Learning the ARTS of Search for Automated Discovery

Gurusha Juneja* **Arnav Kumar Jain†** **Deepak Nathani*** **William Yang Wang***

Xin Eric Wang*

Abstract

Scientific discovery can be formulated as an iterative search process over the space of hypotheses and experiments. Contemporary methods navigate this space using heuristics such as MCTS. These algorithms conflate the merit of a hypothesis with the quality of its experimental execution. A promising hypothesis with preliminary execution is therefore ranked below a modest hypothesis whose execution is refined. Moreover, prior methods prune the search logs as the search progresses because the accumulated history outgrows the context window. We propose **A**gentic **R**easoning for **T**ree **S**earch (ARTS), where we deploy a reasoning language model to navigate this space. The model inspects prior execution logs, diagnoses whether earlier failures arose from faulty implementations or bad hypotheses, and selects the hypothesis to build on next. To mitigate challenges with context length, ARTS uses test-time training to instill the knowledge of search tree in the model weights. Across 22 tasks from MLGym and MLEBench, we show that ARTS outperforms leading algorithms, with over 15.3% relative improvement in the normalized score. With test-time training we show that a Qwen3-4B agent can match performance with closed-source frontier models like Gemini-3 Pro and GPT o3-reasoning with upto $5\times$ lower inference cost. We further observe that on partially observable RL tasks, the test-time trained Qwen3-4B scientist surpasses ARTS with the o3 scientist by rediscovering the human-best recurrent-memory solution that heuristic methods prune away.

1 Introduction

Scientific discovery is an iterative *search process* in which multiple hypotheses are proposed, tested, and refined to arrive at novel insights. The advent of large reasoning models have led to their use as AI Scientist conducting this search [36, 31]. In the past, these systems have produced new constructions in extremal combinatorics [51], improved classical algorithms for matrix multiplication [39], planned and executed autonomous chemical syntheses [4], and generated novel hypotheses on antimicrobial resistance [15]. The success of these systems often rely on large sampling budgets and repeated execution of expensive experiments [51, 39]. This cost is significant in ML research [50, 38, 7, 53], where each hypothesis must be implemented as code, trained, debugged, and evaluated. Scaling automated research, therefore, requires efficient searching algorithms.

Existing search algorithms fall into three categories: linear search, tree-based search, and evolutionary search. Searching linearly [38, 20, 31] makes it difficult for the agent to revisit and improve upon an initial bad hypothesis [42]. Tree-based [50, 22, 9, 34, 8] and evolutionary [39, 44, 33, 52] algorithms alleviate this concern, but they search using score-based heuristics. This conflates hypothesis quality with it's execution in code. In such methods, a modest hypothesis that is well implemented is likely to be preferred over a promising hypothesis with preliminary execution. For instance, an

*University of California, Santa Barbara

†Université de Montréal and Mila- Quebec AI Institute

Preprint.

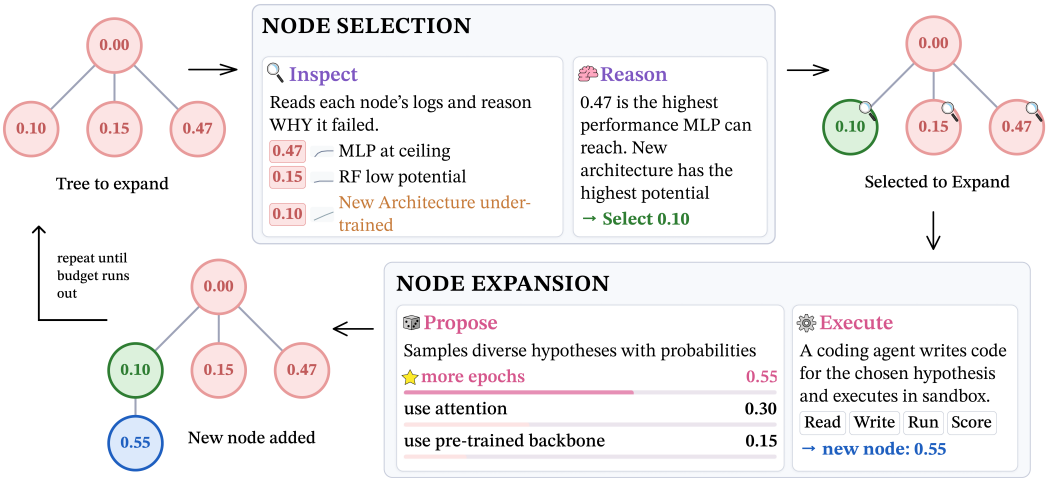


Figure 1: **A single search step of ARTS.** A node is one validated experiment with a hypothesis, code, logs, and score. Starting from the current search tree (Tree to expand), the `scientist` inspects the candidate nodes and reasons about why each scored as it did, then selects the most promising node rather than the one with the highest score. In this figure, it picks the node with score 0.10 since the low score comes from under training rather than a weak hypothesis (Node selection). Through verbalized sampling the `scientist` then proposes diverse hypotheses with probabilities and samples one, which the `executor` implements, runs, and scores (Node expansion). The resulting node (0.55) becomes the new best and is added to the tree. This loop repeats until the budget is exhausted. Red nodes are explored candidates, green marks the node selected for expansion, and blue marks the newly added node.

untuned Transformer scores below a LSTM on a sequence modeling task. Score-dependent heuristic search would prefer exploring LSTMs further even though the Transformer might surpass them upon tuned. Moreover, the reasoning models driving these searches suffer from diversity collapse [60] and the search procedures themselves are sensitive to the exploration-exploitation tradeoff [17], often committing to sub-optimal directions [56]. Lastly, as the search progresses, the accumulated history outgrows the model's context. Existing algorithms either summarize [50] or prune [31] this history and lose information, or retain it and reason less reliably over long contexts [35, 18, 61].

In this work, we propose Agentic Reasoning for Tree Search (ARTS)[3] that uses a Large Reasoning Model (LRM) to navigate the search space by exploring diverse hypotheses (Figure 1; Algorithm 1). When the search tree exceeds the context limits, ARTS* can learn from prior experiments instilling the knowledge into it's weights (Algorithm 2). In ARTS, an agentic LRM, which we call the *scientist* recieves the current search tree. It can inspect any existing node to learn about the prior explored hypothesis, execution errors, and performance. Using this information, the scientist *"reasons"* to select the node for expansion and generates the next hypothesis. This reasoning let's scientist judge if a poor score is the outcome of a fundamental limitation in the hypothesis or a correctable fault in its execution. To improve hypothesis diversity, ARTS uses an adapted version of Verbalized Sampling [60]. Our experiments show that, ARTS improves the mean normalized score by 15.3% compared to leading methods across tasks in MLGym and MLEBench.

ARTS* handles context-length overflow via test-time training [47], fine-tuning the scientist using the current search tree. Prior work has shown that such finetuning encodes knowledge into model weights [3, 6]. In our experiments, we find that the trained Qwen3-4B scientist reaches the performance of the o3 scientist at **5**× less inference cost. Remarkably, ARTS* with Qwen3-4B scientist outperforms all other methods, including ARTS with o3 scientist, on MetaMaze task with an optimized LSTM-based solution. Heuristic methods propose the recurrent-memory hypothesis but prune it after early instability, and ARTS with the o3 scientist returns to it repeatedly yet drifts as context grows. Test-time training instills this search experience in the Qwen-4B scientist's weights, allowing it to return to and refine the direction.

[3]Code and Trajectories at: https://auto-discovery.github.io/arts/.

The contributions of this paper are as follows:

- We propose **Agentic Reasoning for Tree Search** (ARTS), in which a Large Reasoning Model inspects prior node's hypotheses, executions, and outcomes and reasons over them to drive search.
- We propose a ARTS*, where the scientist is **Test-Time Trained** to handle context overflow that emerges as the search history grows. The scientist is finetuned using the past history which help it retain information in its weights akin to a memory mechanism.
- On MLGym and MLEBench, ARTS **outperforms leading methods** on 16 of 22 tasks and improves the mean normalized score **by 15.3%**. With ARTS*, a Qwen3-4B scientist matches frontier closed-source models on several tasks at a fifth of the inference cost.

2 Related Work

ML research as search. Automated ML research can be viewed as search over hypotheses, code, and execution traces. Benchmarks such as MLGym [38], MLEBench [7], RE-Bench [53], and DiscoveryBench [37] make this setting concrete by evaluating agents on tasks where each node is a real training run. Existing methods mainly differ in how they choose the next node to expand.

Linear search. Linear agents follow a single trajectory and condition on their previous actions and outcomes. This includes ReAct [57], Reflexion [45], AI Scientist v1 [36], and AutoResearch [31]. Reasoning models can revise ideas inside one generation [16, 42], but this is still bounded by the current context. In ML research, long traces of code, logs, and failed runs accumulate quickly, and long-context studies show that retrieval and reasoning degrade as context grows [35, 18]. Linear search is therefore simple, but it has weak global backtracking.

Tree and evolutionary search. Tree-search methods such as AIDE [22], AIRA [50], and AI Scientist v2 [36] maintain explicit trees and select nodes using score-based heuristics such as greedy value or UCT. This is problematic for ML research because a node score conflates hypothesis quality with implementation quality. A good idea can fail because of a missing import, a shape mismatch, or an unstable training loop. Evolutionary systems such as FunSearch [51], AlphaEvolve [39], OpenEvolve [44], and MLEvolve [13] search over populations of programs, but ML training code is tightly coupled and expensive to evaluate. Mutation and crossover can be effective for compact programs, but they are a brittle primitive for full ML pipelines.

Reasoning-guided search and test-time training. ARTS replaces heuristic node selection with an agentic scientist that inspects prior execution traces and chooses what to expand next. Verbalized sampling [60] is used to keep the proposal distribution diverse instead of greedily following one direction. We also build on test-time training [47, 2, 59, 46], but train a tree-structured scientist rather than a one-turn proposal policy. The extended discussion is in Appendix H.

3 Research as a Search Problem

Research tasks. For a Machine Learning research task, we provide the agent with an initial workspace $\mathcal{W}_0$, an evaluation metric m, a baseline score s_0, and a wall-clock budget T. We denote the task by $\mathcal{P} = (\mathcal{W}_0, m, s_0, T)$. The workspace contains task instructions, data access, and an evaluation harness. The agent interacts with the workspace by editing code inside a sandbox and querying the evaluator. The evaluator returns execution logs and a score when the run succeeds. When evaluation fails, it returns error messages and partial logs.

Experiments and search histories. Each evaluated experiment creates a node v in the search tree $\mathcal{G}_t = (V_t, E_t)$. The node stores a parent p_v, a hypothesis h_v, a code state c_v, execution logs ℓ_v, a score s_v, and a depth d_v. The parent's code state c_{p_v} serves as the starting point on which the agent builds the new code implementation c_v for h_t. The search history $\mathcal{G}_t$ can be a chain, a tree, or a graph.

At step t, the research agent chooses an action $a_t = (p_t, h_t, c_t)$, where $p_t \in V_t$ is the parent node to expand, h_t is the hypothesis to test, and c_t is the code implementation for h_t starting from the parent code state c_{p_t}. The evaluator runs c_t inside the sandbox and returns logs ℓ_t and score s_t. We call the component that proposes h_t the `scientist`. We call the component that writes c_t the `executor`. Prior methods [50, 13, 31] use the same agent as both `scientist` and `executor`, and pick the parent either as the last node (Linear) or by a score heuristic like UCB (MCTS, evolutionary).

4 Agentic Reasoning for Tree Search

We present ARTS (**A**gentic **R**easoning for **T**ree **S**earch) that leverages two design choices. First, we use different models for the scientist and the executor, since choosing a research direction and writing correct code require different abilities. Second, instead of selecting the parent node p_t with a fixed heuristic, we let the scientist choose both the parent p_t and the hypothesis h_t. The scientist is shown a compact view of the tree, including each node's identifier, score, and hypothesis. It also has access to persistent memory, and can inspect prior nodes to read their code and execution logs. To mitigate diversity collapse, we use verbalized sampling [60]. The scientist outputs a proposal distribution over K candidate actions, and the system samples from it rather than always taking the top-ranked action. The executor then implements the sampled hypothesis from the selected parent's code state.

4.1 The `Scientist`

The `scientist` selects which node to expand and what hypothesis to try, both requiring reasoning over multiple past experiments. For instance, concluding that the bottleneck is the loss function and not the model capacity after finding that (a) increasing model capacity failed to improve validation accuracy, and (b) training curves show that training accuracy also failed to improve, requires strong reasoning capabilities. We therefore use o3 as the `scientist` owing to it's strong multi-step reasoning abilities [40, 10, 14, 43]. Below, we describe the components used in Algorithm 1.

Node Inspection. A node's score alone does not give enough information to judge why it succeeded or failed. A node can have low score due to a number of reasons, including wrong hypothesis, wrong code execution, correct hypothesis but insufficient training time, etc. Without knowing the exact reason for failure or success, the next hypothesis is just a guess. The INSPECT tool returns the code, training curves, and program output of any prior node in the search tree. This lets the `scientist` identify the actual cause, for example, observing that a sparse-reward run never received a positive reward, and proposing reward shaping rather than an bigger model. After INSPECT, the `scientist` reasons to choose the appropriate next node to expand and the hypothesis to test.

Baseline Audit. Many tasks have structural biases that can be exploited to arrive to clever solutions. For example, on the Vesuvius ink-detection task only a handful of samples have labels, but the unlabeled data is huge; self-supervised pretraining can be exploited in such cases. To ensure that the `scientist` is aware of these biases while reasoning about hypotheses, we reserve the first R calls to the `scientist` as AUDITS. During this phase no node may be expanded and no code may be changed; the `scientist` may only read the data, baseline code, and evaluation script. The audit must produce statements about the task, such as the structure of the metric (e.g. ordinal, sparse, class-imbalanced) or the largest information-loss step in the baseline.

Hypothesis sampling. Exploring diverse solutions is essential in research. If we ask the scientist to provide the next hypothesis to explore, it returns most probable continuation, which usually are small tweaks over the learning rate. To ensure that the explored solution space is diverse, we use verbalised sampling [60]. Here, the scientist enumerates K candidate hypothesis h_t^k each associated with a probability π_t^k assigned by the scientist to the hypothesis. One candidate is drawn by an external sampler, since when the scientist samples from its own list it tends to pick the first candidate. At deeper nodes we ask the scientist to sample from the tail of the hypothesis distribution.

Memory. Over a long search the tree accumulates hundreds of nodes, and re-reading them all on every step would exhaust the context window. We complement the scientist with an editable memory module. After each expansion it writes one short insight (the score achieved, the failure mode, etc.) to this memory, appended only if not already present. The scientist can read this memory at all times, including when selecting a node and generating the next hypothesis.

