Adaptive Proxy Evaluation for Autonomously Improving ML Agents

Samuel Verboomen [1] **Alesia Ivanova** [2] **Yogendra Manawat** [1] **Prannay Hebbar** [1] **Selvam Palanimalai** [1]
Kunal Bhatia [1] **Vignesh Baskaran** [1]

Abstract

Autonomous ML agents search over many candidate solutions, but full training runs are too expensive to evaluate each one, so agents rely on proxy evaluations: shorter training, smaller data, fewer gradient steps. These proxies make search tractable but are unreliable. A weak proxy promotes candidates that look good early and fail later; a strong proxy is too costly to run at scale; and when each candidate defines its own evaluation, the resulting self-reported metrics make cross-candidate comparison untrustworthy. We address this with an external, adaptive proxy layer that evaluates every candidate under identical conditions and progressively raises fidelity as the search converges, giving fast feedback during exploration and reliable feedback during refinement. We implement the system in MLEvolve, a Monte Carlo Graph Search framework, with no task-specific configuration, pilot runs, or human intervention. On the MLE-bench Ventilator Pressure Prediction task, a single run reaches a state-of-the-art MAE of 0.1354 within a 12-hour budget, outperforming all prior methods.

1. Introduction

Recent large language model agents can autonomously perform machine learning engineering (MLE) tasks: proposing candidate solutions, training and evaluating them, and using results to guide further search. The central challenge is evaluation cost. A single full training run can take hours, so an agent wanting to try hundreds of ideas cannot fully train each one. Similar bottlenecks appear in scientific simulation, hyperparameter optimization, neural architecture search, and agentic coding loops with long-running tests.

[1]Hexo Labs, Palo Alto, California, USA [2]Department of Computer Science, University of Oxford, Oxford, United Kingdom. Correspondence to: Vignesh Baskaran <vignesh@hexo.ai>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

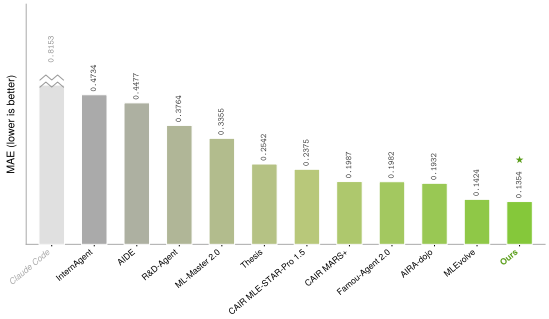

Figure 1. State-of-the-art MAE on MLE-Bench Ventilator Pressure Prediction within a 12-hour budget.

Search systems work around this by using *proxy evaluations*: running each candidate with cheaper settings such as shortened training, subsampled data, or fewer gradient steps. Proxies make search possible but are often unreliable. A proxy that is too weak picks candidates that look good early but fail later; one that is too strong shrinks the number of ideas the agent can try. Reliability also varies across model families, so a short training run may predict final performance well for one architecture and mislead for another. The right fidelity shifts during the search itself, since exploration and refinement have different needs: a proxy fast enough to screen many ideas is rarely faithful enough to choose between the best ones. A further problem arises when each candidate defines its own evaluation procedure, producing uncontrolled comparisons across the search tree and systematic underestimation of compute cost.

We present an adaptive proxy and oversight layer addressing these challenges. The key design choice is to separate code generation from evaluation: the LLM proposes solutions under constraints, and a deterministic harness evaluates them under identical conditions. We implement this inside MLEvolve (Du et al., 2026), a Monte Carlo Graph Search framework that achieves the highest score among open-source frameworks on MLE-bench (Chan et al., 2025) (61.3% medal rate, 12-hour budget). Our contributions are: (i) **bounded proxy evaluation** that scores every candidate within a phase under identical, deterministic conditions; (ii) **trajectory-based scoring** that combines area under the curve, gain, and instability across fixed checkpoints, re-

warding stable learning over noisy endpoint loss; (iii) a **Bradley-Terry arena** that ranks candidates from different model families through pairwise code-only comparisons judged by an LLM, correcting biases the proxy alone introduces; and (iv) an **Overseer Agent** that calibrates initial proxy settings, schedules phase transitions, and manages iteration and time budgets without making ML architecture decisions. On the Ventilator Pressure Prediction task from MLE-bench, a single run within a 12-hour budget and no task-specific tuning achieves a final MAE of 0.1354 on the full dataset, surpassing all previously published methods.

2. Related work

General software engineering agents such as OpenHands (Wang et al., 2025) and SWE-Agent (Yang et al., 2024) rely on deterministic environments where fast unit tests provide immediate correctness signals, and lack mechanisms for managing the stochastic, long-horizon training cycles inherent to ML workflows. Specialized MLE agents frame pipeline optimization as iterative search: AIDE (Jiang et al., 2025) formulates trial-and-error as tree search over code modifications, R&D-Agent (Yang et al., 2025) introduces a Researcher-Developer architecture with parallel exploration, and AutoMLGen (Du et al., 2025) reformulates the search space as a directed graph enabling trajectory reuse via Monte Carlo Graph Search. Building on this line, MLEvolve (Du et al., 2026) achieves the highest score among open-source agents on MLE-bench (Chan et al., 2025) under a 12-hour budget. Across this body of work, evaluation is treated as fixed scaffolding rather than a first-class search component; our contribution lies in this gap. Extended discussion in Appendix A.

3. Method

Unlike previous MLE agentic frameworks, we treat evaluation as a first-class component of the search system. Our approach introduces (i) a *controlled proxy* that evaluates every candidate solution under identical conditions and scores it from its full training trajectory, (ii) an *adaptive schedule* that adjusts the proxy as the search progresses, (iii) a *pairwise ranking arena* that compares candidates from different model families (the distinct initial branches of the search), and (iv) an *Overseer Agent* that manages the schedule without making ML architecture decisions.

