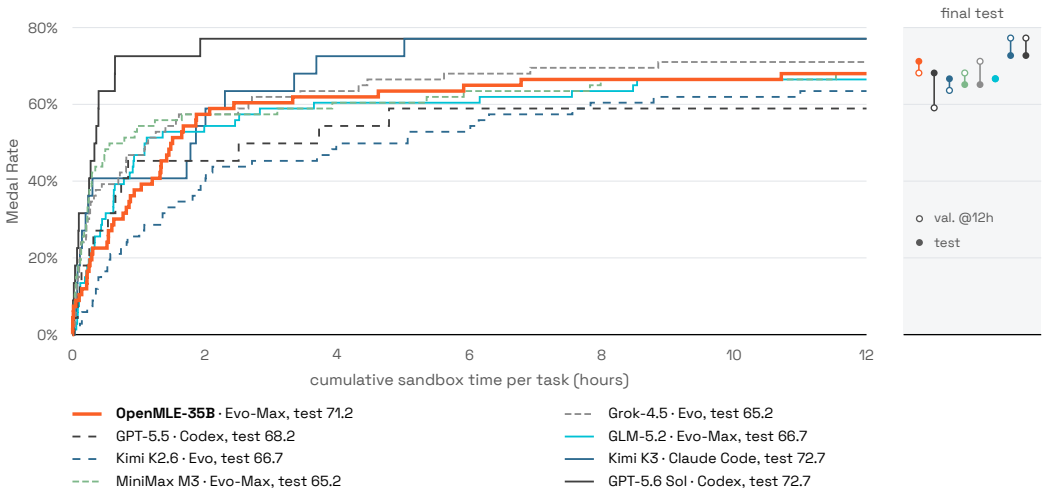

Figure 19: Validation Medal Rate against cumulative sandbox time per task on MLE-Bench Lite for eight model–harness pairs. Curves use the best-so-far validation score; for OpenMLE-Evo and OpenMLE-Evo-Max, sandbox time is the summed execution time of the task's candidate programs, the quantity the 12-hour budget counts. The right strip bridges validation Medal Rate at 12 hours (open) to final test Medal Rate after validation-based node selection (filled).

Aggregate trend. Figure 19 separates how fast a system finds medals from how long it keeps finding them. GPT-5.6 Sol with Codex is both the fastest and the strongest: it passes 60% within half an hour and reaches its final 77.27% validation Medal Rate before two hours. Kimi K3 with Claude Code reaches the same level more slowly, crossing 60% at 2.3 hours and making its last gain at about five. The three general-purpose harness runs all stop improving by about hour five; 16, 20, and 18 of the 22 GPT-5.6 Sol, Kimi K3, and GPT-5.5 tasks use the full 12 hours. Every OpenMLE-Evo and OpenMLE-Evo-Max run instead keeps improving past hour eight. OpenMLE-35B with OpenMLE-Evo-Max crosses 60% at 2.4 hours, close to Kimi K3, and climbs from 63.64% at five hours to 68.18% at twelve, with its last gain at 10.7 hours. Under OpenMLE-Evo and OpenMLE-Evo-Max, late budget thus still yields validation gains, narrowing but not closing the gap to the two strongest general-purpose runs.

Two case studies. Figures 20 and 21 follow single runs on two tasks in which OpenMLE-35B and three comparison models (its Qwen3.6 base, Step-3.7 Flash, and MiniMax-M2.7) share the OpenMLE-Evo protocol. On *leaf-classification*, two Debug calls first stabilize cross-validation and the image branch; an Improve then combines image embeddings with engineered shape and texture features, and two Crossover calls fuse the image and tabular branches, lifting validation Human Rank from 0.35 at step 2 to 0.54 at step 29. The largest single gain comes at step 45, where an Improve switches the image features to ConvNeXt-Tiny embeddings and raises it to 0.77. In the held-out panel (best of three runs), OpenMLE-35B reaches Human Rank 0.945 and a Bronze medal, while the best of the three comparison models, MiniMax-M2.7, ends at 0.630 on validation and 0.732