4.2 The `Executor`

The `executor` is a coding agent that takes the `scientist's` hypothesis h_t and the selected parent's validated code c_{p_t}, and writes a new code state c_t. We use Gemini 3 Flash as the `executor` since it is fast, inexpensive, and strong at code generation [12]. The executor has access to three tools: READ_FILE (to read existing code), WRITE_FILE (to write new code), and VALIDATE (to execute the code in the sandbox and get the validation score). It can edit and validate the code until either a valid score is returned or the maximum action budget is reached. It is prompted to implement h_t as given

Algorithm 1: `ARTS` **search.**

Input: Task $\mathcal{P}$, time budget T, `scientist`, `executor`
Initialize: Tree $\mathcal{G}$ = (code c_{base}, score s_{base})
AUDIT: data, code, and metric. // No node expanded.
while time < T **do**
  // scientist calls inspects tree to get logs
  $\ell_t \leftarrow \textsc{Inspect}(\mathcal{G}_t, \mathcal{P})$
  //parent selection and hypothesis sampling (h_t^k, π_t^k)
  $p_t, \{(h_t^k, \pi_t^k)\}_{k=1}^K \leftarrow \texttt{scientist}(\ell_t, \text{memory})$
  // external sampler draws one hypothesis
  $h'_t \leftarrow \textsc{Sample}(\{(h_t^k, \pi_t^k)\}_{k=1}^K)$
  // implement and validate
  $c_t, s_t \leftarrow \textsc{Execute}(p_t, h'_t, c_{p_t})$
  $\mathcal{G}_{t+1} \leftarrow \mathcal{G}_t \cup (p_t, h'_t, c_t, s_t)$ // add new node to tree
end while
return best validated node

Algorithm 2: `ARTS`* **with test time training.**

Input: Task $\mathcal{P}$, group size B, `scientist` π_θ, `executor`
Initialize: $\mathcal{G}$ = (c_{base}, s_{base}) LoRA adapters.
for each GRPO step **do**
  $\ell_t \leftarrow \textsc{Inspect}(\mathcal{G}_t, \mathcal{P})$
  // parent selection and hypothesis sampling
  $\{(p_i, \{(h_i^k, \pi_i^k)\}_{k=1}^K)\}_{i=1}^B \sim \pi_\theta(\mathcal{G}_t)$
  $h'_i \leftarrow \textsc{Sample}(\{(h_i^k, \pi_i^k)\}_{k=1}^K)$ for each i
  $c_i, s_i \leftarrow \texttt{executor}(p_i, h_i, c_{p_t})$ for each i
  // percentile reward
  $r_i \leftarrow R(s_i, \text{tree})$
  Update LoRA with GRPO on $\{(p_i, h_i, r_i)\}_{i=1}^B$
  // Add rollouts to the tree.
  $\mathcal{G}_{t+1} \leftarrow \mathcal{G}_t \cup \{(p_i, h_i, c_i, s_i)\}_{i=1}^B$
end for
return trained scientist $\pi_{\theta'}$, best validated node

by the scientist, we find that without this rule it substitutes h_t with alternatives of its own. Once the executor produces a successful VALIDATE, the resulting node is appended to the search tree with its score s_t (or null if no valid submission was produced within the action budget).

4.3 `ARTS`*: Test-Time Training the `Scientist`

The search tree $\mathcal{G}_t$, especially on hard problems like drug discovery, can grow exponentially with depth and exceed `scientist`'s context limits. A tree with depth 8 and branching factor 3 has over 6000 nodes. Prior algorithms prune earlier nodes in the search tree. Further, it is also known that long contexts degrade reasoning quality of models [35, 61]. To solve this problem we propose test time train (TTT) the `scientist` on the search tree it has explored so far. We fine-tune the model with LoRA [19] at regular intervals to distill the information from experiments in the model weights. Algorithm 2 summarizes the `ARTS`* procedure.

Rollout. Prior work [59] has explored TTT for hypothesis generation, but the action is only the generated hypothesis and node selection is heuristic. In our setting we make two decisions per step, which node p_t in the search tree to expand from, and what hypothesis h_t to test. Given the state $\mathcal{G}_t$ (see §3), the `scientist` agent outputs the action $a_t = (p_t, h_t)$. We sample N such actions from the current state $\mathcal{G}_t$ and compute a reward for each, as explained below. The N rewards form a GRPO group and is used to update the adapter weights. Since each action creates a new node, they are appended to the tree before the next batch of rollouts is sampled. This teaches the model both which parent to pick and what hypothesis to explore for the largest gains in the validation score.

Reward. The reward is based on the score s_t that the action $a_t = (p_t, h_t)$ receives after the code implementation c_t by the `executor` is executed. We find that single-step percentile-based rewards work best:

$$r_t = \begin{cases} -0.5 & s_t = \text{null}, \\ -0.2 & s_t < s_{\text{base}}, \\ 0 & s_{\text{base}} \le s_t < s_{70}(\mathcal{G}_t), \\ 1 & s_t \ge s_{70}(\mathcal{G}_t), \end{cases} \tag{1}$$

where $s_{70}(\mathcal{G}_t)$ is the 70th percentile of all scores observed up to timestep t. This signal is denser than rewarding only the single best node, since the top 30% of nodes get a positive reward than just the maximum one. The percentile based adaptive threshold rises as the policy improves, preventing reward saturation. Learning with one-step rewards [21] works better than rewarding longer multi-step trajectories because it can provide clearer credit assignment to each action.

Diversity. Training on score based rewards can lead to diversity collapse, biasing the policy towards solutions that gave higher results early. But, we find that this is not the case even after training, since verbalized sampling ensures that the samples hypotheses in one group are diverse (See §6.4).

Table 1: Performance comparison of ARTS with prior works and human best scores. We test on 22 tasks across MLGym and MLEBench (3 easy, 3 medium, and 4 hard tasks). The table reports the mean and standard error of the best scores achieved in every run. Every experiment is ran 3 times. Human Best is Kaggle top-1 for MLEBench and published SOTA for MLGym. Highlighted cells show the best performing methods.

		Prior Works			*Ours*	
Task	**Baseline**	**Linear**	**AIRA**	**MLEvolve**	ARTS	**Human Best**
MLGym						
Titanic (acc ↑)	0.766	$0.951_{\pm.004}$	$0.944_{\pm.001}$	$0.946_{\pm.002}$	$\mathbf{0.984}_{\pm.004}$	0.830
CIFAR-10 (acc ↑)	0.497	$\mathbf{0.956}_{\pm.002}$	$\mathbf{0.964}_{\pm.002}$	$\mathbf{0.959}_{\pm.003}$	$\mathbf{0.971}_{\pm.017}$	0.994
Fashion MNIST (acc ↑)	0.848	$0.946_{\pm.006}$	$0.947_{\pm.000}$	$0.948_{\pm.001}$	$\mathbf{0.958}_{\pm.001}$	0.968
House Price (R^2 ↑)	0.880	$0.940_{\pm.000}$	$\mathbf{0.944}_{\pm.001}$	$\mathbf{0.943}_{\pm.001}$	$0.939_{\pm.003}$	0.990
MNLI (acc ↑)	52.51	$\mathbf{84.42}_{\pm0.05}$	$83.77_{\pm0.05}$	$\mathbf{84.26}_{\pm0.08}$	$\mathbf{84.71}_{\pm0.49}$	92.50
Lang. Modeling (loss ↓)	4.673	$3.986_{\pm.169}$	$4.673_{\pm.000}$	$4.015_{\pm.146}$	$\mathbf{3.827}_{\pm.088}$	3.500
Battle of Sexes (payoff ↑)	1.023	$1.448_{\pm.001}$	$1.442_{\pm.001}$	$1.446_{\pm.000}$	$\mathbf{2.000}_{\pm.000}$	1.667
Prisoner's Dilem. (payoff ↑)	2.372	$2.453_{\pm.102}$	$2.501_{\pm.129}$	$\mathbf{2.635}_{\pm.012}$	$\mathbf{2.857}_{\pm.225}$	3.000
Blotto (score ↑)	−0.248	$-0.076_{\pm.327}$	$0.247_{\pm.003}$	$\mathbf{0.250}_{\pm.001}$	$\mathbf{0.249}_{\pm.001}$	0.500
MountainCar (reward ↑)	33.79	$45.44_{\pm6.27}$	$80.82_{\pm11.75}$	$84.84_{\pm6.38}$	$\mathbf{95.73}_{\pm.69}$	99.00
Breakout (reward ↑)	48.82	$64.03_{\pm10.75}$	$57.94_{\pm3.37}$	$\mathbf{83.28}_{\pm3.84}$	$78.00_{\pm2.23}$	100.00
Meta Maze (reward ↑)	15.73	$46.80_{\pm2.61}$	$36.42_{\pm6.47}$	$45.35_{\pm3.27}$	$\mathbf{51.20}_{\pm1.03}$	52.50
MLEBench						
Spaceship Titanic (acc ↑)	0.000	$0.825_{\pm.006}$	0.831	$0.834_{\pm.001}$	$\mathbf{0.836}_{\pm.001}$	0.828
Nomad 2018 (MCW-RMSLE ↓)	1.000	$\mathbf{0.063}_{\pm.001}$	$\mathbf{0.062}_{\pm.001}$	$\mathbf{0.063}_{\pm.000}$	$0.064_{\pm.001}$	0.051
Jigsaw Toxic (CW-AUC ↑)	0.500	$\mathbf{0.980}_{\pm.000}$	$\mathbf{0.980}_{\pm.000}$	$\mathbf{0.980}_{\pm.000}$	$\mathbf{0.980}_{\pm.000}$	0.989
APTOS 2019 (QWK ↑)	0.000	$\mathbf{0.926}_{\pm.001}$	$0.922_{\pm.006}$	$0.914_{\pm.004}$	$\mathbf{0.930}_{\pm.004}$	0.936
Plant Pathology (MCW-AUC ↑)	0.500	$0.994_{\pm.001}$	$0.997_{\pm.000}$	$\mathbf{0.998}_{\pm.000}$	$0.995_{\pm.003}$	0.984
Histopath. Cancer (AUC ↑)	0.500	$0.990_{\pm.001}$	$\mathbf{0.995}_{\pm.000}$	$0.994_{\pm.000}$	$\mathbf{0.995}_{\pm.000}$	1.000
Vesuvius Ink Det. ($F_{0.5}$ ↑)	0.000	$0.479_{\pm.067}$	$0.309_{\pm.112}$	$\mathbf{0.549}_{\pm.011}$	$\mathbf{0.551}_{\pm.021}$	0.831
Kuzushiji Recog. (F1 ↑)	0.000	$0.894_{\pm.040}$	$0.872_{\pm.026}$	$\mathbf{0.921}_{\pm.009}$	$0.843_{\pm.034}$	0.950
HMS Brain Activity (KL-div ↓)	1.462	$0.543_{\pm.055}$	$0.550_{\pm.020}$	$0.583_{\pm.013}$	$\mathbf{0.499}_{\pm.008}$	0.272
RSNA Brain Tumor (AUC ↑)	0.500	$0.638_{\pm.005}$	$0.649_{\pm.014}$	$\mathbf{0.656}_{\pm.011}$	$\mathbf{0.673}_{\pm.021}$	0.621

5 Experiments

Tasks. We evaluate on 22 tasks from MLGym [38] and MLEBench [7] [4]. MLGym contains regression, Language Modeling, Vision, RL and Game Theory tasks. MLEBench consists of tasks taken from kaggle competitions, and we evaluate on 4 hard, 3 medium and 3 easy tasks. These benchmarks have well-defined metrics and realistic ML tasks. Each task provides the initial workspace $\mathcal{W}_0$ (see § 3) and a held-out test set. We report human-best scores using Kaggle top-1 for MLEBench and achieve SOTA on MLGym. See Appendix J for the sources used.

Baselines. We compare ARTS with three prior search methods from § 2. Linear Search uses the AutoResearch prompt [31], Tree Search uses AIRA (MCTS) [50], and Evolutionary Search uses MLEvolve [13], and methods receive same executor, container, action set, and runtime budgets.

Models. We use OpenAI o3 [40] as the scientist and Gemini 3 Flash [12] as the executor. The prompts for both the scientist and executor are in Appendix K. We also experiment with Qwen3-4B-Instruct [55] as the scientist, and update it with test-time training to handle long context.

Evaluation. All methods run inside Apptainer containers with the same action set, executor sandbox, and 8-hour wall-clock budget. Akin to prior approaches that evaluate with fixed-budget runs [31, 50, 13, 7], we use time rather than node count in our experiments. We limit the runtime budget to 8 hours. Since agents sometimes can modify evaluation scripts, we restore them separately for validation (see Appendix I). During evaluation, we evaluate over 3 independent runs and report the mean and standard error of the best validation score reached in a run. Each method in Table 1 is given one

[4] Not the entire suite of tasks because of compute and API constraints

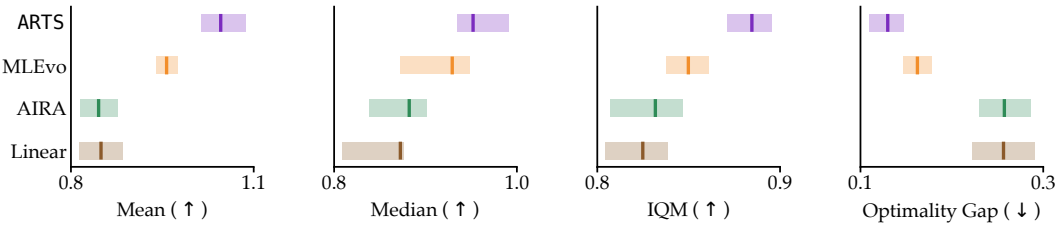


Figure 2: **Aggregate normalized performance**. We report the reliable metrics [1] and observe that ARTS significantly outperforms baselines at Mean and IQM while achieving lower optimality gap.

40 GB A100 GPU to train any downstream model during the search process. We provide details about implementation, prompts, finetuning with RL, and compute in Appendix I. To compute the normalized score for rliable [1] we use the baseline and the human-best score.

6 Results

Through our experiments, we answer four questions. First, does ARTS search more effectively than prior methods? It reaches the best score on 16 of 22 tasks under the same budget (§6.1). Second, what makes ARTS better? We find that these gains come from three properties of its search: failure attribution, the diversity of the hypotheses it explores, and their quality §6.2 Third, does test-time training help? It improves a Qwen3-4B scientist to match the o3 scientist on half the tasks at a fraction of the inference cost §6.3 and finally, ablations isolate the contribution of each components in §6.4.

6.1 How does ARTS compare to prior methods?

Table 1 reports per-task performance. ARTS has largest gains on the harder tasks from MLEBench and MLGym where each hypothesis is an expensive training run. There is no obvious text-book obvious recipe to solve these tasks and committing to a poor choice early leads to sub-optimal utilization of the search budget. Solving them requires reasoning over what to try next, and which hypothesis has the most potential. ARTS performs best on most of the hardest tasks, including HMS Brain Activity and RSNA Brain Tumor, where it even beats the human best. Problems like House Price and Jigsaw Toxic are close to saturated, every method is already near the ceiling.

Figure 2 aggregates the per-task scores into the reliable metrics of Agarwal et al. [1]. ARTS attains the highest inter-quartile mean (IQM) among all the contemporary methods, this shows that the typical performance of ARTS is better than prior methods. Because IQM trims the top and bottom 25% of runs, this score cannot be inflated by a few lucky outliers. The optimality gap measures how far a method stays below the human best score reached on each task. We find that ARTS closes most of the distance to the per-task optimum. These results show that improvement is reliable and not tail driven.