3.1. Proxy and evaluation design

Each candidate is an executable Python script exposing a fixed CLI (gradient steps, batch size, learning rate, dataset size). The framework invokes every candidate in a phase with identical arguments under a fixed timeout, so comparisons are controlled rather than self-reported. A single final-loss number is a noisy signal under short training budgets: two candidates with the same endpoint can have very different trajectories, and one still improving at the cutoff would overtake an early-plateauing peer under a longer schedule. We therefore score each run from its full validation trajectory, recorded at fixed checkpoints (e.g., 150, 300, 500 gradient steps), combining three components: (i) an AUC-style term for average trajectory quality, (ii) a gain term for start-to-finish improvement, and (iii) an instability penalty for noise. This rewards candidates that learn stably under a constrained budget and lets slower-converging architectures earn credit for late-stage gains.

3.2. Adaptive proxy

Evaluation needs shift as the search progresses. Early on, a cheap proxy (short training on a small subsample) suffices to filter clearly weak candidates. Later, two problems emerge: performance gaps between top candidates narrow, demanding lower-noise rankings; and short training systematically favors fast-converging architectures over deeper networks or ensembles that would win at full fidelity. Both call for higher fidelity, but applying it from the start would drastically cut the number of ideas explored. We therefore raise proxy fidelity adaptively, in discrete phases triggered by the Overseer (§3.4).

3.3. Bradley-Terry arena

Proxy scores systematically favor fast-converging families: a short training budget penalizes deeper networks or ensembles that would win at full fidelity. We add an independent ranking signal based on pairwise comparisons judged by a language model, aggregated into per-candidate strengths via the Bradley-Terry model. The judge sees only source code and the task description, never proxy scores or metrics; withholding the proxy signal is what lets pairwise judgments correct proxy bias rather than echo it. The judge also flags solutions that game the evaluation protocol (for example, hardcoded budgets or nested loops substituting for gradient steps), and these are penalized regardless of proxy score. Battles are conducted cross-family to prevent any single family from dominating selection through fast early convergence. The one exception is intra-family replay across phases, which keeps rankings continuous when proxy settings change. For each new candidate, opponents are drawn from a bounded pool: the top-k by proxy score and the top-k by current BT strength from each opposing family. After each battle, all strengths are re-estimated globally by maximum likelihood over the full battle history, yielding rankings invariant to arrival order, unlike incremental schemes such as ELO. The composite score used during MCTS expansion combines both signals: $s = \alpha \cdot \tilde{p} + \beta \cdot w_b(n_b) \cdot \tilde{b}$, where $\tilde{p}$ is the min-max normalized proxy score within the current phase, $\tilde{b}$ the normalized log BT strength, and

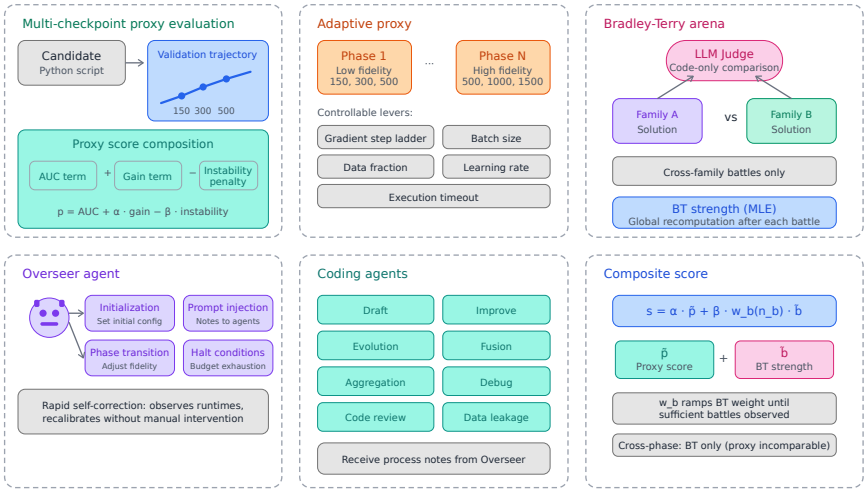

Figure 2. **Method overview.** An *Overseer Agent* runs the search in *phases*. Within a phase, an external harness scores every candidate with the same cheap *proxy* (short training run on a data subsample), ensuring equal footing. In parallel, an LLM judge picks winners in pairwise code comparisons across families, and a Bradley–Terry model converts these into strength scores. Proxy and BT scores combine into a composite score driving MCTS expansion. When the proxy can no longer separate top candidates, the Overseer triggers a *phase transition*, raising data fraction, gradient steps, and timeout to shift from exploration to refinement.

$w_b = \min(1, n_b/n_{\min})$ ramps BT in as enough battles accumulate. Across phase boundaries, proxy scores are incomparable and selection relies on BT alone.

3.4. Overseer Agent

The Overseer acts as a project manager for the search: it controls the adaptive proxy schedule and the time budget, but makes no ML architecture decisions. It has four responsibilities. **(i) Initialization.** The Overseer inspects task metadata and a data sample to propose initial proxy settings (training unit configuration, gradient step ladder, data fraction, batch size, execution timeout), replacing the manual pilot runs that static configurations require. **(ii) Prompt injection.** The Overseer can append process notes to the system prompt of all coding agents. Notes address structural and environmental failures (unavailable imports, output truncation, submission format violations) but never prescribe architectures or hyperparameters. **(iii) Phase transition.** The Overseer monitors wall-clock time and median node duration and may shift the proxy along four levers: the gradient step ladder (checkpoints at which validation is recorded, e.g. $150, 300, 500 \rightarrow 500, 1000, 1500$), batch size, execution timeout, and data fraction. The direction is not always upward: if nodes time out or produce unstable results, the Overseer reduces these settings. A separate execution timeout enforces determinism and prevents jobs from bypassing the agreed gradient-step budget. **(iv) Halt conditions.** The Overseer terminates the run only on unrecoverable structural errors or wall-clock budget exhaustion. A key property is *rapid self-correction*: the system converges to appropriate proxy settings within a few completed evaluations, without manual pilot runs.

3.5. Experimental setup

We evaluate on the Ventilator Pressure Prediction task from MLE-bench, treated as a representative medium-complexity MLE benchmark rather than a target for state-of-the-art performance. The task is time-series regression: predict airway pressure at each timestep from breath-level sensor sequences of ∼80 measurements over ∼6M training samples, scored by mean absolute error (MAE). We use an 80/20 train/validation split and transform the metric as $\text{perf} = 1/(1 + \text{MAE})$ so higher is better. Search progress is measured in *journal steps* (one fully evaluated candidate per step). The run uses a 12-hour budget with Gemini-3-Pro-Preview; the Overseer is invoked every 25 steps with a 75-step minimum between phase changes. Full configuration is in Appendix B.