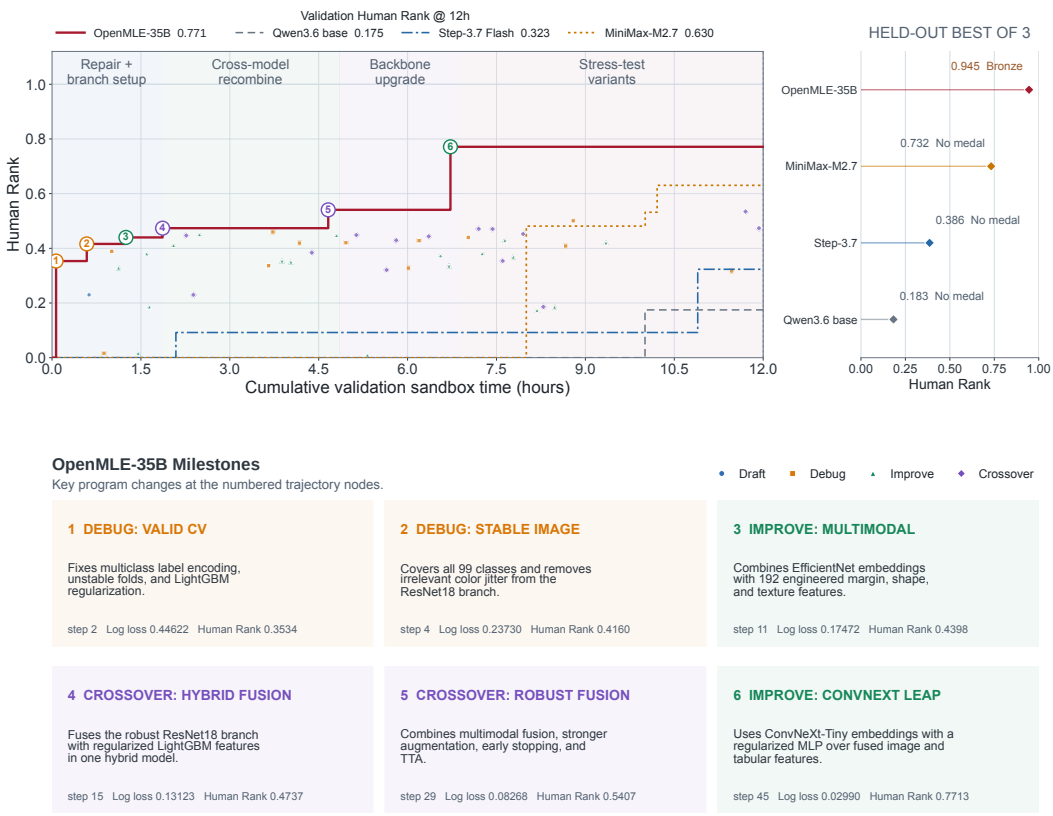

Figure 20: Search trajectories on *leaf-classification* under OPENMLE-EVO. Left: best-so-far validation Human Rank, with numbered OPENMLE-35B milestones described in the cards below. Right: held-out best-of-three test Human Rank.

on test without a medal. On *mlsp-2013-birds*, an early DEBUG only makes the submission valid (0.27 at step 5), and progress comes from IMPROVE and CROSSOVER calls that build audio branches: focal loss with test-time augmentation at step 119 lifts validation Human Rank from 0.44 to 0.64, and a final memory-guided CROSSOVER at step 150 reaches 0.73. The best of three held-out runs earns Silver (0.889), while all three comparison models stay at or below 0.30 on validation and earn no medal. In both runs most of the validation gain arrives after the first valid solution, through IMPROVE and CROSSOVER calls that extend and combine earlier branches.