Figure 3 plots the best score against wall-clock time under the same 8-hour budget. We observe that ARTS typically trails for the first one or two hours, catches up mid-run, and finishes above the strongest contemporary method. This pattern of a slow start followed by a higher ceiling shows that ARTS spends its early budget exploring broadly instead of committing onto the first locally good hypothesis, as opposed to other methods,. leading to better utilizaton of the inference budget.

6.2 Why does ARTS outperform prior methods?

We trace ARTS's gains to three factors: *failure attribution*, the *diversity* of hypotheses it explores, and the *quality* of each hypothesis, as described below.

> **Finding 1.** ARTS does not **prematurely abandon a promising hypothesis.**

Prior tree-search methods do not separate why a node scored low from the fact that it did. A logical coding bug, bad hyperparameters, an under-trained model, or a genuinely poor hypothesis all surface as the same low score. The next draft pivots to a different hypothesis even when the original one was sound. ARTS is designed to explicitly enforce this distinction, the `scientist` inspects nodes's code and reasons whether the idea was wrong or the implementation was. If the implementation was

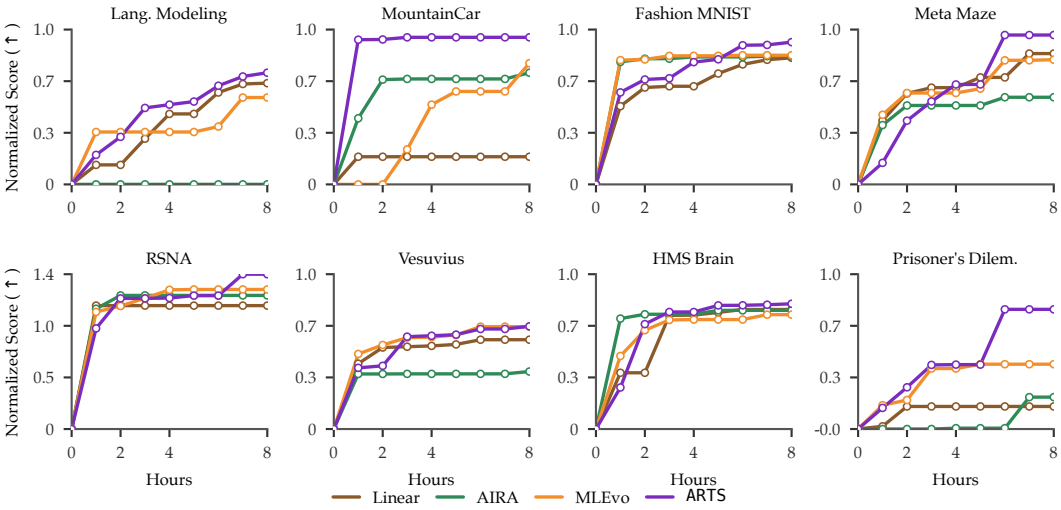


Figure 3: Hourly progression of normalized score on randomly selected tasks under the same 8-hour budget. Lines show mean(of the best) performance across 3 runs at each hour. All baselines use same executor agent (Gemini-3-Flash) as ARTS.

wrong, the next experiment stays on the same axis as the chosen node's hypothesis with corrected instructions for the executor; if the idea was wrong, the search is free to pivot to a different axis. This ensures that a promising idea is no longer discarded prematurely.

For instance, on HMS Brain Activity task (Fig. 5; lower KL is better; baseline = 1.462). ARTS drafts a pretrained ResNet-50 over 3-channel log-mel spectrograms at depth 1 and reaches **KL 0.557**. The next two children, a backbone swap to EfficientNet-B0 (0.702) and an added SpecAugment (0.565), both regress. A score-only selection algorithm, AIRA for instance, reads these regressions as evidence that the spectrogram-CNN family is exhausted, and its next two drafts pivot to hand-crafted features, and the children of those pivots diverge catastrophically to KL 4.08 and 9.43 while the original spectrogram-CNN family is never refined. ARTS instead re-reads the two regressed nodes' training logs, observes that the validation KL is still decreasing at the last epoch in both, and classifies them as implementation-wrong (under-trained under heavy augmentation), retaining the ResNet-50 + spectrogram family. The next few expansions improves the KL further to 0.516; and continued refinement on the same family, that AIRA abandoned early, ultimately reaches KL 0.467 making ARTS the best method on this task.

A few more illustrations of the same pattern are APTOS (a constant-prediction collapse at lr$= 10^{-3}$ misattributed by score-only methods to ConvNeXt itself; ARTS attributes it to the LR schedule and recovers ConvNeXt to QWK 0.92) and Vesuvius (an 11-channel DeepLabV3 misscored at $F_{0.5} = 0.165$ because the validation split was set to zero in the training script; ARTS reads the log line and repairs the split rather than discarding the wide-input variant) with trees and node-level diagnoses are in Appendix G.

> **Finding 2.** ARTS distributes its expansions across **diverse hypothesis** families compared to score-based methods that concentrate around a small set of hypotheses.

Exploring a diverse set of hypotheses is important in open-ended search problems, like ML research. This is because many different approaches can succeed, a new architecture, loss, data representation, or optimizer. The search must explore widely, refining the approach that scored best first is not guaranteed to give global optimal solution. The prior score-only methods concentrate the search to a small set of leading hypotheses and only explore its close variants. In contrast, we find that ARTS proposes a higher range of diverse hypotheses, widening the explored set of hypothesis substantially more compared to score-only methods.

Table 2: ARTS*: **test-time training a Qwen3-4B scientist on MLGym tasks.** The Qwen3-4B scientist is test time trained on its own search history; we report mean ± standard error of the best score over 3 runs, with the best method highlighted. We find that the small 4B scientist match or surpass the o3 scientist on about half the tasks at a fraction of the inference cost.

		Prior Works			*Ours*			
Task	Metric	Linear	AIRA	MLEvolve	ARTS (o3)	ARTS (Qwen 4B)	ARTS* (4B)	Human Best
Titanic	acc ↑	$0.951_{\pm.004}$	$0.944_{\pm.001}$	$0.946_{\pm.002}$	$0.984_{\pm.004}$	$0.949_{\pm.009}$	**0.998**$_{\pm.002}$	0.830
CIFAR-10	acc ↑	$0.956_{\pm.002}$	$0.964_{\pm.002}$	$0.959_{\pm.003}$	$0.971_{\pm.017}$	$0.957_{\pm.002}$	**0.982**$_{\pm.006}$	0.994
Fashion MNIST	acc ↑	$0.946_{\pm.006}$	$0.947_{\pm.000}$	$0.950_{\pm.001}$	**0.958**$_{\pm.001}$	$0.946_{\pm.002}$	$0.948_{\pm.007}$	0.968
House Price	R^2 ↑	$0.940_{\pm.000}$	**0.944**$_{\pm.001}$	$0.943_{\pm.001}$	$0.939_{\pm.003}$	$0.929_{\pm.001}$	$0.940_{\pm.001}$	0.990
MNLI	acc ↑	**84.42**$_{\pm0.05}$	$83.77_{\pm0.05}$	**84.26**$_{\pm0.08}$	**84.71**$_{\pm0.49}$	$83.33_{\pm.08}$	$84.01_{\pm1.81}$	92.50
Lang. Modeling	loss ↓	$3.986_{\pm.169}$	$4.673_{\pm.000}$	$4.015_{\pm.146}$	$3.827_{\pm.088}$	$4.34_{\pm.48}$	**3.518**$_{\pm.276}$	3.500
Battle of Sexes	payoff ↑	$1.448_{\pm.001}$	$1.442_{\pm.001}$	$1.446_{\pm.000}$	**2.000**$_{\pm.000}$	$1.441_{\pm.000}$	$1.442_{\pm.000}$	1.667
Prisoner's Dilemma	payoff ↑	$2.453_{\pm.102}$	$2.501_{\pm.129}$	**2.635**$_{\pm.012}$	**2.857**$_{\pm.225}$	**2.641**$_{\pm.001}$	**2.633**$_{\pm.318}$	3.000
Blotto	score ↑	$-0.076_{\pm.327}$	**0.247**$_{\pm.003}$	**0.250**$_{\pm.001}$	**0.249**$_{\pm.001}$	**0.251**$_{\pm.002}$	**0.344**$_{\pm.33}$	0.500
MountainCar	reward ↑	$45.44_{\pm6.27}$	$80.82_{\pm11.75}$	$84.84_{\pm6.38}$	**95.73**$_{\pm.69}$	$56.65_{\pm1.69}$	$91.94_{\pm2.83}$	99.00
Breakout	reward ↑	$64.03_{\pm10.75}$	$57.94_{\pm3.37}$	**83.28**$_{\pm3.84}$	$78.00_{\pm2.23}$	$48.82_{\pm.00}$	$75.95_{\pm4.83}$	100.00
Meta Maze	reward ↑	$46.80_{\pm2.61}$	$36.42_{\pm6.47}$	$45.35_{\pm3.27}$	$51.20_{\pm1.03}$	$30.35_{\pm20.67}$	**53.00**$_{\pm.57}$	52.50

We first validate this by labeling every proposed hypothesis into one of eight axes using a coding agent (GPT-5.5). We find that ARTS reaches the widest coverage with **4.43** axes per run compared to 4.05 for MLEvolve, which is 1.05 more than the number of initial draft nodes. This shows that prior methods like MLEvolve do not explore much beyond the hypotheses proposed in the first step.

We also measure the entropy of the distribution of a run's proposals over the eight axes. Entropy tells us how evenly those proposals are spread across the axes, so a higher value means the search distributes its budget more uniformly and explores more broadly. We find that ARTS produces an entropy of **1.73**, against 1.35 for MLEvolve and 1.48 for AIRA. This shows that ARTS spreads its budget across the axes more evenly than the baselines, which keep returning to a few.

Next, we perform human validation, we hand-label 111 expansions across 15 runs as either a completely new hypothesis, for instance architectural changes, or a minor change within the current hypothesis, like hyperparameter tuning. We find that **45%** of ARTS's expansions open a new completely new hypothesis, against only 3% for AIRA and 0% for MLEvolve. This means a massive portion of score-only baselines is exploring minor tweaks within the current hypothesis, limiting the exploration required to solve such tasks.

Finally, we calculate the within-run pairwise TF-IDF distance between hypotheses. We find that it averages to **0.72** for ARTS and 0.23 for AIRA and 0.14 for MLEvolve, showing that hypotheses explored by ARTS are lexically far, whereas for the prior methods, the hypotheses are lexically similar.

To illustrate this diversity qualitatively, we present the Vesuvius ink-detection search trees (Appendix Fig. 6). Comparing ARTS and AIRA we find that ARTS holds five distinct axes as siblings at the root (architecture, slice representation, loss, training data, augmentation), identifies that adding the second fragment is a promising strategy ($F_{0.5} = 0.479$), and deepens that one branch with augmentation to reach $\mathbf{0.575}$. Whereas, AIRA focuses on tweaking architectures (3D U-Net, 2.5D U-Net with a pretrained encoder, CRNN, sparse 3D conv, Video-ViT, Swin-UNETR) with a single deep chain.

> **Finding 3.** ARTS spends the exploration budget on **higher quality hypotheses** compared to score-only methods.

Exploring many directions helps only when the hypotheses explored are of high quality. Since each hypothesis is an expensive training run, a search algorithm that fills the 8-hour budget with low quality hypothesis cannot reach to a good solution. In ARTS, before proposing next hypothesis, the `scientist` reads previous nodes' code and training log to identify a specific failure, and proposes a hypothesis that addresses it. Each candidate is therefore well reasoned and high quality, rather than a random mutation of the current best code or an extension of one of the highest-scoring branches.

To qualitatuvely demonstrate this, we present the trees on MetaMaze (Appendix Fig. 7; baseline reward 15.73, partial-observability). The problem requires both preserving 2D maze geometry at the input and recurrent memory. ARTS looks at the baseline logs and diagnoses that the baseline sets flattens the input, vectorising each 2D maze slice before the network sees it, this destroys the wall and goal geometry. ARTS fixes the geometry first by using CNN encoder (23.03), and richer egocentric observation (48.57) it then combines the input fix with an LSTM recurrent policy to reach 53.0, above human best. AIRA applies the textbook PO-MDP prior, LSTM, as its first draft (48.04) without inspecting the input pipeline, then stacks larger nets on the LSTM with improper execution that destabilises the training (17.5–22.3) eventually leading to the method to discard the LSTM family. MLEvolve never leaves PPO knob mutation (entropy, `n_steps`; mean 45.35).

6.3 Does test-time training help ARTS* search efficiently?

Table 2 demonstrates that performance of ARTS improves using our proposed test-time training procedure. On multiple MLGym tasks, test-time training raises the mean normalized score of Qwen3-4B by **40**% from 0.72 to **1.01**. A Qwen3-4B scientist updated with test-time training surpass performance of o3-scientist on half of the tasks and is close on other tasks. We observe prominent performance gains on tasks of language modeling, where the loss improves from 4.34 to **3.52** and attains human performance, and on MetaMaze where our approache exceeds human performance.

Through our qualitative analysis, we find that TTT helps when search finds a good direction but the scientist stops using it after long trajectories. For instance, in MetaMaze, which has a partially observable state and requires memory based solutions, prior algorithms propose LSTM/GRU based policy architectures. Since LSTM/GRU can be unstable with initial implementations, prior appoaches discards them during the search process. In contrast, ARTS with an o3 scientist identifies the need for memory but switches to high-scoring MLP/PPO. However, Qwen3-4B scientist (TTT) reconsiders the recurrent-memory direction and derives a well optimized LSTM-based solution. This suggests that TTT instills useful search experience in the scientist's weights and reduces the dependence on increasingly long context for extended search. Figure 9 visualises the three trajectories side by side.

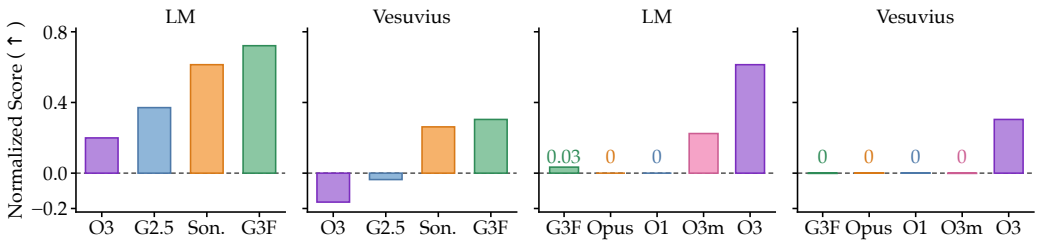
6.4 Ablations.

We study the impact of crucial components of ARTS: the executor model, the scientist model, token and executor-call usage, and the search components audit, verbalized sampling, memory, and initial breadth. Fig. 4 shows that the executor matters even when the scientist is fixed: normalized score rises from 0.20 to 0.72 on Language Modeling and from −0.16 to 0.30 on Vesuvius. With the executor fixed, o3 is the only scientist that reliably moves beyond baseline on these two hard tasks. ARTS also uses fewer estimated tokens than the baselines, 0.72M versus 0.83M for AIRA, 1.53M for Linear, and 2.30M for MLEvolve. Removing audit or verbalized sampling gives the largest component drops, from 0.809 to 0.470/0.506 on HMS and from 0.887 to 0.573/0.614 on Kuzushiji (Appendix Fig. 10). Full numbers are in Appendix L.