4. Results

The experiment consists of a single run with no task-specific configuration, no pilot tuning, and no human intervention after initialization (Fig. 3). Within the current run, the system generated 392 evaluated candidates across 6 solution families. The journal comprises 401 nodes in total, all with metrics and BT-arena entries, with an overall buggy node rate of 2.5%. The best discovered solution combines deep LSTMs with EMA weight averaging and Gaussian input noise and achieves **a state-of-the-art MAE of 0.1354** on the Ventilator Pressure Prediction task after full training on

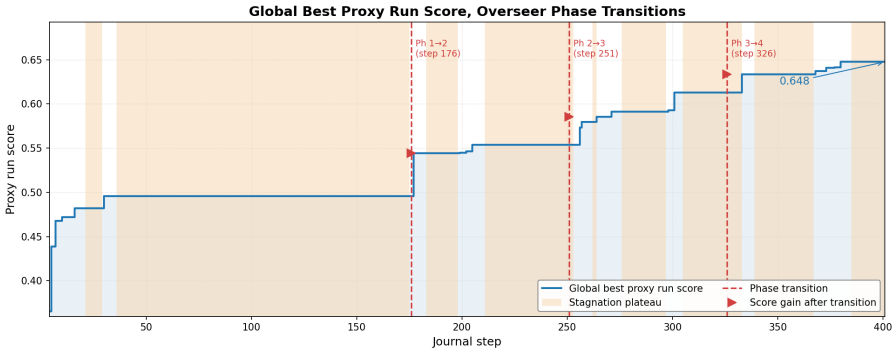

Figure 3. Global best proxy score over 392 nodes (Ventilator Pressure Prediction). Dashed red lines: Overseer phase transitions (steps 176, 251, 326). Orange regions: stagnation plateaus where the proxy fails to separate candidates; red arrows: post-transition gains. Transitions align with plateau boundaries; minor offsets reflect in-flight nodes completing under the previous phase.

the full dataset (Fig. 1). Competing ML frameworks report their best results under 24-hour budgets, except for MLEvolve, R&D-Agent, and InternAgent, which report results after 12-hour runs.

Table 1. Overseer-issued proxy configurations. Gradient steps scale with data fraction to hold local epoch coverage at ∼11.3 sequences. Batch size 64 throughout.

Phase	Step	Grad steps	Data frac.	Trigger band (proxy)	Median duration
1	0	200, 400, 600	0.05	—	25 s
2	176	400, 800, 1200	0.10	0.475–0.495	54 s
3	251	800, 1600, 2400	0.20	0.53–0.55	104 s
4	326	1000, 2000, 3000	0.25	0.59–0.61	156 s

Proxy stagnation and Overseer-driven phase transitions. The Overseer issued three phase transitions over the run, each triggered by a stagnation plateau in which top families had converged to within a narrow proxy band (Table 1). The longest plateau preceded the first transition, spanning 142 evaluations (∼4.5 hours). Each transition produced a visible score jump as higher-fidelity evaluation revealed differences masked by the cheaper proxy, at the cost of a 6× rise in per-node evaluation time from Phase 1 to Phase 4. Alongside fidelity changes, the Overseer revised the suggested search-step budget (500 → 400 → 350) and injected targeted process notes addressing structural failures such as code-truncation in feature engineering blocks, without prescribing architectures or hyperparameters. At step 401 it issued a halt after 836.8 min elapsed, overrunning the 720-minute budget by 16%. See App. E for the full trace.

Table 2. Ablation of proxy-side components; lower MAE better.

Configuration	Test MAE ↓
Full (BT arena + adaptive proxy / Overseer)	**0.1354**
BT arena only, no Overseer (fixed proxy)	0.5980
No BT arena, no Overseer (static baseline)	0.6840

Ablation studies To isolate the contributions of each component, we compare the full system against two ablated variants: (i) a static proxy baseline with no Bradley–Terry arena and no Overseer agent, using fixed proxy settings throughout the run; and (ii) the Bradley–Terry arena without the Overseer agent, which retains the pairwise code-quality comparisons but freezes the proxy configuration (no phase transitions, no adaptive data fraction or the number of gradient steps, no note injection). We used identical compute budget and the medium-sized proxy from Phase 2 in Table 1. Table 2 reports the best proxy run score and the corresponding test MAE achieved by each configuration.

5. Discussion

Our results suggest that reliable autonomous MLE depends as much on *how candidates are evaluated* as on *how they are generated*. The gains come from four design choices: (i) bounded proxy evaluation, where every candidate is judged under identical conditions rather than scoring itself; (ii) trajectory-based scoring, which rewards stable learning under a constrained budget instead of relying on a single noisy endpoint; (iii) cross-family pairwise ranking, so fast-converging architectures do not crowd out slower but ultimately stronger ones; and (iv) an Overseer Agent that adapts proxy fidelity over the run, cheap and broad early, faithful and narrow later, without making ML architecture decisions. Together, these reach state-of-the-art from a single run, cut human configuration time (base setup, evaluation wiring, proxy sizing, bug triage) by $\geq 3\times$, and resolve search-time failures such as code truncation autonomously via note injection. Remaining headroom is reachable through process-level adjustments (more runs, more solution families, larger time and compute budgets) requiring no ML expertise. Future work includes validating the approach across more MLE-bench tasks and extending the Overseer beyond evaluation control to managing exploration–exploitation and the proxy/Bradley–Terry weighting dynamically.

References

Chan, J. S., Chowdhury, N., Jaffe, O., Aung, J., Sherburn, D., Mays, E., Starace, G., Liu, K., Maksin, L., Patwardhan, T., Weng, L., and Madry, A. Mle-bench: Evaluating machine learning agents on machine learning engineering, 2025. URL https://arxiv.org/abs/2410.07095.