F.5 NATUREBENCH TRANSFER DETAILS

To limit compute cost, we evaluate on a fixed subset of ten NatureBench tasks, which spans six scientific domains and multiple input structures. Hidden test labels and paper solutions are not available to the agent, web search is disabled, and each task receives a four-hour search budget. For a submitted metric value m and the source paper's reported value m_{SOTA}, NatureBench scores a task by the direction-normalized relative gap

$$g = d\,\frac{m - m_{\mathrm{SOTA}}}{|m_{\mathrm{SOTA}}|},$$

with $d = +1$ for metrics that are maximized and $d = -1$ for metrics that are minimized, averaged over instances when a task has several (Wang et al., 2026a). A task matches the paper result when $g \geq 0$ and surpasses it when $g > 0.1$. The three controlled rows of Table 3 share the NatureBench task containers, evaluator, and four-hour budget, and change one component at a time. The reference-agent rows are the runs reported by Wang et al. (2026a), restricted to these ten tasks; that protocol also gives four hours per task, one GPU for GPU tasks, and no web search. Their counts are taken from

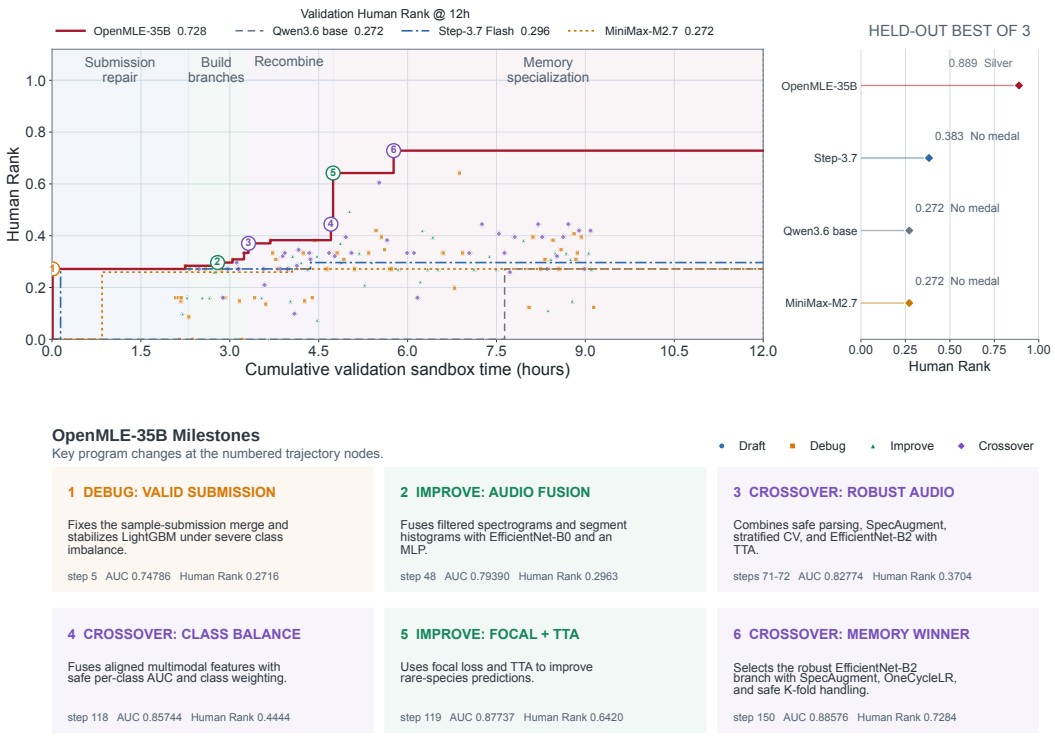

Figure 21: Search trajectories on *mlsp-2013-birds* under OpenMLE-Evo, laid out as in Figure 20.

that report, which precedes the benchmark leaderboard's re-assessment of all runs with an updated validity judge on 2026-07-30.

Table 15: Composition of the ten-task NatureBench subset.