For ARTS with TTT, we ablate the reward function, the GRPO episode structure, and whether training collapses diversity. The final percentile reward is best on all three reward-ablation tasks, with mean normalized score **1.03** versus 0.29 for the strongest alternative. Single-step GRPO improves mean normalized score from 0.89 to **1.48** over tree-per-episode rollouts. Training also preserves diversity: runs cover 6–10 strategy categories per task, and entropy increases from early to late rollouts on 3 of 4 measured tasks. Appendix Figs. 11 and 12 give the TTT ablations.

7 Discussion

ARTS replaces heuristic parent selection with reasoning over code, logs, scores, and memory. The qualitative analysis suggests that this helps the agent diagnose why experiments fail, preserve promising directions after weak executions, and open new axes when the current tree is narrow. Across 22 tasks, ARTS gives the best automated-search result on 16 tasks, improves normalized score by 15.3% over the strongest baseline, and has the best IQM, 0.93 versus 0.87 for MLEvolve. It also explores more broadly, with 4.43 axes per run and entropy 1.73. Test-time training raises Qwen3-4B from 0.72 to **1.01** mean normalized score on MLGym.



(a) (b)

Figure 4: Swap ablations. (a) Executor swap with the scientist fixed to o3. (b) Scientist swap with the executor fixed to Gemini-3-Flash. The dashed line marks normalized score 0. Small labels show near-zero normalized scores.

Limitations and Ethical Concerns `ARTS` depends on the quality of its scientist and executor, though the modular design lets stronger reasoning and coding models be swapped in independently. Stronger research agents may also concentrate access among groups with more compute or be misused in unsafe domains; deployment should retain task constraints, logging, and human oversight.

Acknowledgments and Disclosure of Funding

AJ is supported by Fonds de Recherche du Quebec (FRQ) (DOI assigned: https://doi.org/10.69777/350253), Calcul Quebec, and Canada Excellence Research Chairs (CERC) program. The research was enabled in part by computational resources provided by the Digital Research Alliance of Canada (https://alliancecan.ca) and Mila (https://mila.quebec).

We thank Roberta Raileanu for her early feedback on the draft.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. Advances in Neural Information Processing Systems, 2021.

[2] Ekin Akyürek, Mehul Damani, Adam Zweiger, Linlu Qiu, Han Guo, Jyothish Pari, Yoon Kim, and Jacob Andreas. The surprising effectiveness of test-time training for few-shot learning. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=asgBo3FNdg.

[3] Seungju Back, Dongwoo Lee, Naun Kang, Taehee Lee, S. K. Hong, Youngjune Gwon, and Sungjin Ahn. Understanding lora as knowledge memory: An empirical analysis, 2026. URL https://arxiv.org/abs/2603.01097.

[4] Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. Nature, 624:570–578, 2023.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016. URL https://arxiv.org/abs/1606.01540.

[6] Bryan Chan, Xinyi Chen, András György, and Dale Schuurmans. Toward understanding in-context vs. in-weight learning. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=aKJr5NnN8U.

[7] 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 Mądry. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2024. URL https://arxiv.org/abs/2410.07095.

[8] Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. Mars: Modular agent with reflective search for automated ai research, 2026. URL https://arxiv.org/abs/2602.02660.

[9] Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. Sela: Tree-search enhanced llm agents for automated machine learning, 2024. URL https://arxiv.org/abs/2410.17238.

[10] Francois Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. Arc prize 2024: Technical report, 2025. URL https://arxiv.org/abs/2412.04604.

[11] Dean De Cock. Ames, iowa: Alternative to the boston housing data as an end of semester regression project. Journal of Statistics Education, 19(3), 2011. doi: 10.1080/10691898.2011.11889627. URL https://doi.org/10.1080/10691898.2011.11889627.

[12] Google DeepMind. Gemini 3 flash, 2025. URL https://deepmind.google/models/gemini/flash/.

[13] Shangheng Du, Xiangchao Yan, Shiyang Feng, Bo Zhang, Lei Bai, et al. MLEvolve: An autonomous system for end-to-end machine learning algorithm design and optimization. https://github.com/InternScience/MLEvolve, 2026. GitHub repository.

[14] Elliot Glazer, Ege Erdil, Tamay Besiroglu, Diego Chicharro, Evan Chen, Alex Gunning, Caroline Falkman Olsson, Jean-Stanislas Denain, Anson Ho, Emily de Oliveira Santos, Olli Järviniemi, Matthew Barnett, Robert Sandler, Matej Vrzala, Jaime Sevilla, Qiuyu Ren, Elizabeth Pratt, Lionel Levine, Grant Barkley, Natalie Stewart, Bogdan Grechuk, Tetiana Grechuk, Shreepranav Varma Enugandla, and Mark Wildon. Frontiermath: A benchmark for evaluating advanced mathematical reasoning in ai, 2025. URL https://arxiv.org/abs/2411.04872.

[15] Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, Yossi Matias, Andrew Carroll, Kavita Kulkarni, Nenad Tomasev, Yuan Guan,

Vikram Dhillon, Eeshit Dhaval Vaishnav, Byron Lee, Tiago R D Costa, José R Penadés, Gary Peltz, Yunhan Xu, Annalisa Pawlosky, Alan Karthikesalingam, and Vivek Natarajan. Towards an ai co-scientist, 2025. URL https://arxiv.org/abs/2502.18864.

[16] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. Nature, 645(8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

[17] Nathan Herr, Tim Rocktäschel, and Roberta Raileanu. Llm-first search: Self-guided exploration of the solution space, 2025. URL https://arxiv.org/abs/2506.05213.

[18] Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

[19] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In International Conference on Learning Representations, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[20] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation, 2024. URL https://arxiv.org/abs/2310.03302.

[21] Arnav Kumar Jain, Gonzalo Gonzalez-Pumariega, Wayne Chen, Alexander M Rush, Wenting Zhao, and Sanjiban Choudhury. Multi-turn code generation through single-step rewards. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=aJeLhLcsh0.

[22] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138.

[23] Kaggle. Titanic - machine learning from disaster, 2012. URL https://www.kaggle.com/c/titanic. Kaggle competition. Accessed: 2026-05-07.

[24] Kaggle. Histopathologic cancer detection, 2018. URL https://www.kaggle.com/competitions/histopathologic-cancer-detection. Kaggle competition. Accessed: 2026-05-07.

[25] Kaggle. Toxic comment classification challenge, 2018. URL https://www.kaggle.com/competitions/jigsaw-toxic-comment-classification-challenge. Kaggle competition. Accessed: 2026-05-07.

[26] Kaggle. APTOS 2019 blindness detection, 2019. URL https://www.kaggle.com/competitions/aptos2019-blindness-detection. Kaggle competition. Accessed: 2026-05-07.

[27] Kaggle. RSNA-MICCAI brain tumor radiogenomic classification, 2021. URL https://www.kaggle.com/competitions/rsna-miccai-brain-tumor-radiogenomic-classification. Kaggle competition. Accessed: 2026-05-07.

[28] Kaggle. Spaceship titanic, 2022. URL https://www.kaggle.com/competitions/spaceship-titanic. Kaggle competition. Accessed: 2026-05-07.

[29] Kaggle. Vesuvius challenge - ink detection, 2023. URL https://www.kaggle.com/competitions/vesuvius-challenge-ink-detection. Kaggle competition. Accessed: 2026-05-07.

[30] Kaggle. HMS - harmful brain activity classification, 2024. URL https://www.kaggle.com/competitions/hms-harmful-brain-activity-classification. Kaggle competition. Accessed: 2026-05-07.

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

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

[33] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

[34] Zujie Liang, Feng Wei, Wujiang Xu, Lin Chen, Yuxi Qian, and Xinhui Wu. I-mcts: Enhancing agentic automl via introspective monte carlo tree search, 2026. URL https://arxiv.org/abs/2502.14693.

[35] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. Transactions of the Association for Computational Linguistics, 12:157–173, 2024. doi: 10.1162/tacl_a_00638. URL https://aclanthology.org/2024.tacl-1.9/.

[36] Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The ai scientist: Towards fully automated open-ended scientific discovery, 2024. URL https://arxiv.org/abs/2408.06292.

[37] Bodhisattwa Prasad Majumder, Harshit Surana, Dhruv Agarwal, Bhavana Dalvi Mishra, Abhijeetsingh Meena, Aryan Prakhar, Tirth Vora, Tushar Khot, Ashish Sabharwal, and Peter Clark. Discoverybench: Towards data-driven discovery with large language models. In The Thirteenth International Conference on Learning Representations, 2025. URL https://openreview.net/forum?id=vyflgpwfJW.

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

[39] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

[40] OpenAI. Introducing openai o3 and o4-mini, Apr 2025. URL https://openai.com/index/introducing-o3-and-o4-mini/.

[41] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, Advances in Neural Information Processing Systems, volume 37, pages 30811–30849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0970. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/370df50ccfdf8bde18f8f9c2d9151bda-Paper-Datasets_and_Benchmarks_Track.pdf.

[42] Tian Qin, David Alvarez-Melis, Samy Jelassi, and Eran Malach. To backtrack or not to backtrack: When sequential search limits model reasoning, 2025. URL https://arxiv.org/abs/2504.07052.

[43] David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. Gpqa: A graduate-level google-proof q&a benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

[44] Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025. URL https://github.com/algorithmicsuperintelligence/openevolve.

[45] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023. URL https://arxiv.org/abs/2303.11366.

[46] Chenglei Si, Zitong Yang, Yejin Choi, Emmanuel Candès, Diyi Yang, and Tatsunori Hashimoto. Towards execution-grounded automated ai research, 2026. URL https://arxiv.org/abs/2601.14525.

[47] Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2025. URL https://arxiv.org/abs/2407.04620.

[48] Christopher Sutton, Luca M. Ghiringhelli, Takenori Yamamoto, Yury Lysogorskiy, Lars Blumenthal, Thomas Hammerschmidt, Jacek R. Golebiowski, Xiangyue Liu, Angelo Ziletti, and Matthias Scheffler. Crowd-sourcing materials-science challenges with the NOMAD 2018 Kaggle competition. npj Computational Materials, 5:111, 2019. doi: 10.1038/s41524-019-0239-3. URL https://doi.org/10.1038/s41524-019-0239-3.

[49] Ranjita Thapa, Kai Zhang, Noah Snavely, Serge Belongie, and Awais Khan. The plant pathology challenge 2020 data set to classify foliar disease of apples. Applications in Plant Sciences, 8(9):e11390, 2020. doi: https://doi.org/10.1002/aps3.11390. URL https://bsapubs.onlinelibrary.wiley.com/doi/abs/10.1002/aps3.11390.

[50] Edan Toledo, Karen Hambardzumyan, Martin Josifoski, RISHI HAZRA, Nicolas Baldwin, Alexis Audran-Reiss, Michael Kuchnik, Despoina Magka, Minqi Jiang, Alisia Maria Lupidi, Andrei Lupu, Roberta Raileanu, Tatiana Shavrina, Kelvin Niu, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H Miller, Abhishek Charnalia, Derek Dunfield, Carole-Jean Wu, Pontus Stenetorp, Nicola Cancedda, Jakob Nicolaus Foerster, and Yoram Bachrach. AI research agents for machine learning: Search, exploration, and generalization in MLE-bench. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2026. URL https://openreview.net/forum?id=RwfrdKSgCE.

[51] Petar Veličković, Alex Vitvitskyi, Larisa Markeeva, Borja Ibarz, Lars Buesing, Matej Balog, and Alexander Novikov. Amplifying human performance in combinatorial competitive programming, 2024. URL https://arxiv.org/abs/2411.19744.

[52] Zhaotian Weng, Antonis Antoniades, Deepak Nathani, Zhen Zhang, Xiao Pu, and Xin Eric Wang. Group-evolving agents: Open-ended self-improvement via experience sharing, 2026. URL https://arxiv.org/abs/2602.04837.

[53] Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas 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, 2025. URL https://arxiv.org/abs/2411.15114.

[54] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. URL https://arxiv.org/abs/1708.07747.

[55] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

[56] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, Advances in Neural Information Processing Systems, volume 36, pages 11809–11822. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

[57] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023. URL https://arxiv.org/abs/2210.03629.

[58] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments, 2019. URL https://arxiv.org/abs/1903.03176.

[59] Mert Yuksekgonul, Daniel Koceja, Xinhao Li, Federico Bianchi, Jed McCaleb, Xiaolong Wang, Jan Kautz, Yejin Choi, James Zou, Carlos Guestrin, and Yu Sun. Learning to discover at test time, 2026. URL https://arxiv.org/abs/2601.16175.

[60] Jiayi Zhang, Simon Yu, Derek Chong, Anthony Sicilia, Michael Tomz, Christopher D Manning, and Weiyan Shi. Verbalized sampling: How to mitigate mode collapse and unlock LLM diversity, 2026. URL https://openreview.net/forum?id=9jQkmGunGo.

[61] Stephen Zhang, Mustafa Khan, and Vardan Papyan. Attention sinks: A 'catch, tag, release' mechanism for embeddings. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2026. URL https://openreview.net/forum?id=r8UWp9JeJi.

Appendix Contents

The four search-tree figures (§A–§D) are referenced from Findings 1–3 in the main text and visualise the full per-node trajectories described there. The remaining appendices (§F–§L) collect supporting analyses, prompts, experimental details, and ablation tables.