Chen, J., Mishra, B. D., Nam, J., Meng, R., Pfister, T., and Yoon, J. Mars: Modular agent with reflective search for automated ai research, 2026. URL https://arxiv.org/abs/2602.02660.

Du, S., Yan, X., Jiang, D., Yuan, J., Hu, Y., Li, X., He, L., Zhang, B., and Bai, L. Automlgen: Navigating fine-grained optimization for coding agents, 2025. URL https://arxiv.org/abs/2510.08511.

Du, S., Zhang, B., Yan, X., and Feng, S. Mlevolve, 2026. URL https://github.com/InternScience/MLEvolve.

Jiang, Z., Schmidt, D., Srikanth, D., Xu, D., Kaplan, I., Jacenko, D., and Wu, Y. Aide: Ai-driven exploration in the space of code, 2025. URL https://arxiv.org/abs/2502.13138.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. Swe-bench: Can language models resolve real-world github issues?, 2024. URL https://arxiv.org/abs/2310.06770.

Li, Z., Zang, Q., Ma, D., Guo, J., Zheng, T., Liu, M., Niu, X., Wang, Y., Yang, J., Liu, J., Zhong, W., Zhou, W., Huang, W., and Zhang, G. Autokaggle: A multi-agent framework for autonomous data science competitions, 2024. URL https://arxiv.org/abs/2410.20424.

Liu, Z., Cai, Y., Zhu, X., Zheng, Y., Chen, R., Wen, Y., Wang, Y., E, W., and Chen, S. Ml-master: Towards ai-for-ai via integration of exploration and reasoning, 2025. URL https://arxiv.org/abs/2506.16499.

Nam, J., Yoon, J., Chen, J., Shin, J., Arık, S. , and Pfister, T. Mle-star: Machine learning engineering agent via search and targeted refinement, 2025. URL https://arxiv.org/abs/2506.15692.

Ou, Y., Luo, Y., Zheng, J., Wei, L., Yu, Z., Qiao, S., Zhang, J., Zheng, D., Mao, Y., Gao, Y., Chen, H., and Zhang, N. Automind: Adaptive knowledgeable agent for automated data science, 2025. URL https://arxiv.org/abs/2506.10974.

Team, I., Zhang, B., Feng, S., Yan, X., Yuan, J., Ma, R., Hu, Y., Yu, Z., He, X., Huang, S., Hou, S., Nie, Z., Wang, Z., Liu, J., Peng, T., Ye, P., Zhou, D., Zhang, S., Wang, X., Zhang, Y., Li, M., Tu, Z., Yue, X., Ouyang, W., Zhou, B., and Bai, L. Internagent: When agent becomes the scientist – building closed-loop system from hypothesis to verification, 2025. URL https://arxiv.org/abs/2505.16938.

Toledo, E., Hambardzumyan, K., Josifoski, M., Hazra, R., Baldwin, N., Audran-Reiss, A., Kuchnik, M., Magka, D., Jiang, M., Lupidi, A. M., Lupu, A., Raileanu, R., Niu, K., Shavrina, T., Gagnon-Audet, J.-C., Shvartsman, M., Sodhani, S., Miller, A. H., Charnalia, A., Dunfield, D., Wu, C.-J., Stenetorp, P., Cancedda, N., Foerster, J. N., and Bachrach, Y. Ai research agents for machine learning: Search, exploration, and generalization in mle-bench, 2025. URL https://arxiv.org/abs/2507.02554.

Wang, X., Li, B., Song, Y., Xu, F. F., Tang, X., Zhuge, M., Pan, J., Song, Y., Li, B., Singh, J., Tran, H. H., Li, F., Ma, R., Zheng, M., Qian, B., Shao, Y., Muennighoff, N., Zhang, Y., Hui, B., Lin, J., Brennan, R., Peng, H., Ji, H., and Neubig, G. Openhands: An open platform for ai software developers as generalist agents, 2025. URL https://arxiv.org/abs/2407.16741.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yang, X., Yang, X., Fang, S., Zhang, Y., Wang, J., Xian, B., Li, Q., Li, J., Xu, M., Li, Y., Pan, H., Zhang, Y., Liu, W., Shen, Y., Chen, W., and Bian, J. Rd-agent: An llm-agent framework towards autonomous data science, 2025. URL https://arxiv.org/abs/2505.14738.

A. Additional related work

Autonomous software engineering agents. General software engineering agents are designed to automate complex coding tasks, resolve bugs, and maintain codebases. Frameworks such as OpenHands (Wang et al., 2025) provide flexible architectures that combine large language model reasoning with general tool execution. SWE-Agent (Yang et al., 2024) demonstrates that custom Agent-Computer Interfaces significantly improve codebase navigation and file editing. While these systems achieve strong results on standardized coding evaluations like SWE-bench (Jimenez et al., 2024), their success relies heavily on deterministic environments where fast unit tests provide immediate correctness signals.They generally lack the mechanisms required to manage the stochastic, long-horizon, and computationally expensive training cycles inherent to machine learning workflows.

Specialized machine learning agents. Specialized machine learning agents frame pipeline optimization as an iterative search problem. AIDE (Jiang et al., 2025), a foundational framework, formulates trial-and-error as tree search over candidate code modifications. AutoKaggle (Li et al., 2024) distributes workflow stages across specialized collaborative agents, while R&D-Agent (Yang et al., 2025) introduces a dual-agent Researcher-Developer architecture with mergeable parallel exploration traces. InternAgent (Team et al., 2025) generalizes multi-agent coordination to broader scientific research through a closed-loop hypothesis-to-verification pipeline.

Several works enrich agents with external knowledge. AutoMind (Ou et al., 2025) grounds agentic tree search in a curated knowledge base from top Kaggle solutions and research papers. MLE-STAR (Nam et al., 2025) retrieves model architectures from the web and performs targeted refinement guided by ablation of individual code blocks. ML-Master (Liu et al., 2025) transfers insights across parallel trajectories through a selectively scoped memory mechanism.

AIRA (Toledo et al., 2025) demonstrates that agent performance depends on the joint design of search strategies and code modification operators, systematically comparing greedy, MCTS, and evolutionary approaches. CAIR MARS (Chen et al., 2026) extends this with budget-aware MCTS planning, modular code construction, and comparative reflective memory for cross-branch lesson transfer.