Task	Domain	Input type
Spatial RNA Velocity Inference	Cellular omics	single-cell/spatial omics
Disease-Specific Variant Effect Prediction	Cellular omics	biological sequence
Metabolomic Profile from Microbial Composition	Cellular omics	feature matrix
Protein Variant Effect Prediction	Protein biology	biological sequence
Lasso Peptide Property Prediction	Protein biology	biological sequence
Anomalous Diffusion OOD Detection	Physical modeling	temporal signal
Zeolite–Molecule Binding Affinity	Physical modeling	molecular structure
Spatial Clustering of Localization Point Clouds	Biomedical modeling	image/volumetric
Molecular Property Prediction	Molecular design	molecular structure
Categorical Counterfactual Outcome Estimation	Relational reasoning	tabular features

Table 15 lists the ten tasks.

With the OpenMLE-Evo NatureBench adapter fixed, replacing the base model by OpenMLE-35B raises Match-SOTA from 5/10 to 7/10 and Surpass-SOTA from 2/10 to 3/10. With the base model fixed, replacing original AIRA-Evo by the OpenMLE-Evo adapter raises Match-SOTA from 2/10 to 5/10 and Surpass-SOTA from 1/10 to 2/10. On these ten tasks both swaps thus raise both counts, consistent with transfer of the post-trained model and of the structured search beyond competition-style MLE.

Table 16: Observable failures, current mitigations, and open verification gaps.

Failure	Observable symptom	Mitigation or open gap
Missing artifact	process exits without a valid submission	contract and artifact check
Runtime failure	exception, timeout, or memory exhaustion	isolated worker and typed terminal state
Metric gaming	score rises without intended task quality	hidden tests and semantic review; robust detection remains open
Visible-split overfit	local validation rises while holdout regresses	separated hidden evaluator
Evaluator leakage	candidate accesses protected metric or data	mounts and access control; adversarial tests needed
Search drift	repeated variants exploit a narrow branch	novelty/progress selection and visit cooling
Provenance loss	external code or data cannot be audited	lineage and artifacts; license analysis remains incomplete

G Release, verification, and reproducibility

G.1 Verification, safety, and reproducibility

Reproducible replay requires the task and evaluator versions, model checkpoint, operator, parent identifiers, random seed, resource limits, complete program, raw execution record, and artifact hashes. A resumed controller may retry an uncommitted attempt, but it must not silently reinterpret a committed node under a new evaluator. Rescoring creates a new evidence record and explicitly invalidates affected descendants when necessary.

The system improves external programs inside a fixed harness. It does not change its evaluator, scheduler policy, permissions, model weights, or release controls during deployment. Scaling should stop on resource exhaustion, repeated non-progress, verifier uncertainty, provenance failure, policy violation, or human intervention. Stronger deployments additionally need restricted networks, immutable images, signed task packages, dependency and license manifests, secret scanning, adversarial sandbox tests, rollback, and a human-controlled halt mechanism.

Table 17: Improvement levels and fixed external boundaries.

Level	Object changed	External boundary
Solution evolution	task program and artifacts	task contract and evaluator
Harnessed evolution	parent choice, operation, context	scheduler, budget, stop policy
Meta-evolution	transformation model learned from trajectories	training algorithm and acceptance rules
Recursive improvement	learner, harness, evaluator, controls	not implemented

Table 17 states which object each improvement level changes and which boundaries stay external.

G.2 Public release surfaces and reported evaluation settings

Table 18 records, for each system, which artifacts are public and the setting under which its MLE-Bench Lite score was reported. Comparisons run under our own sandbox protocol and budget are in Figure 7, Table 2, and Table 12.

Each Lite Medal entry is the figure reported by its source under its own setting; the systems differ in backbone model, wall-clock and accelerator budget, hardware, external-resource access, number of runs, and aggregation. A tick marks an artifact that was publicly accessible and independently checkable from the cited paper or repository at the time of audit; *NR* marks a field the source does not report, and we leave it unreported rather than imputing a value. Run settings give the per-task wall-clock limit and sandbox accelerator allocation only; remote model-serving compute, CPU, and RAM are omitted because most sources do not state them.

Table 18: Public release surfaces and reported evaluation settings for representative MLE agents and resources. Lite Medal is each system's own reported MLE-Bench Lite figure under its own setting.