A HMS Search Trajectory: ARTS versus AIRA

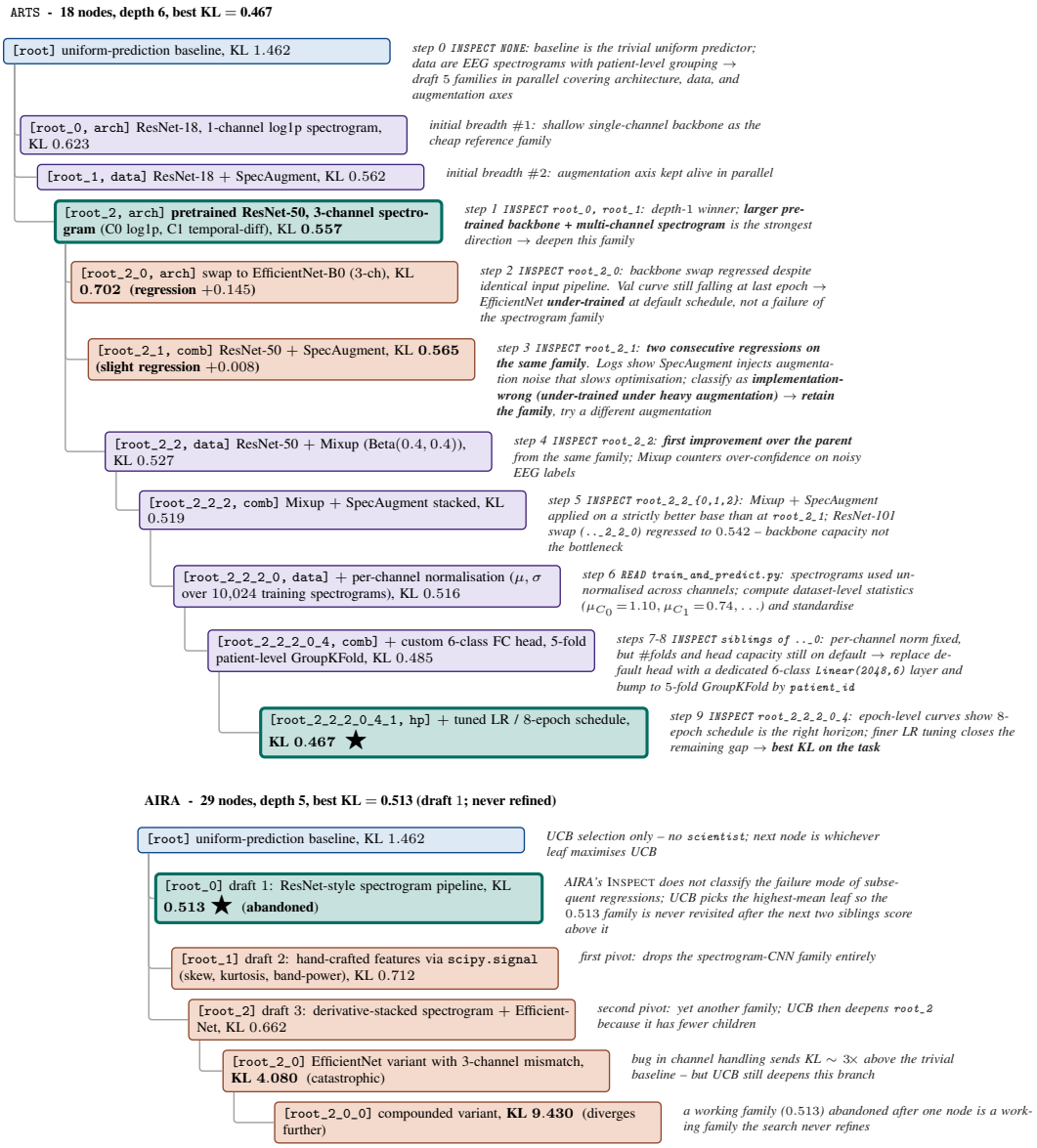


Figure 5: HMS Brain Activity search trees. ARTS (top) drafts a pretrained ResNet-50 over 3-channel log-mel spectrograms at depth 1 and reaches KL **0.557**. The next two children regress (EfficientNet-B0, 0.702; SpecAugment, 0.565). ARTS re-reads the regressed nodes' training logs, classifies them as implementation-wrong (under-trained under heavy augmentation), retains the ResNet-50 + spectrogram family, and the next several expansions compound Mixup, SpecAugment, per-channel normalisation, a 5-fold patient-level GroupKFold, and an LR / schedule sweep to reach **KL 0.467** – the best on the task. AIRA (bottom) reaches a comparable family at its first draft (0.513) but its next two drafts pivot to hand-crafted features and a derivative-stacked EfficientNet; UCB then deepens the latter pivot whose children diverge catastrophically to KL **4.08** and **9.43**, and AIRA never returns to the original 0.513 family.

We contrast ARTS and AIRA on HMS Harmful Brain Activity Classification (MLE-bench; KL divergence, lower is better; uniform-prediction baseline KL 1.462). Tree nodes are coloured **blue** (baseline), **purple** (progress), **teal** (new best, ★), and **red** (regression / abandoned family). Italic

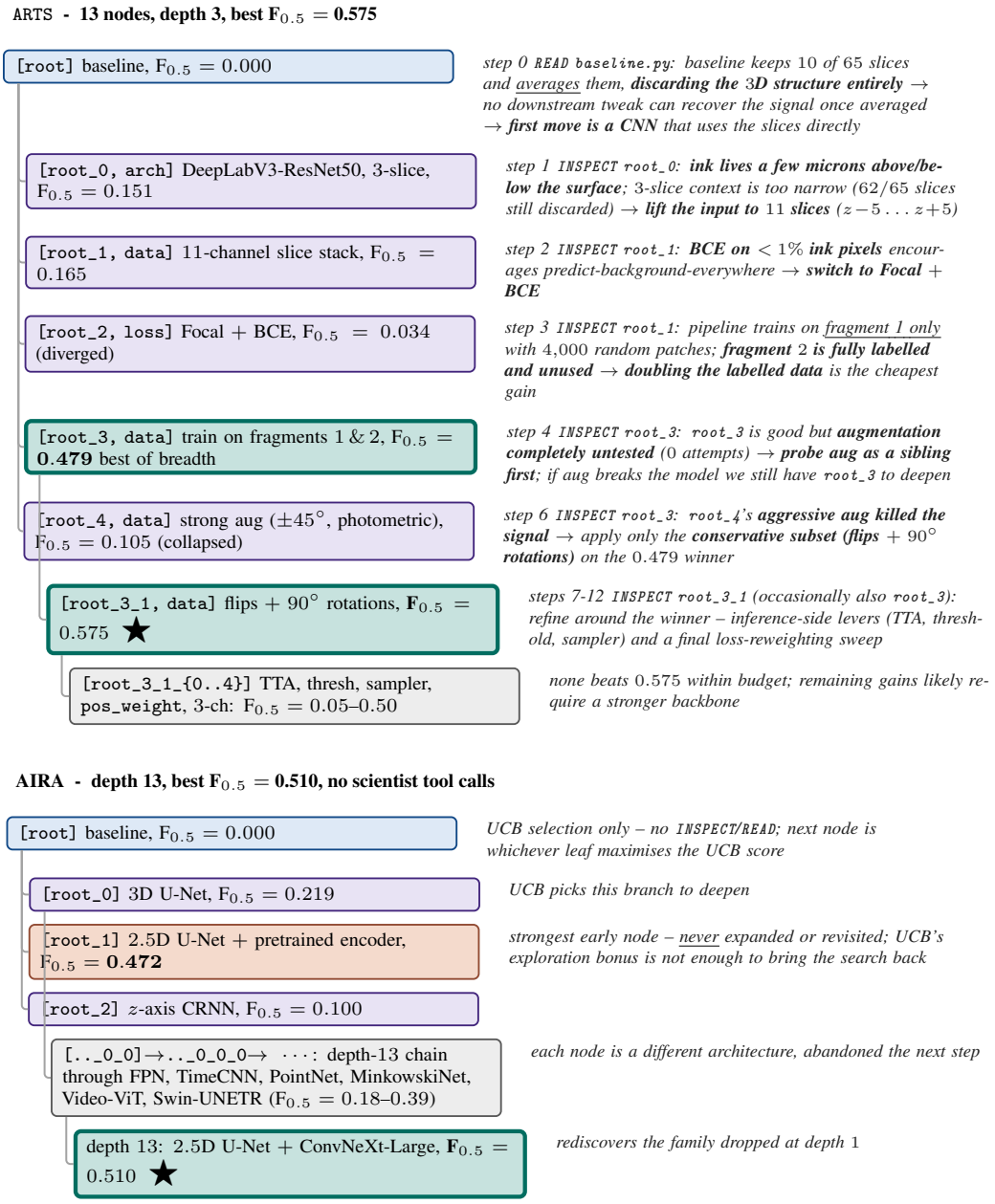


Figure 6: Vesuvius Ink Detection search trees. ARTS (top) keeps five distinct axes as siblings of root, deepens the dominant lever (root_3, both fragments, 0.479) with augmentation to **0.575**. AIRA (bottom) commits to the 3D-U-Net branch via UCB, abandons a stronger 2.5D-pretrained sibling at depth 1, and only rediscovers that family 12 nodes deeper at 0.510.

margin notes give the scientist's tool call (INSPECT/READ) at that step together with the reasoning that drove the next experiment. ARTS reaches best KL **0.467**; AIRA pivots after its first draft (KL 0.513) and the children of its pivot diverge to KL 4.08 and 9.43, never returning to the working family.

B Vesuvius Search Trajectory: ARTS versus AIRA

We contrast ARTS and AIRA on Vesuvius Challenge Ink Detection (MLE-bench; $F_{0.5}$, higher is better). Tree nodes are colored **blue** (baseline), **purple** (progress), **teal** (new best, ★), **red** (regression

/ abandoned strong node). Depth is shown by indentation, `par=X` labels, and gray parent→child L-branches. Italic margin notes give the scientist's tool call and the reasoning that drove the next experiment.

C MetaMaze Search Trajectory: ARTS versus AIRA

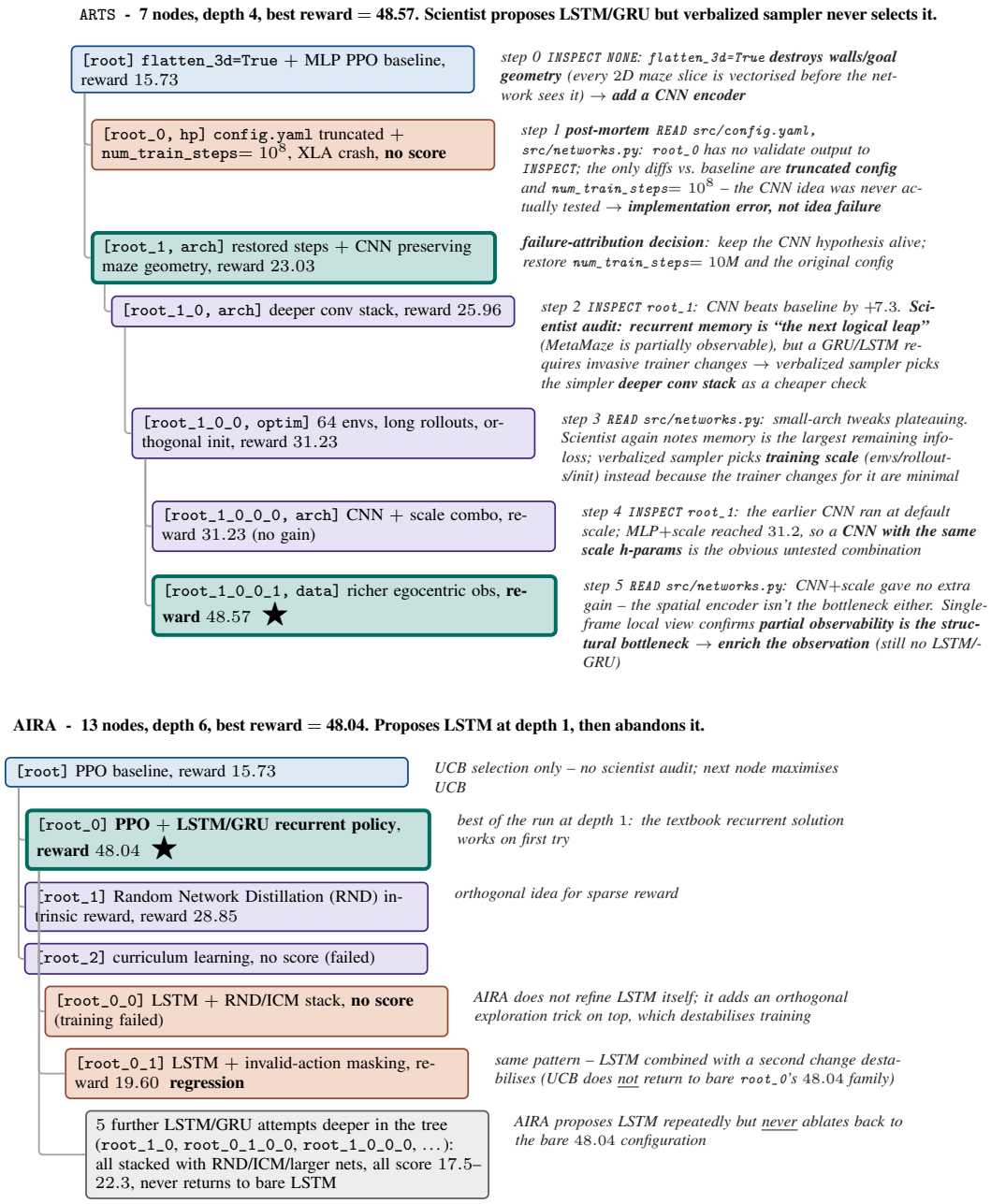


Figure 7: MetaMaze search trees. ARTS (top, 7 nodes) recovers from a config-truncation crash via failure attribution, then climbs 15.73 → 48.57 on a pure-MLP path – the scientist explicitly identifies recurrent memory as the missing axis at steps 2, 3 and 5 but the verbalized sampler always selects a cheaper alternative. AIRA (bottom, 13 nodes) proposes a PPO+LSTM recurrent policy at root_0 and reaches **48.04** on first try, but UCB then deepens elsewhere and never returns to refine the bare LSTM; 5 further LSTM attempts deeper in the tree all combine LSTM with RND/ICM/larger nets and destabilise at 17.5–22.3. Both methods peak near 48 for opposite reasons; the full LSTM gain (48 →≈90) is recovered by test-time training (Table 2, MetaMaze: 30.4 → **53.0**).

We contrast ARTS and AIRA on MLGym MetaMaze (mean evaluation reward, higher is better; ceiling ≈ 100). Both methods reach nearly identical peaks (48.57 for ARTS, 48.04 for AIRA), but for opposite reasons. MetaMaze is partially observable, so a recurrent policy (LSTM/GRU) is the textbook solution. ARTS's scientist repeatedly identifies recurrent memory as "the next logical leap" in its reasoning but the verbalized sampler keeps picking simpler alternatives (deeper conv, training scale, richer observation), so no LSTM/GRU node is ever actually run. AIRA proposes LSTM at `root_0` and reaches the run's best (48.04) on its first try, then deepens the wrong branch, abandons `root_0`, and stacks RND/ICM on top of 5 further LSTM attempts which all destabilise at 17.5–22.3.

D MountainCar Search Trajectory: ARTS versus AIRA

ARTS and AIRA on MLGym MountainCarContinuous (mean reward, higher is better; ceiling $\approx$ 99). Peak scores are similar (~ 95) but ARTS reaches reward 95.73 in 4 steps at depth 4; AIRA reaches 94.82 at depth 13 after 39 expansions ($\sim 10\times$ the compute), with one root draft diverging to $-9{,}708.99$. The MountainCar margin in Table 2 is primarily a robustness gap (mean ARTS $95.73_{\pm.69}$ vs. mean AIRA $80.82_{\pm 11.75}$). Italic margin notes give the scientist's tool call and the reasoning that drove the next experiment.