AutoMLGen (Du et al., 2025) reformulates the search space as a directed graph, enabling trajectory reuse and multi-solution fusion through Monte Carlo Graph Search combined with a curated domain knowledge base. Building on this foundation, MLEvolve (Du et al., 2026) further advances graph-based agentic search by introducing improved collaboration mechanisms and targeted code refinement. Despite operating under a 12-hour budget, it achieves the highest score among open-source agents on MLE-bench (Chan et al., 2025) and near-leading results among all systems.

B. Configuration touchpoints

The values below are the effective configuration for the ventilator pressure run reported in the Results section (a single run with ∼12-hour budget; the nominal wall-clock budget of 720 min was exceeded by 116.8 min before the Overseer issued a `halt_run`).

All values are run-specific instantiations of configurable system parameters and may vary across tasks, budgets, and runtime conditions.

Models. All code generation, feedback, and planning use Gemini-3-Pro-Preview. The Overseer full-assessment calls also use Gemini-3-Pro-Preview. Bradley–Terry battles use Gemini-2.5-Pro.

Run budget.

Key	Value
`time_limit` (wall-clock)	720 min (12 h) target
`steps` (soft MCTS step cap)	600
`initial_drafts`	3
`num_drafts` (max families)	6
`parallel_search_num`	3

Overseer schedule.

Key	Value
`overseer_interval`	25 (review every 25 completed steps)
`overseer_cooldown`	75 (note-only steps after a config change)
`overseer_min_transition_delta`	0.1 (veto transition if all numeric deltas ¡ 10 %)
`overseer_init_enabled`	true (Phase 1 proxy set by Overseer init)
`overseer_init_phase1_duration_min`	0.5 (target ∼0.5 min/node for Phase 1)
`overseer_timeout_margin_perc`	0.30 (automatic ×1.3 margin on `exec_timeout`)

Phase 1 proxy (initial calibration). Initial proxy configuration proposed by the Overseer during initialization for this run, based on task characteristics and target per-node runtime.

Key	Value
`eval_fraction`	0.05 (∼3,395 training sequences)
`--batch-size`	64
`--eval-grad-steps`	[200, 400, 600]
`--lr`	0.001
`exec_timeout` (raw)	360 s (468 s after ×1.3 margin)

Bradley–Terry arena and composite score.

Key	Value
`cross_family_only`	true
`bt_cross_family_n_best_proxy_run`	1 (top-1 by proxy score per family)
`bt_cross_family_n_best_bt`	1 (top-1 by BT strength per family)
`bt_n_min`	10 (tunable hyperparameter, controls number of battles before BT weight ramps to full β)
`score_alpha` (proxy weight)	0.50
`score_beta` (BT weight)	0.50

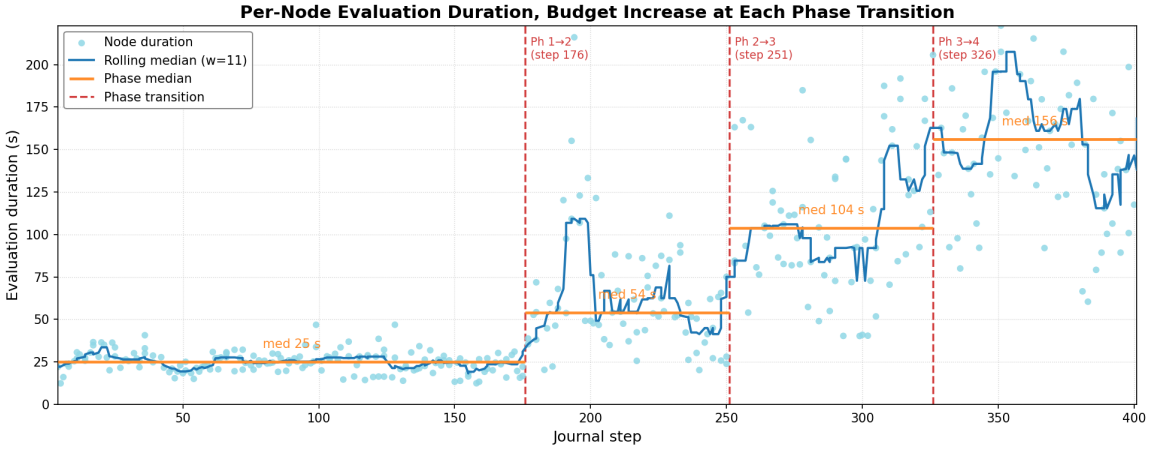

Figure 4. Per-node evaluation time across journal steps (scatter) with rolling median (blue). Orange segments denote per-phase medians. Phase transitions induce step increases in cost via higher proxy evaluation budgets. Phase medians: ∼25 s, 54 s, 104 s, and 156 s; variance increases in later phases.

C. Per-family dynamics

Table 3 reports per-family best proxy scores and peak Bradley-Terry strengths. Family 3 dominates both signals: its best-proxy candidate, at journal step 380, produced by the *improve* operator, following the Phase 3→4 fidelity increase, reaches proxy 0.6477 (Fig. 5), and a separate Family 3 candidate attains the highest Bradley–Terry strength of 19.07 (Fig. 6). The leading architectures in this family share a common pattern noted by the Overseer during Phase 3: deep LSTMs with EMA weight averaging and Gaussian input noise. Family 5 reaches the second-highest proxy (0.6003) via *evolution*;