Work	Data	Sandbox	Train	RL	Eval code	Weights	Lite Medal	Run setting	Best model
No trained MLE model released									
AIDE (Jiang et al., 2025)	×	✓	×	×	✓	×	35.91%	24h · 1×A10	o1-preview
AutoMLGen (Du et al., 2025)	×	✓	×	×	✓	×	62.12%	12h · 1×A800	DeepSeek-R1
ML-Master 2.0 (Liu et al., 2025a; Zhu et al., 2026)	×	✓	×	×	✓	×	75.76%	24h · 2×4090	DeepSeek-V3.2-Speciale
MLE-STAR-Pro-1.5 (Nam et al., 2025)	×	✓	×	×	✓	×	68.18%	24h · 2×A100	Gemini-2.5-Pro
MLZero (Fang et al., 2025)	×	✓	×	×	✓	×	36.36%†	24h · 8×A100	Claude-Sonnet-3.7
AIRA-dojo (Toledo et al., 2025)	×	✓	×	×	✓	×	55.00%	24h · 1×H200	o3
Famou-Agent 2.0 (Li et al., 2025a)	×	×	×	×	✓	×	80.30%	24h · 1×A800	Gemini-3-Pro-Preview
MLEvolve (Du et al., 2026)	×	✓	×	×	✓	×	80.30%	12h · 1×H200	Gemini-3-Pro-Preview
AIBuildAI (Zhang et al., 2026)	✓	×	×	×	✓	×	77.27%	24h · 1×A100	Claude-Opus-4.6
R&D-Agent (Yang et al., 2025b)	×	✓	×	×	✓	×	68.18%	12h · 1×V100	GPT-5
MLAgentBench (Huang et al., 2023)	✓	✓	×	×	✓	×	–	–	–
MLGym (Nathani et al., 2025)	✓	✓	×	×	✓	×	–	–	–
MLE-Dojo (Qiang et al., 2025a)	✓	✓	×	×	✓	×	–	–	–
MLE-Smith (Qiang et al., 2025b)	×	×	×	×	×	×	–	–	–
Post-trained MLE agents									
RL-MLE (Yang et al., 2025a)	×	×	×	✓	×	×	–	–	–
ML-Agent (Liu et al., 2025b)	×	×	✓‡	✓	×	×	–	–	–
MLE-RL (Li et al., 2025b)	×	×	×	✓	×	×	33.30%	12h · A10 (count NR)	MLE-RL-32B-S
AceGRPO (Cai et al., 2026)	×	×	×	✓	×	×	51.52%	12h · GPU NR	Ace-30B
OpenMLE§	✓	✓	✓	✓	✓	✓	71.21%	12 GPU-h · 4090 (12 GB)	OpenMLE-35B

Criteria. Data requires downloadable task or training data, or scripts that reconstruct it. *Sandbox* requires released code that can run model-generated programs or build their execution environment. *Train* requires model-training entry points or configurations. *Eval code* requires released evaluation code that scores a submission and recomputes the reported metric, or the benchmark's official per-run grading reports. *Weights* requires downloadable trained MLE-agent weights. A tick under *RL* means the work develops or applies an MLE-specific RL post-training method; it does not by itself imply that the implementation is public. †MLZero evaluated 21 of the 22 tasks and the displayed rate uses the standard 22-task denominator. ‡ML-Agent releases exploration-enriched SFT training code only. ✓ available at the time of audit; × not available. Prior systems are scored on what was publicly downloadable at the time of audit; the OpenMLE row applies the same criteria at submission. §The method code accompanies the supplementary material, and the complete OpenMLE-Gym corpus, the SFT trajectories, and both post-trained checkpoints are available at an anonymous repository (https://huggingface.co/datasets/nono314/openmle).

Read as a whole, the table shows that high-scoring systems typically release an inference framework and an evaluation entry point, resource projects release tasks and environments, and training-oriented papers describe RL designs without the surrounding stack. The combination of executable training data, sandbox, training code, RL method, evaluation, and weights remains rare, which is the gap this work targets.