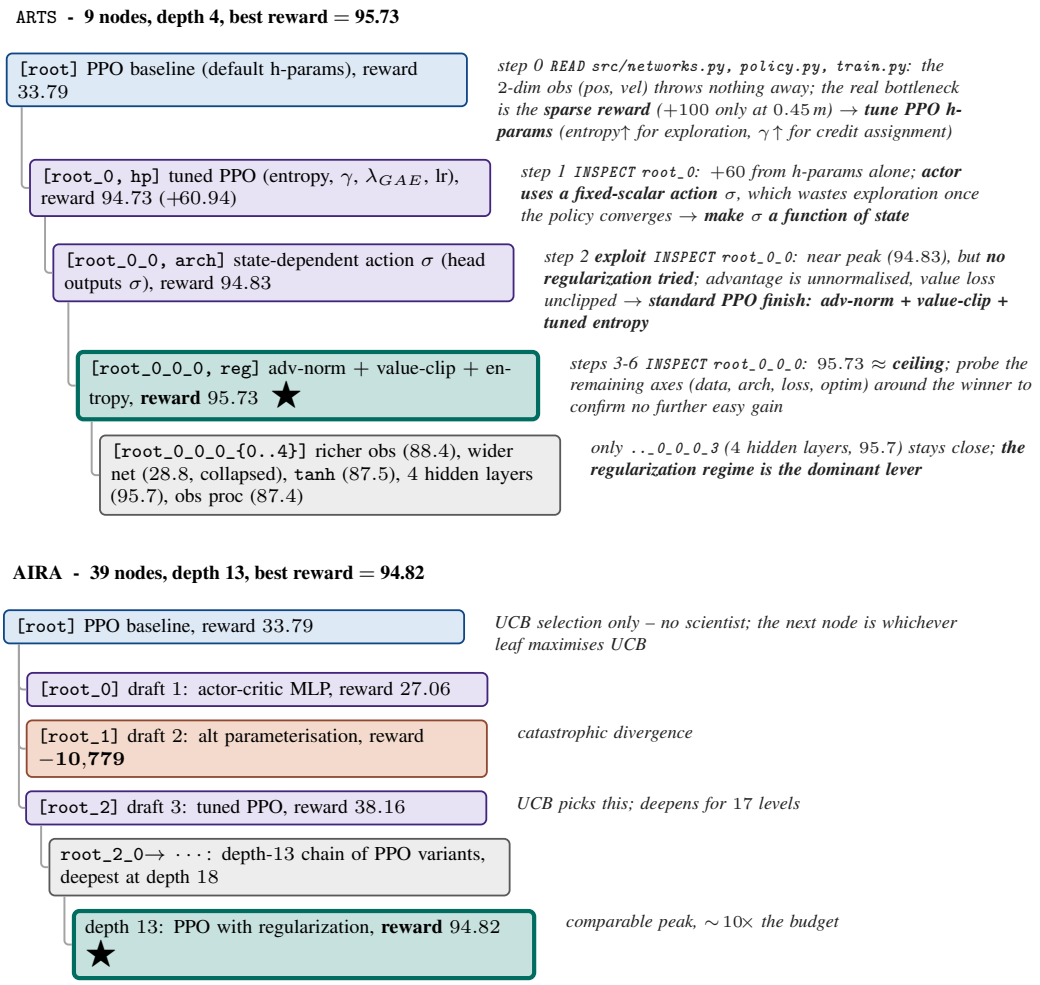


Figure 8: MountainCar search trees. ARTS (top) makes three sequential moves — tune PPO h-params, add state-dependent action σ, add regularization — and reaches reward $\mathbf{95.73}$ at depth 4. AIRA (bottom) drafts three roots; one diverges catastrophically, UCB commits to the 38.16 draft, chains 17 further levels to 94.82. The table margin reflects baseline runs that frequently diverge.

E LLM Use

We use LLMs as coding assistance to write code for this project and we manually verify the code at all times. It was also used for light refinement of the paper like making text terse.

F Qualitative Analysis of Search Efficiency

This section studies whether ARTS searches more efficiently than the baselines. We use the logs to ask the following questions.

1. Can ARTS separate a promising hypothesis from a bad execution?
2. Does auditing identify the root cause of failed or regressed experiments?
3. Does the scientist reason from logs rather than only follow final scores?
4. Does reasoning-based parent selection spend budget better than heuristic selection?
5. Can the scientist propose directions that are not already in the tree?
6. Does verbalized sampling preserve enough diversity?
7. How do baselines fail differently under the same budget?

We answer each question separately below.

F.1 Can ARTS separate a promising hypothesis from a bad execution?

Answer: yes. The logs show that ARTS often preserves a promising hypothesis after a flawed implementation. On HMS Brain Activity, the task is to predict seizure-related activity from EEG spectrograms, and a central difficulty is how to aggregate multiple windows for the same patient. Using more spectrogram windows is promising because the single-window baseline discards much of the labeled signal. The first all-window attempt regressed from 0.963 to 1.237 KL. ARTS did not abandon the idea. It kept the same broad hypothesis and changed the implementation. On Vesuvius, where the task is to segment hidden ink from 3D scroll scans, an EMA regularization attempt returned 0.000 $F_{0.5}$ because the model predicted an all-zero mask. ARTS treated this as a broken EMA implementation, not as evidence that EMA was a bad idea. On FineWeb language modeling, a longer-training attempt timed out after reducing the per-device batch size. ARTS kept the longer-training hypothesis and later moved validation loss from 4.67 to 4.29 and then to 3.97.

F.2 Does auditing identify the root cause of failed or regressed experiments?

Answer: yes. The audit is useful when the score alone is ambiguous. In HMS, the failed all-window node looked like a bad direction from the metric alone. The audit instead found two concrete implementation errors: duplicated `eeg_ids` in the split and averaging probabilities after softmax instead of aggregating logits before softmax. In Vesuvius, the all-zero prediction under EMA was traced to how the EMA model was maintained and evaluated. The next proposal maintained shadow weights after each optimizer step and used the EMA model only for validation and inference. In FineWeb, the timeout was diagnosed as a throughput problem, not as evidence against training longer. These diagnoses are specific enough to change the next experiment.

F.3 Does the scientist reason from logs rather than only follow final scores?

Answer: yes. Several decisions depend on logged failure modes rather than the final score alone. On FineWeb, the scientist inspected the training curves and concluded that the model was undertrained, so it extended training instead of only sweeping batch size or attention heads. On APTOS, a naive resolution increase to 384^2 ran successfully but reduced QWK from 0.932 to 0.906. ARTS did not keep increasing resolution. It reasoned that resolution would only help with better cropping, augmentation, or regularization. On HMS, replacing ResNet18 with ResNet34 under the same pipeline worsened KL from 0.541 to 0.627. Since the run was valid, the scientist treated this as evidence against simply scaling the backbone. These are log-level inferences, not score-only choices.

F.4 Does reasoning-based parent selection spend budget better than heuristic selection?

Answer: yes, by both qualitative evidence and automatic proxies. A score alone cannot tell whether a node failed because the idea was weak or because the implementation was bad. ARTS uses inspection, memory, and reasoning to choose which parent deserves another child. This lets it return to a promising branch after an implementation failure, while avoiding branches that ran cleanly but regressed. We also compute a rough proxy for node-selection quality. We define a valid-but-regressing node as a node that produces a valid score, does not have a fatal execution marker, and is worse than its parent. Such nodes are an imperfect proxy for bad ideas with good execution. On selected tasks with known metric direction, ARTS expanded only 1.4% of valid-but-regressing nodes, compared with 92.7% for AIRA and 64.2% for MLEvolve. The mean number of descendants below such nodes was 0.76 for ARTS, 30.3 for AIRA, and 6.23 for MLEvolve. Among internal nodes, the fraction that eventually led to an improving descendant was 80.4% for ARTS, compared with 75.8% for AIRA and 64.1% for MLEvolve. These numbers are noisy because execution status is not uniformly logged across methods, but they support the qualitative pattern that ARTS spends less budget on cleanly executed but unpromising branches.

F.5 Can the scientist propose directions not already in the tree?

Answer: yes. The scientist is not restricted to mutating the current best code. It can read the tree, inspect logs, consult memory, and propose a new hypothesis before the executor writes code. On HMS, after a strong ResNet18 branch, the scientist did not simply continue local tuning. It listed three directions: combine full-window coverage with the strong backbone, try a larger pretrained architecture, and change the loss for class imbalance. These candidates were assigned probabilities of 0.50, 0.30, and 0.20, and the sampled experiment combined the relevant branches. This is evidence that the scientist can propose directions not already represented by the current best lineage.

F.6 Does verbalized sampling preserve enough diversity?

Answer: yes. Verbalized sampling forces the scientist to list multiple candidate hypotheses with explicit probabilities before the system samples one. The probabilities represent the scientist's current belief about which hypotheses are worth trying after inspecting the tree, logs, and memory. This makes the proposal distribution visible and keeps the search from collapsing to a single local edit. We measure rough strategy coverage with keyword labels over node hypotheses. ARTS visits 4.11 unique axes per run on average, compared with 2.20 for AIRA and 2.43 for MLEvolve. The axis-distribution entropy is 1.16 for ARTS, 0.43 for AIRA, and 0.58 for MLEvolve. These numbers support the claim that verbalized sampling increases diversity. However, diversity alone is not enough. The stronger finding is that ARTS combines diversity with inspection: it opens new axes when the audit or memory says an axis is missing, and it returns to old parents when a promising idea failed for an execution reason.

F.7 How do baselines fail differently under the same budget?

Answer: the baselines fail in method-specific ways. Linear search is brittle because it continues the last node and has no explicit mechanism to return to an older promising parent after a bad implementation. AIRA can sometimes describe that a failure is likely a bug, but the diagnosis is less reliably coupled to the next parent choice. In HMS, AIRA expanded an unstable spectrogram-CNN branch whose score had already degraded to 4.08 KL; the next child worsened further to 9.43 KL. On FineWeb, AIRA produced root-level attempts without a valid submission, including a CUDA out-of-memory failure during validation, and did not recover a scored branch from that failure. MLEvolve explores through mutation and crossover, which can be useful for compact programs, but its reflections are sometimes weakly grounded in the code that actually ran. In MetaMaze, for example, it sometimes attributes performance to recurrent LSTM/GRU memory even when the logged branch still uses a flattened feedforward representation. These failures are not just lower scores. They show weaker coupling between evidence, parent selection, and the next executable experiment.

G Failure Attribution: Detailed Examples

This appendix expands the main-text Finding 1 with full per-node trajectories. Each example contrasts what a score-only policy would read from the regression with what ARTS's idea-wrong / implementation-wrong attribution actually inferred, and what the next expansion did as a consequence.

G.1 HMS Brain Activity: full trajectory

HMS Harmful Brain Activity Classification (MLE-bench; KL divergence, lower is better; uniform-prediction baseline KL 1.462). The trajectory below is from the ARTS run `llmg_C_vs_C_v3` reported in Table 1. The full tree with scientist tool calls is in Fig. 5 (Appendix A).

Initial drafts (depth 1). ARTS drafts five families in parallel. The strongest is a pretrained ResNet-50 over 3-channel log-mel spectrograms (`root_2`, KL 0.557); a 1-channel ResNet-18 baseline (`root_0`, 0.623) and a data-axis variant (`root_1`, 0.562) are sibling families.

Two consecutive regressions (depth 2). The two children of the strong ResNet-50 family both regress: `root_2_0` swaps the backbone to EfficientNet-B0 and lands at KL 0.702 (a $+0.145$ regression); `root_2_1` keeps ResNet-50 and adds SpecAugment (0–2 frequency masks ≤ 12 bins, 0–2 time masks $\leq 20\%$ of time steps) and lands at KL 0.565 (a small $+0.008$ regression). A score-only policy reads two regressions in a row as evidence that the spectrogram-CNN family is exhausted.

Failure-attribution decision. ARTS re-reads the training logs of both regressed nodes. The validation KL is still decreasing at the last epoch in both runs — SpecAugment in particular makes the optimisation slower because it injects augmentation noise. The scientist classifies the regression as **implementation-wrong** (under-trained under heavy augmentation, not a failure of the spectrogram representation) and retains the ResNet-50 + spectrogram family.

Recovery and refinement (depth 3+). The next expansions stay on the ResNet-50 + spectrogram axis and improve KL monotonically: `root_2_2` adds Mixup with Beta$(0.4, 0.4)$ to ResNet-50 (KL 0.527, the first improvement over the parent); `root_2_2_2` stacks Mixup and SpecAugment together (0.519); `root_2_2_2_0` adds per-channel normalisation with dataset-level statistics (μ, σ computed across the $10{,}024$ training spectrograms: $\mu_{C_0} = 1.10$, $\mu_{C_1} = 0.74$, μ_{C_2} similar) to reach 0.516. Continued refinement on the same family ultimately reaches **KL $\mathbf{0.467}$** — the best on the task.

Contrast with AIRA on the same task. AIRA's first draft also finds a strong spectrogram-CNN family at KL 0.513. Its next two expansions, however, pivot to entirely different pipelines: a hand-crafted feature pipeline using `scipy.signal` with `skew`/`kurtosis` statistics, and a derivative-stacked EfficientNet over the same spectrograms. The children of these pivots diverge catastrophically — `root_2_0` reaches KL 4.08 (roughly $3\times$ worse than baseline) and `root_2_0_0` reaches KL 9.43 — and AIRA never returns to the original 0.513 family. The cost of conflating "why" with "that" is visible here: a working family abandoned after one node is a working family the search never refines.

G.2 APTOS Diabetic Retinopathy: pretrained-feature collapse

APTOS 2019 (MLE-bench; quadratic-weighted kappa, higher is better).

The collapsed first attempt. The initial draft is a ConvNeXt-Base backbone, fully fine-tuned end-to-end at learning rate 10^{-3}. The training loss decreases over the first epoch but the held-out QWK is exactly $\mathbf{0.000}$: the high learning rate erases the pretrained ImageNet features within the first epoch and the model converges to a constant prediction.

Score-only attribution. A score-only policy reads the 0.000 as evidence that ConvNeXt is the wrong backbone for this task and pivots to a different architecture family (typical baseline responses are ResNet-50, EfficientNet, or a custom shallow CNN).

ARTS's attribution. The scientist inspects the training log, observes that the train loss decreases monotonically while the validation QWK is pinned at 0.000, and reads this as collapse to a degenerate constant prediction — consistent with the learning rate being too large to preserve the pretrained features. The regression is classified as **implementation-wrong (learning rate schedule)** and the ConvNeXt-Base backbone is retained.

Corrected execution. The next expansion edits the training script to: (i) train the classification head only for one epoch with the backbone frozen (linear-probe warmup); then (ii) unfreeze the backbone and continue at learning rate 3×10^{-5} with the same optimiser. The corrected run reaches QWK $\mathbf{0.92}$.

G.3 Vesuvius Ink Detection: a broken validation split

Vesuvius Ink Detection (MLE-bench; $F_{0.5}$, higher is better).

A misleadingly low score. The first DeepLabV3-ResNet50 draft uses a 3-channel input ($z-1, z, z+1$) and reaches $F_{0.5} = 0.151$. A subsequent expansion widens the input to 11 channels ($z-5$ to $z+5$, intended to capture more z-axis context) and lands at $F_{0.5} = 0.165$ — barely above the 3-slice version.

Score-only attribution. A score-only policy reads the 0.165 as evidence that the wider receptive field does not help, prunes the wide-input branch, and pivots to a different architecture (this is what AIRA does on the same task — see Fig. 6 — chaining 3D U-Net, 2.5D U-Net, CRNN, PointNet, sparse-3D conv, Video-ViT, and Swin-UNETR down a single chain, each abandoned after one node).

`ARTS`**'s attribution.** The scientist re-reads the training log line by line. The log contains the line `DATA: train=4000 val=0`: the validation split was set to zero by the training script, so `val_metric` is constant at zero throughout training and the reported $F_{0.5} = 0.165$ is computed on an unrepresentative held-out shard. The regression is classified as **implementation-wrong (broken validation split)** rather than as a failure of the wide-input idea.