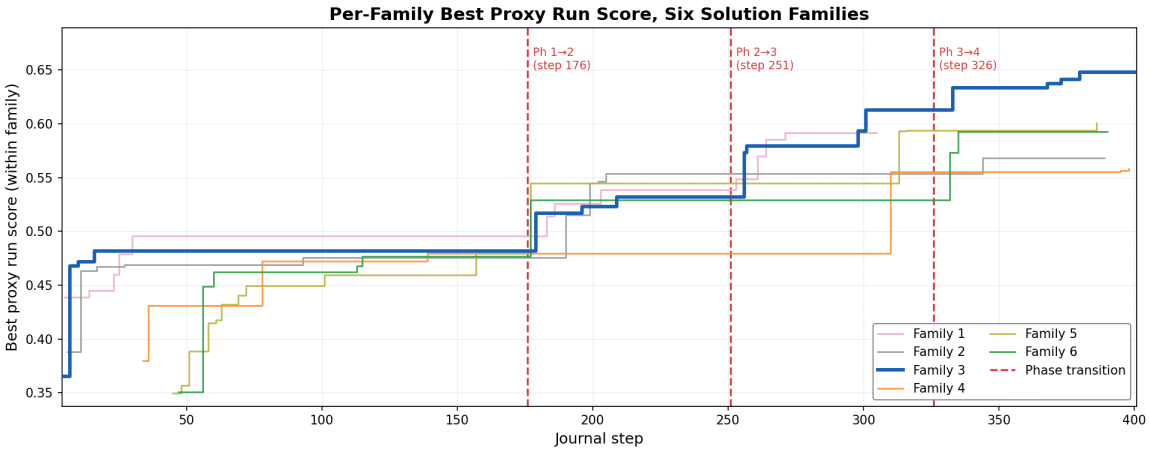

Figure 5. Best proxy score per family over journal steps. Family 3 (blue, emphasised) tracks the pack through Phases 1 and 2, begins to pull ahead during Phase 3, and separates decisively after the Phase 3→4 transition (step 326), reaching a peak of 0.6477. The remaining five families finish within a narrow band between 0.558 and 0.600. Top nodes are produced by *improve* in five of six families, with Family 5 topped by *evolution*, indicating that diversity is maintained across families despite Family 3's dominance on the global best.

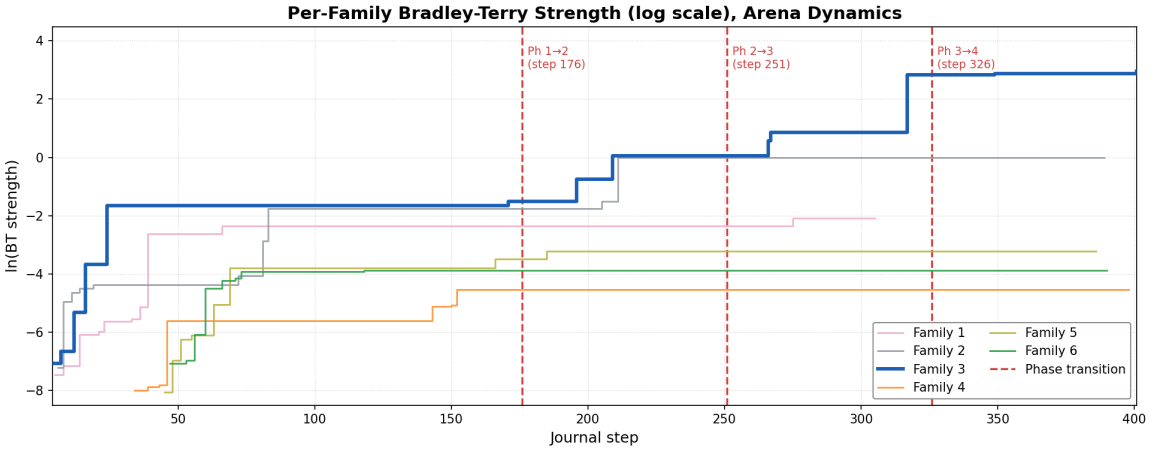

Figure 6. Per-family Bradley–Terry (BT) strength over journal steps (natural-log scale; ln(BT) plotted on a linear axis). Family 3 (blue, emphasised) separates sharply, ending at BT ≈ 19.07 — more than an order of magnitude above every other family. Family 2 attains a moderate final BT (≈ 0.99) despite one of the lowest proxy scores, while Families 1, 4, 5, and 6 remain weak (BT ≤ 0.13). This proxy–BT divergence highlights the complementarity of the two signals: BT captures code quality and generalization not fully reflected by short proxy trajectories.

Families 6, 1, 2, and 4 follow with top nodes via *improve*, all within a narrow 0.558–0.593 band. On the BT axis, Family 3 separates by more than an order of magnitude from every other family: its closest BT competitor is Family 2 at 0.99, despite Family 2 ranking fifth on the proxy. This proxy–BT divergence illustrates the complementarity of the two ranking signals: within Family 3, the top-proxy node and the top-BT node are distinct, so no single scalar of either kind identifies a Pareto-optimal candidate on its own. Family 2 further illustrates the point in the opposite direction, its BT rises above 0.99 while its best proxy stays the second-lowest in the pool, suggesting code quality not reflected in short proxy trajectories. The composite score ($\alpha = \beta = 0.5$) correctly elevates Family 3 as the overall leader.

D. Empirical validation of the Bradley–Terry cross-family signal

On 45 nodes from 6 solution families, we measure pairwise ranking agreement with the final full-training metric (perf@150, where perf $= 1/(1 + \text{MAE})$), stratified by intra- and inter-family pairs (Fig. 7). Within families, proxy scores slightly

Table 3. Final metrics by solution family. Family 3 dominates on both axes, reaching the highest proxy score and a BT strength more than an order of magnitude above every other family. The remaining five families cluster within a narrow proxy band (0.558–0.600) but rank inconsistently on BT, reflecting the independence of the two signals.