Corrected execution. The next expansion retains the DeepLabV3 + wide-slice family with the validation split repaired, then adds the second scroll fragment to the training data (`root_3`, $F_{0.5} = 0.479$ — a large jump), and finally adds tile/flip augmentation on top of fragment-2 (`root_3_1`, $F_{0.5} = \mathbf{0.575}$), the best on the task.

G.4 Why the same pattern shows up across tasks

The three examples are not isolated. They share a common structure: a hypothesis that is sound as a research direction surfaces a low score because of an execution-side artefact (an over-aggressive learning rate, an under-trained model under augmentation, a broken validation split). A score-only policy reads the low score as evidence against the hypothesis and pivots; the working family is then either lost entirely (HMS, Vesuvius under AIRA) or rediscovered many nodes later in a degraded form (Vesuvius: AIRA only returns to the 2.5D-pretrained variant at depth 13, $F_{0.5} = 0.510$). `ARTS`'s requirement to label every regression as idea-wrong or implementation-wrong forces the scientist to inspect the code and the log before reassigning the budget, which in each case here points to a specific, fixable execution issue rather than to abandonment.

H Extended Related Work

Autonomous scientific discovery. Recent systems use language models as proposal engines for scientific discovery. FunSearch [51] combines an LLM with an evaluator in an evolutionary loop and discovers new constructions in extremal combinatorics. AlphaEvolve [39] uses Gemini as a code mutation engine and reports improvements in algorithmic domains such as matrix multiplication. CosScientist [4] and Google's AI Co-Scientist [15] study hypothesis generation and planning in wet-lab settings. The Sakana AI Scientist [36] automates parts of the ML research pipeline, including hypothesis generation, code writing, execution, and write-up. These systems differ from our setting in both domain and search structure. We study ML research tasks where each candidate requires writing and running training code, and where the main question is how to choose which prior node to expand.

Linear agents. MLGym [38] and MLAgentBench [20] evaluate agents that interact with ML workspaces through sequential actions. AutoResearch [31] adds planning, but still follows a single trajectory. This structure is computationally simple and easy to deploy, but it commits strongly to early decisions. Reasoning models can backtrack within a chain of thought [16, 42], but the revision happens inside the current context window. In long ML runs, the agent accumulates code, logs, validation scores, and failure traces. Prior work on long-context models shows that information retrieval and reasoning degrade as relevant evidence is pushed deeper into context [35, 18]. This makes linear search fragile when the useful direction is an earlier hypothesis that failed for implementation reasons.

Tree search. Tree-based agents keep an explicit search tree and choose a node to expand. Greedy variants such as AIDE [22] expand the best-scoring node, while MCTS variants such as AIRA [50] and AI Scientist v2 [36] use UCT-style selection. These methods import a useful abstraction from games, but ML research does not have cheap or reliable rollouts. A node score is a noisy mixture of hypothesis quality, implementation correctness, data handling, training stability, and evaluation details. A valid idea can receive a poor score because of a coding error, and a weak idea can receive a high score because the implementation is easier. Score-based backpropagation therefore has the wrong primitive for deciding which scientific direction deserves more attempts.

Evolutionary search. Evolutionary and population-based systems such as FunSearch [51], AlphaEvolve [39], OpenEvolve [44], and MLEvolve [13] maintain a population of programs and apply mutation or crossover. This is effective when the artifact being searched is short and compositional. ML training pipelines are less modular. Data loading, model architecture, losses, augmentations, optimizers, and validation logic interact in tightly coupled ways. Copying a block from one branch into another can break hidden assumptions rather than combine useful ideas. Evolutionary search also spends many evaluations on program variants, which is expensive when each node can require minutes to hours of GPU time.

LLM-guided search. Several works use language models to replace or augment hand-designed search heuristics. Verbalized sampling [60] asks the model to express a distribution over candidates, which can reduce collapse to a single likely continuation. We use this idea when the scientist proposes hypotheses, because greedy continuation often produces small variants of the current best branch. The key difference in `ARTS` is that the scientist also selects the parent node. It can inspect logs and code from prior attempts and distinguish a failed implementation from a failed hypothesis.

Test-time training. Test-time training was introduced as a way to adapt a model on each test instance [47]. Parameter-efficient variants use adapters such as LoRA to update only a small set of weights [2]. TTT-Discover [59] and execution-grounded agent training [46] apply on-policy optimization to research agents, using rollout outcomes as the learning signal. Our setting differs in the structure of the policy. We train a tree-structured scientist that chooses what to inspect, which parent to expand, and which hypothesis to test. This lets the model absorb useful search experience into its weights while still using execution feedback from the current task.

I Additional Experimental Details

Execution environment. All runs execute inside Apptainer containers with the same executor sandbox, action set, and validation interface. We restore the evaluation script before every validation. We added this guard because agents sometimes attempted to modify evaluation code despite instructions not to do so. Restoring the script prevents metric hacking and keeps the comparison focused on legitimate changes to the submission or training code.

Models and prompts. For the main ARTS experiments, the scientist is OpenAI o3 [40] and the executor is Gemini 3 Flash [12]. For baselines that do not separate scientist and executor roles, we use Gemini 3 Flash for both reasoning and coding. We also report ablations that swap the scientist and executor models. The scientist and executor prompts are given in Appendix K.

ARTS **settings.** Each run receives an 8-hour wall-clock budget. ARTS uses an audit length $R = 3$ scientist calls. During proposal, the scientist enumerates $K = 5$ candidate hypotheses using verbalized sampling, as described in §4.1. One candidate is sampled and passed to the executor, which implements and validates the experiment in the sandbox.

TTT settings. For test-time training, we use Qwen3-4B-Instruct [55] as the scientist. TTT uses LoRA adapters of rank 32 with $\alpha = 64$ and dropout 0 on all attention and MLP linear projections. We train with AdamW using learning rate 1×10^{-5}, constant schedule, $\beta = (0.9, 0.999)$, weight decay 0.01, and gradient clipping at 1.0. Each GRPO step samples $N = 8$ rollouts per group and applies one inner update with KL penalty $\beta_{\mathrm{KL}} = 0.01$ against the frozen pretrained base. Per-token importance ratios are clipped to $[0.8, 1.28]$.

Rollout limits. Rollouts use sequence length 8192 with up to 120 scientist turns of 1024 tokens each. Each sampled candidate is executed once, with no best-of-N reranking. LoRA adapters are checkpointed and broadcast to the inference server every 5 GRPO steps. Training is capped at 100 GRPO steps or the 8-hour wall-clock budget, whichever comes first.

Compute. All experiments use 40 GB A100 GPUs. Each inference-only experiment, including Linear, AIRA, MLEvolve, and ARTS without TTT, uses one GPU for the executor sandbox, for 8 GPU-hours per run. Each TTT experiment uses three GPUs, one for the LoRA trainer, one for the vLLM inference server hosting Qwen3-4B, and one for the executor sandbox, for 24 GPU-hours per run.

J Sources for Human-Best Scores

Table 3 records the provenance of the human-best values reported in Table 1. For MLEBench tasks, the value is the top-1 Kaggle leaderboard score for the corresponding competition. For MLGym tasks, the value is the published best-known result for the benchmark task or environment. We include the table to make the normalization constants auditable.

Task	Human Best	Source
Titanic	0.830	MLGym [38]; Kaggle task [23]
CIFAR-10	0.994	MLGym [38]; CIFAR-10 [32]
Fashion MNIST	0.968	MLGym [38]; Fashion-MNIST [54]
House Price	0.990	MLGym [38]; Ames Housing [11]
MNLI	92.50	MLGym [38]; MultiNLI
Lang. Modeling	3.500	MLGym [38]; FineWeb [41]
Battle of Sexes	1.667	MLGym task specification [38]
Prisoner's Dilemma	3.000	MLGym task specification [38]
Blotto	0.500	MLGym task specification [38]
MountainCar	99.00	MLGym [38]; OpenAI Gym [5]
Breakout	100.00	MLGym [38]; MinAtar [58]
Meta Maze	52.50	MLGym task specification [38]
Spaceship Titanic	0.828	MLE-bench/Kaggle leaderboard [7, 28]
Nomad 2018	0.051	MLE-bench/Kaggle leaderboard [7, 48]
Jigsaw Toxic	0.989	MLE-bench/Kaggle leaderboard [7, 25]
APTOS 2019	0.936	MLE-bench/Kaggle leaderboard [7, 26]
Plant Pathology	0.984	MLE-bench/Kaggle leaderboard [7, 49]
Histopathologic Cancer	1.000	MLE-bench/Kaggle leaderboard [7, 24]
Vesuvius Ink Detection	0.831	MLE-bench/Kaggle leaderboard [7, 29]
Kuzushiji Recognition	0.950	MLE-bench/Kaggle leaderboard
HMS Brain Activity	0.272	MLE-bench/Kaggle leaderboard [7, 30]
RSNA Brain Tumor	0.621	MLE-bench/Kaggle leaderboard [7, 27]

Table 3: Sources for the human-best scores used to normalize results. MLGym values follow the published benchmark task definitions. MLEBench values use the public Kaggle leaderboard protocol from MLE-bench, with task-level sources shown for auditability.

K Scientist Prompts

The scientist operates in two turns per node expansion (three during the research phase). Turn 1 is shared between the active and research phases; Turn 2 differs. All `{placeholders}` are filled at runtime.

K.1 Turn 1 — Inspection

Scientist – Turn 1: Inspection

```
You are an ML research ADVISOR. You propose ONE experiment per turn.
A separate coder (the "executor") executes it. You must NEVER write
code (no ‘‘‘ blocks, no import statements, no def/class).

Think like a scientist: analyze what has been tried, identify gaps,
form a hypothesis, and propose a direction.

Be aware that per-node training cost varies by task -- some nodes
finish in minutes, others take 30+ minutes. Coupled changes (LR
schedule + longer training, augs + the regularization that enables
them) bundle naturally; unrelated levers do not.

## Task
{task_description}

## Task Details (this is what the executor sees)
{task_details}

The metric is: {metric_name} ({direction} is better)
Baseline score (no model, just default): {baseline_score}

## How This Works

Each time you propose an experiment, the executor writes code from
scratch in a container, runs it, and validates. It has {max_actions}
actions (shell commands) per attempt. Each attempt creates one "node"
in your search tree. The executor can run any single command for up
to 45 minutes.

## Budget

Total run time: {time_budget_min} min.
Time elapsed: {time_elapsed_min} min.
Time remaining: {time_remaining_min} min.
Nodes scored so far: {nodes_done}
  (avg {avg_per_node_min:.1f} min each).
At the current per-node pace, ~{nodes_remaining_est} more nodes fit
in the remaining budget. Factor this into how ambitious each
candidate is.

IMPORTANT: The executor already has ALL source files from the
workspace pre-loaded in its context. Do NOT waste a node asking it
to "read" or "examine" files -- every experiment should be an
ACTIONABLE change (modify config, swap architecture, tune
hyperparameters), never exploration.

INTEGRITY: Do NOT modify evaluation files, opponent strategy files
(e.g. target.py), or any read-only starter code. Do NOT use
monkey-patching, sys.modules hacking, or any technique to manipulate
the scoring system. All such modifications are reverted before
evaluation. Only modify YOUR submission files (e.g. strategy.py,
baseline.py, train_and_predict.py). Legitimate improvements only.

## Your Search Tree
{tree_view}

## Your Accumulated Knowledge
{memory_section}

## Your Task Now
```

```
Before making a decision, you have two tools:

1. INSPECT nodes -- see the actual commands and output the executor
   ran for any node. Lets you understand EXACTLY what was tried and
   why it succeeded or failed.

2. READ files -- see the contents of any workspace file (e.g.
   target.py, evaluate.py, baseline.py, strategy.py). Lets you
   understand the task code, opponent strategy, evaluation logic, or
   data format before proposing an experiment.

Respond in EXACTLY this format:

INSPECT: node_id_1, node_id_2
[OR]
INSPECT: NONE

READ: filename1.py, filename2.py
[OR]
READ: NONE

Brief explanation of what you want to understand.
```

K.2 Turn 2 — Decision (Active Search)

Scientist – Turn 2: Decision

```
Good. Now make your decision.

{code_inspection}

You are an ML research ADVISOR. You propose ONE experiment per turn.
You must NEVER write code (no ``` blocks, no import statements, no
def/class).

Think like a scientist: analyze what has been tried, identify gaps,
form a hypothesis, then propose a specific experiment.

Tree stats: {explore_stats}
Time remaining: {time_remaining_min} min
  ({nodes_done} scored, ~{avg_per_node_min:.1f} min/node).

## Rules

- NO CODE. Describe everything in precise English.
- DIVERSITY: If one axis dominates (>50% of attempts), deliberately
  explore an UNEXPLORED axis. Axes: architecture, loss,
  data_representation, data_augmentation, regularization, optimizer,
  hp, combination. Note: data_representation (what ENTERS the model
  -- input channels, slice ranges, resolution, patch size) and
  data_augmentation (transforms applied to each sample -- flips,
  rotations, mixup, etc.) are TWO DIFFERENT axes. Be specific --
  name exact technique names and parameter values.
- BASELINE AUDIT (mandatory before any loss/optimizer/aug/reg tweak):
  audit the data pipeline. What signal does the raw input contain,
  and what fraction does the current pipeline preserve? Identify the
  single largest information-loss step. If a downstream tweak cannot
  plausibly close a gap the pipeline itself creates, fix the pipeline
  FIRST. State the bottleneck in your ANALYSIS.
- EVOLVE: Two valid ways to build on prior nodes:
  (a) LAYER -- extend ONE high-scoring parent with a new idea. Set
      PARENT to that parent, COMBINES: NONE.
  (b) MERGE -- combine high-scoring nodes from DIFFERENT branches
      that succeeded on DIFFERENT axes. Set PARENT to the stronger;
      list others in COMBINES. Reserve MERGE for when the tree has
      >=2 distinct branches with non-trivial scores and 3+ recent
      single-parent attempts have stalled.
- BUILD ON SIGNAL: Early in a task, prefer single-axis probes. Once
  axes are individually validated, compound them.
- Failed nodes are DATA -- analyze WHY they failed. A code failure
  does NOT mean the approach is wrong.
- REGRESSION INVESTIGATION: if any recently completed node scored
  >20% below its parent, you MUST have INSPECTed it and emit a
  CLASSIFICATION: line. Valid values:
    IDEA-WRONG: <node_id> -- <brief why, from inspection>
    IMPLEMENTATION-WRONG: <node_id> -- <specific code bug>
    NONE (only if no regression >20% exists)
  If IMPLEMENTATION-WRONG: your EXPERIMENT must target the SAME
  AXIS with corrected executor instructions.
- Do NOT abandon promising nodes prematurely (< 3 children).
- If 3+ refinements in the same family stalled, switch families
  (classical ML -> deep learning, CNN -> transformer, etc.).

## Your Output

Respond in EXACTLY this format:
```

```
ANALYSIS:
[Thorough analysis:
 - What has been tried and what scores?
 - What axes are MISSING or underexplored?
 - BASELINE AUDIT: largest information-loss step?
 - Which high-scoring nodes could improve further?
 - What do the failures tell us?]

Enumerate {n_candidates} fundamentally different candidates. Span
DIFFERENT axes OR DIFFERENT families within an axis.
{cap_instruction}

The system samples ONE candidate weighted by <probability>. Be
honest -- DO NOT inflate your favorite. The system, not you, picks.

<candidates>
<response>
<direction>[1 sentence naming the family/technique]</direction>
<probability>NUMBER</probability>
<plan>
HYPOTHESIS: [1-2 sentences: why this improves performance]
EXPERIMENT: [3-6 sentences: WHAT (exact models, values), HOW
  (which components to swap), WHY (expected outcome). As precise
  as a methods section. No code.]
PARENT: [node_id to build on, or "root"]
COMBINES: [node_ids to merge, or NONE]
AXIS: [architecture | loss | data_representation |
       data_augmentation | regularization | optimizer |
       hp | combination]
MODE: [explore | exploit]
</plan>
</response>
[...exactly {n_candidates} responses; probs sum to 1.0...]
</candidates>

MEMORY:
[One sentence: what you LEARNED. Must include evidence (score,
error) and an insight. Do NOT repeat prior memory.
GOOD: "CatBoost (0.91) + LightGBM (0.90) plateau -- try
      feature engineering next."
BAD:  "CatBoost works well." (no new insight)
Write NONE if no genuinely new insight.]

CLASSIFICATION:
[Mandatory. Exactly ONE of:
  IDEA-WRONG: <node_id> -- <brief reason from inspection>
  IMPLEMENTATION-WRONG: <node_id> -- <specific bug>
  NONE
If IMPLEMENTATION-WRONG, EXPERIMENT above must target the SAME
AXIS as the regressed node with corrected instructions.]
```

K.3 Turn 2 — Research Phase

Scientist – Research Phase Turn 2

```
You are in the RESEARCH PHASE -- NOT proposing experiments yet.

{code_inspection}

Your job is to BUILD A MENTAL MODEL of the task before running
experiments. A good researcher spends the first day understanding
the problem: what the data looks like, what the metric rewards, what
the baseline computes, where the easy wins are, where the hidden
traps are. That is what this turn is for.

Do NOT propose an experiment. Do NOT pick a node to expand. Write
structured findings that future-you will use to guide the real
search.

## Your Output

Respond in EXACTLY this format:

FINDINGS:
[3-8 specific, concrete, actionable observations. Examples:
 - "Input volumes have 65 z-slices but the baseline uses only
   slices 28-33 -- ~90% of z-axis signal is discarded."
 - "Evaluation uses F0.5 which weights precision 2x recall --
   threshold choice will matter a lot."
 - "Fragment 1 has ~20x more positive pixels than Fragment 2 --
   class imbalance differs across training fragments."
No proposals. Findings only.]

OPEN QUESTIONS:
[2-4 concrete things you still don't understand and would
investigate via READ / INSPECT in future research turns.
Be specific -- not "how does training work" but "what exactly
does the eval script consider a valid submission format?"]

MEMORY:
[1-3 one-sentence insights for the active search phase. Each must
be novel, evidence-backed, and action-implying.
GOOD: "Baseline averages 5 central z-slices then triples them --
      ~90% of z-axis info discarded; any downstream tweak
      ceiling-limits around this loss."
BAD:  "The task is hard." (too vague, no action implied)
Write NONE if no genuinely new insight.]
```

L Extended Ablations

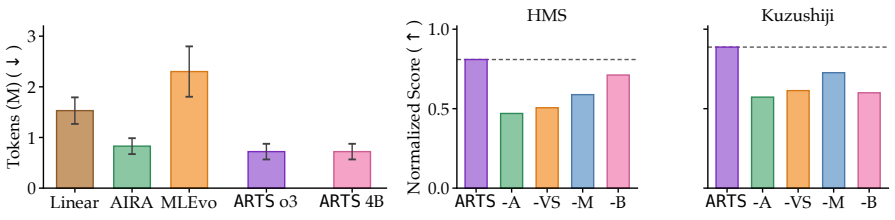
This section reports the full ablations summarized in §6.4. We include raw task scores and normalized scores using the same baseline and human-best normalization as the main results.

L.1 Model Swaps

Table 4: Model-swap ablations. In the executor swap, the scientist is fixed to o3. In the scientist swap, the executor is fixed.

Ablation	Task	Model	Raw score	Norm.
Executor	LM (loss ↓)	o3	4.439	0.199
Executor	LM (loss ↓)	Gemini 2.5 Pro	4.238	0.371
Executor	LM (loss ↓)	Claude Sonnet	3.953	0.614
Executor	LM (loss ↓)	Gemini 3 Flash	3.827	0.721
Executor	Vesuvius (F_β ↑)	o3	0.000	-0.164
Executor	Vesuvius (F_β ↑)	Gemini 2.5 Pro	0.091	-0.036
Executor	Vesuvius (F_β ↑)	Claude Sonnet	0.304	0.262
Executor	Vesuvius (F_β ↑)	Gemini 3 Flash	0.334	0.304
Scientist	LM (loss ↓)	Gemini 3 Flash	4.633	0.034
Scientist	LM (loss ↓)	Claude Opus	4.673	0.000
Scientist	LM (loss ↓)	o1	4.673	0.000
Scientist	LM (loss ↓)	o3-mini	4.410	0.224
Scientist	LM (loss ↓)	o3	3.953	0.614
Scientist	Vesuvius (F_β ↑)	Gemini 3 Flash	0.116	-0.001
Scientist	Vesuvius (F_β ↑)	Claude Opus	0.118	0.001
Scientist	Vesuvius (F_β ↑)	o1	0.118	0.001
Scientist	Vesuvius (F_β ↑)	o3-mini	0.116	-0.001
Scientist	Vesuvius (F_β ↑)	o3	0.334	0.304

L.2 Token Usage and Components



(a) (b)

Figure 10: Token and component ablations. (a) Token usage estimated from saved trajectory text. (b) Component ablations, where A, VS, M, and B denote audit, verbalized sampling, memory, and initial breadth. The dashed line marks the full ARTS score.

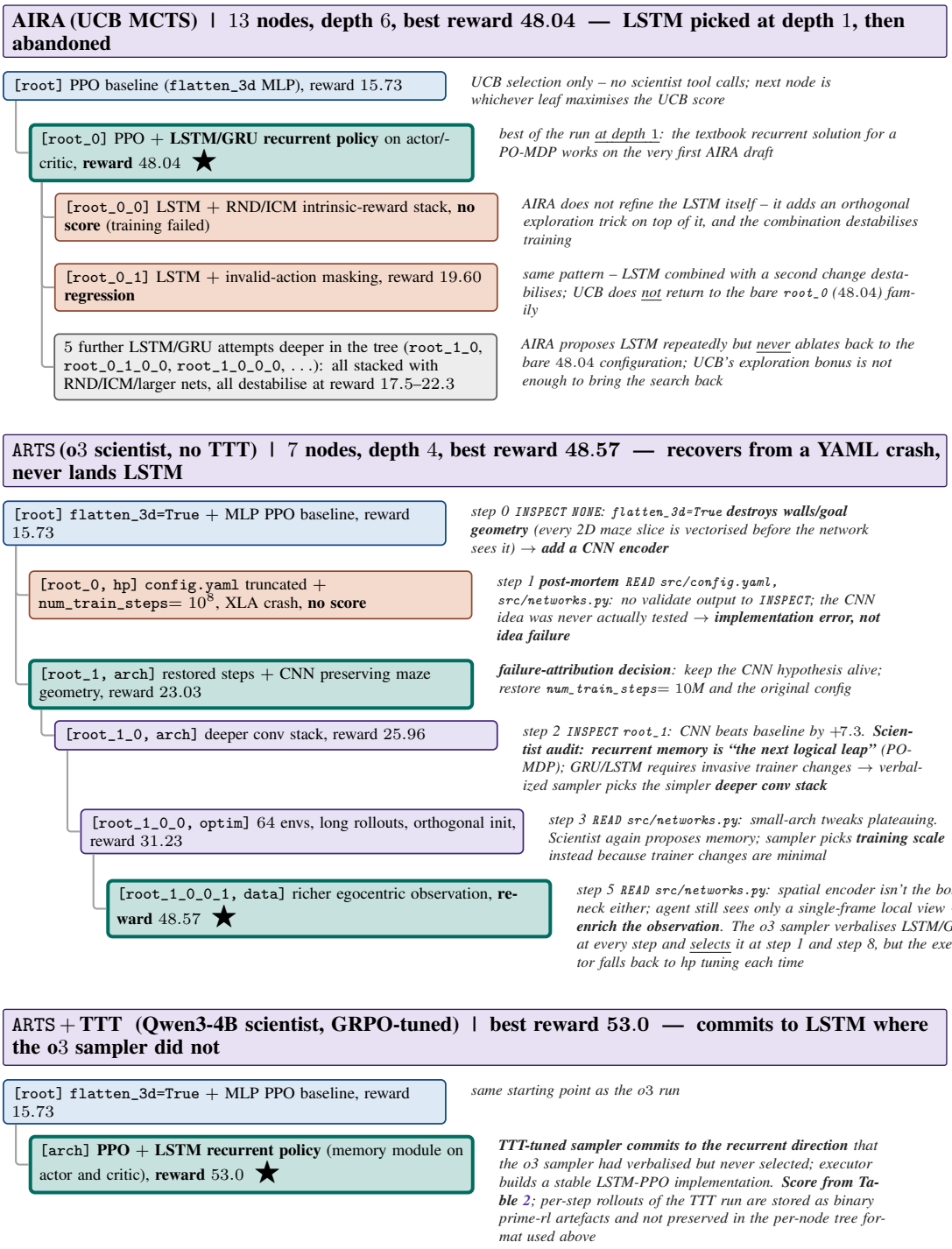


Figure 9: **MetaMaze: AIRA, ARTS-o3, and ARTS+TTT search trees on the same task and budget.** Nodes are coloured blue (baseline), purple (progress), teal (new best, ★), and red (regression / abandoned). AIRA picks LSTM at depth 1 (48.04) and never refines it. ARTS-o3 recovers from a crash via failure attribution and climbs 15.7 → 48.57 on an MLP path, but never lands LSTM. ARTS+TTT commits to the recurrent direction and reaches **53.0**. Full appendix versions in Appendix C.

Table 5: Estimated token and executor-call usage on the 11 MLGym tasks with recovered event counts. Tokens are estimated from saved trajectory text using characters divided by four.

Method	Tokens (M)	SE (M)	Runs	Calls
Linear	1.53	0.26	26	417
AIRA	0.83	0.16	24	244
MLEvolve	2.30	0.50	33	605
ARTS (o3)	0.72	0.15	26	209
ARTS (Qwen)	0.72	0.15	26	209

Table 6: Component ablations for ARTS. Normalized scores use the same baseline and human-best normalization as the main results.

Task	Variant	Removed component	Raw score	Norm.
HMS (KL ↓)	ARTS	None	0.499	0.809
HMS (KL ↓)	A	Audit	0.903	0.470
HMS (KL ↓)	B	Verbalized sampling	0.860	0.506
HMS (KL ↓)	C	Memory	0.762	0.588
HMS (KL ↓)	D	Initial breadth	0.615	0.712
Kuzushiji (F1 ↑)	ARTS	None	0.843	0.887
Kuzushiji (F1 ↑)	A	Audit	0.544	0.573
Kuzushiji (F1 ↑)	B	Verbalized sampling	0.583	0.614
Kuzushiji (F1 ↑)	C	Memory	0.690	0.726
Kuzushiji (F1 ↑)	D	Initial breadth	0.570	0.600

L.3 TTT Reward and Episode Structure

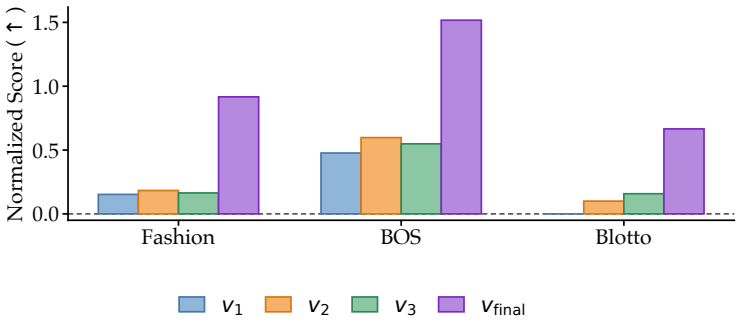


Figure 11: Reward-function ablation for test-time training. Scores are normalized by the task baseline and human-best value.

Table 7: Reward-function ablation. Each task entry reports raw score followed by normalized score in parentheses.

Version	Reward	Fashion MNIST	BOS	Blotto	Mean norm.
v_1	$\mathbf{1}[s_t > s_{\text{base}}]$	0.866 (0.152)	1.330 (0.477)	-0.248 (0.000)	0.210
v_2	$\{-0.5, 0, 0.2, 1\}$ with $+1$ if $s_t \geq 0.80$	0.870 (0.183)	1.408 (0.598)	-0.173 (0.100)	0.294
v_3	$\mathbf{1}[s_t > s_t^\star]$	0.868 (0.164)	1.376 (0.549)	-0.130 (0.158)	0.290
v_{final}	$\{-0.5, -0.2, 0, 1\}$ with $+1$ if $s_t \geq s_{70}(\mathcal{G}_t)$	0.958 (0.917)	2.000 (1.517)	0.250 (0.666)	1.033

Table 8: Episode-structure ablation. Each entry reports raw score followed by normalized score in parentheses.

Task	Tree-per-episode	Single-step GRPO
Fashion MNIST	0.853 (0.04)	0.948 (0.83)
BOS	1.408 (0.60)	1.442 (0.65)
Blotto	-0.173 (0.10)	0.344 (0.79)
Titanic	0.947 (2.83)	1.000 (3.66)
Mean norm.	0.89	1.48

L.4 Diversity After TTT

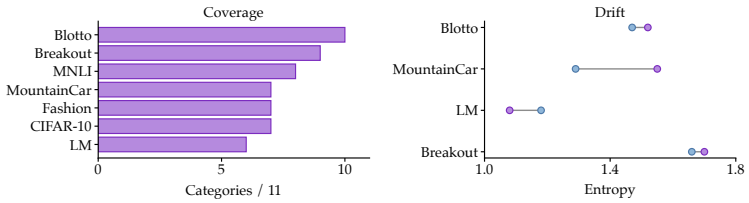


Figure 12: Diversity after TTT. Left: number of strategy categories used out of 11. Right: early (blue) and late (purple) entropy across rollouts.

Table 9: Diversity after TTT. Categories are assigned by keyword matching over proposed strategy text. Entropy is reported for tasks with early/late rollout statistics.

Task	Top categories by share	Used / 11	Early H	Late H
Blotto	Other 37%, feature_eng 36%, data_aug 8%	10	1.47	1.52
Breakout	rl_specific 38%, CNN 19%, MLP 18%	9	1.66	1.70
MNLI	hyperparam 22%, data_aug 21%, rl_specific 19%	8	—	—
MountainCar	feature_eng 47%, MLP 22%, rl_specific 15%	7	1.29	1.55
Fashion MNIST	data_aug 57%, CNN 21%, hyperparam 11%	7	—	—
CIFAR-10	data_aug 55%, hyperparam 30%, CNN 5%	7	—	—
LM	Other 65%, feature_eng 12%, data_aug 10%	6	1.18	1.08