Family	Proxy score	BT strength	Best agent
1	0.5915	0.124	improve
2	0.5682	0.990	improve
3	**0.6477**	**19.07**	improve
4	0.5582	0.011	improve
5	0.6003	0.040	evolution
6	0.5925	0.021	improve

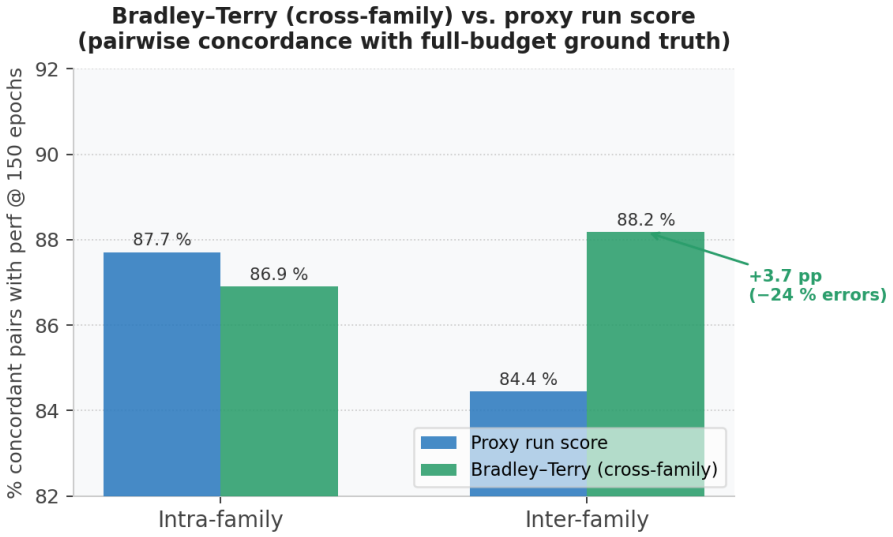

Figure 7. Pairwise agreement with perf@150 (full-training ground-truth performance), split by intra- and inter-family pairs. Cross-family BT (green) exceeds proxy scores (blue) by 3.7% on inter-family pairs (24% fewer ranking errors), while within-family performance differs by $\leq 1\%$.

improve agreement (87.7% vs. 86.9%) while requiring less computational budget than a full Bradley–Terry tournament and being more responsive to subtle code changes whose full impact an LLM judge may not fully capture. Across families, BT improves agreement by 3.7% (88.2% vs. 84.4%), yielding 24% fewer ranking errors. Beyond agreement, BT enables (i) cross-phase comparability under non-aligned proxies, (ii) robustness to budget exploitation, and (iii) retention of structurally diverse but temporarily underperforming families.

E. Full Overseer reasoning trace, Ventilator Pressure run

This appendix records every Overseer call made during the run reported in the Results section. The Overseer fires once at initialisation and then every 25 journal steps, in either `full` mode (may issue a phase transition or adjust `suggested_max_steps`) or `note` mode (may update the prompt note but cannot change the proxy configuration). For each call we list the governing step range, the decision type, whether a configuration change was emitted, the stagnation signal that triggered the call, the agent's verbatim reasoning (quoted from the dashboard log), and the configuration delta (if any).

Steps 0–25 — pre-run calibration. **Decision type:** `init` | **Config change:** yes | **Stagnation trigger:** pre-run calibration (no stagnation).
Reasoning: "The task is sequence-based (80 rows per breath), so models will likely process full breaths (LSTMs, GRUs, Transformers). With a 12-hour budget, we want to maximize architecture exploration early on. A 5% data fraction yields ∼3,395 training sequences. Using a batch size of 64, 600 gradient steps provides roughly 11.3 local epochs. At ∼0.1s per step for RNNs, a node will take about 1–2 minutes, allowing thousands of iterations to establish a robust ranking signal

before any heavy exploitation."
Phase 1 configuration: `--batch-size 64 --eval-grad-steps 200,400,600 --lr 0.001`; `eval_fraction`=0.05; `exec_timeout`=360 s; `local_epoch`=11.3.
Initial note (targets: `draft`, `improve`): "Focus on sequence models (LSTM, GRU, 1D CNN, Transformer) since the data is grouped by breath_id with 80 time steps each. Ensure your PyTorch Datasets yield sequences of shape (80, num_features)."

Steps 26–50. Decision type: `note` | **Config change:** no | **Stagnation trigger:** routine note-mode check.
Reasoning: "The agents have successfully adopted sequence models (LSTMs) and are correctly reshaping the data to (80, num_features) as instructed. They are also correctly masking the loss for the inspiratory phase. The current note remains relevant for any new drafts."
Delta: none.

Steps 51–75. Decision type: `note` | **Config change:** yes (note only) | **Stagnation trigger:** routine note-mode check (code-truncation bugs observed).
Reasoning: "Observed multiple draft nodes failing due to code truncation (scripts ending abruptly, e.g., at 'n_breaths = len'). The agents have successfully adopted sequence models and correct reshaping, so I am updating the note to advise them to keep initial drafts concise to avoid output token limits."
Note update (targets: `draft`, `improve`, `evolution`): "Sequence models and (80, num_features) reshaping are working well. However, multiple draft nodes are failing due to code truncation (scripts ending abruptly). Keep initial draft implementations concise with a minimal feature set to avoid output token limits. Add complex feature engineering later during the improve phase."

Steps 76–100. Decision type: `full` | **Config change:** no | **Stagnation trigger:** 9-step stagnation detected (criteria for phase transition not met).
Reasoning: "The search is still in the early exploration phase (18% of wall-clock budget elapsed). The current proxy is cheap (26s/node) and provides a comfortable throughput of $\sim$1362 possible nodes. Although there is a 9-step stagnation, multiple families are actively developing and scores are reasonably spread (0.43 to 0.49). Strict conditions for increasing proxy cost are not met since we are not in late-stage exploitation and there is no evidence of extreme ranking noise. We will maintain the current configuration to maximize architecture exploration."
Delta: none (Phase 1 held).

Steps 101–125. Decision type: `full` | **Config change:** yes (note + `suggested_max_steps`) | **Stagnation trigger:** 9-step stagnation (phase-transition criteria still not met).
Reasoning: "The search is at 25% of the wall-clock budget with 6 active families exploring diverse sequence models. The current proxy (eval_fraction=0.05, max_steps=600) yields a median execution time of 25s and $\sim$14.1 local epochs, providing a fast and informative ranking signal. Although there is a 9-step stagnation, we are still in the early exploration phase, so increasing proxy cost is not justified under STRICT CONDITIONS. I will maintain the current proxy to maximize throughput. I am updating the suggested_max_steps to 400 to reflect the empirical wall-clock pace ($\sim$1.8 min/step) and refining the overseer note to encourage architectural diversity while keeping feature engineering concise to prevent code truncation."
Delta: `suggested_max_steps`: $500 \rightarrow 400$; proxy configuration unchanged.
Note update (targets: `draft`, `improve`, `evolution`, `fusion`): "Draft nodes previously failed due to code truncation from massive feature engineering blocks. Keep feature engineering concise and vectorized. Focus on architectural improvements (e.g., skip connections, multi-scale convolutions, auxiliary losses) which are showing strong results."

Steps 126–150. Decision type: `note` | **Config change:** no | **Stagnation trigger:** routine note-mode check.
Reasoning: "The current note effectively addresses the code truncation issues by encouraging concise, vectorized feature engineering and focusing on architectural improvements. The agents are successfully exploring diverse sequence models (LSTMs, Conv1D, Attention) and performance is steadily improving. No new structural issues have emerged that require a note update."
Delta: none.

Steps 151–175. Decision type: `note` | **Config change:** no | **Stagnation trigger:** routine note-mode check.
Reasoning: "The current note effectively addresses the previous code truncation issues and correctly guides agents toward concise, vectorized feature engineering and architectural improvements. The agents are successfully implementing these suggestions (e.g., ResBiLSTM, 1D Convs, skip connections) and the bug rate remains low."
Delta: none.

Steps 176–200 — Phase 1 → 2 transition. Decision type: `full` | **Config change:** yes (phase transition) | **Stagnation trigger:** 9-step global stagnation; 6 families clustered in 0.475–0.495 MAE.
Reasoning: "The search has reached 45% of the wall-clock budget with 6 active families clustered tightly between 0.475 and 0.495 MAE. A 9-step global stagnation indicates that the current proxy (eval_fraction=0.05, max_steps=600) may lack the resolution to separate these highly competitive sequence models. To provide a more robust ranking signal, we are doubling the data diversity by increasing eval_fraction to 0.10 and compensating by doubling max eval-grad-steps to 1200. This will increase node execution time to ∼45s, which is well within our budget, allowing for over 500 more evaluations in the remaining 392 minutes."
Phase 2 configuration delta: `--eval-grad-steps` $200, 400, 600 \rightarrow 400, 800, 1200$; `eval_fraction`: $0.05 \rightarrow 0.10$; `local_epoch`: $14.1 \rightarrow 11.3$; `suggested_max_steps`: $400 \rightarrow 500$.
Local-epoch rationale (verbatim): "We are increasing eval_fraction from 0.05 to 0.10 to double the data diversity and better separate the closely clustered families. To support this, max eval-grad-steps is increased from 600 to 1200. This shifts the previous local_epoch of 14.1 down slightly to 11.3. This intentional reduction in coverage depth trades marginal training convergence for significantly broader data exposure, which is acceptable for ranking quality. The new proxy will take ∼45s per node, allowing over 500 evaluations in the remaining 392 minutes of wall-clock time."
Note update (targets: `improve`, `evolution`, `fusion`): "Families are closely clustered around 0.47–0.49 MAE. To break the 9-step stagnation, focus on architectural refinements (e.g., attention mechanisms, deeper residual connections) rather than just adding more lag/lead features, which recently caused KeyErrors in Family 3."

Steps 201–225. Decision type: `note` | **Config change:** yes (note only) | **Stagnation trigger:** routine note-mode check (stagnation broken, scores pushed to ∼0.54).
Reasoning: "The previous note referenced outdated proxy scores (0.47–0.49) and a stagnation that has since been broken. Agents have successfully pushed scores to ∼0.54 using hybrid CNN-RNN-Attention architectures and auxiliary losses. I am updating the note to reinforce these successful strategies and encourage further refinement and fusion of these components."
Note update (targets: `improve`, `evolution`, `fusion`): "Scores have improved to ∼0.54 proxy score. Successful architectures combine CNN stems, deep BiLSTMs/BiGRUs with residual connections, and Attention/Transformer layers. Auxiliary losses (e.g., temporal derivatives, expiratory phase weighting) are also proving effective. Continue refining these hybrid architectures and tuning auxiliary loss weights."

Steps 226–250. Decision type: `note` | **Config change:** yes (note only) | **Stagnation trigger:** routine note-mode check (scores up to ∼0.55).
Reasoning: "Scores have improved to ∼0.55. Agents have successfully introduced Regression-via-Classification (Two-Hot Encoding), mixed precision with gradient accumulation, and explicit expiration timing features. Updating the note to encourage combining these new techniques with the existing hybrid architectures."
Note update (targets: `improve`, `evolution`, `fusion`): "Scores have reached ∼0.55. Top models use deep residual BiLSTMs/BiGRUs with CNN stems. Recent successful innovations include Regression-via-Classification (Two-Hot Encoding), mixed precision with gradient accumulation, and explicit expiration timing features. Combine these techniques to push performance further."

Steps 251–275 — Phase 2 → 3 transition. Decision type: `full` | **Config change:** yes (phase transition) | **Stagnation trigger:** late-stage exploitation, top families clustered in 0.53–0.55 MAE.
Reasoning: "The search is at 68% of the wall-clock budget, entering the late-stage exploitation phase. Top families are tightly clustered around 0.53–0.55 MAE, making it difficult for the current proxy to reliably separate the best candidates. This meets the strict conditions for increasing proxy cost. By doubling the eval_fraction to 0.20 and proportionally increasing eval-grad-steps to 2400, we double the data diversity while maintaining the same local epoch coverage. The estimated node time will increase to ∼100s, which still comfortably allows for ∼138 more evaluations in the remaining 230 minutes, preserving sufficient throughput for final convergence and fusion."
Phase 3 configuration delta: `--eval-grad-steps` $400, 800, 1200 \rightarrow 800, 1600, 2400$; `eval_fraction`: $0.10 \rightarrow$

0.20; `exec_timeout`: 360 → 600 s; `local_epoch` held at 11.3; `suggested_max_steps`: 500 → 400.
Local-epoch rationale (verbatim): "We are increasing eval_fraction from 0.10 to 0.20 to provide more data diversity for better ranking resolution among the tightly clustered top models. To keep the local_epoch flat at ∼11.3 sequences (previous local_epoch was also 11.3), we are doubling max eval-grad-steps from 1200 to 2400. The estimated node time will increase to ∼100s, which still allows for ∼138 nodes in the remaining 230 minutes, preserving sufficient throughput for the late-stage search."
Note update (targets: `improve`, `evolution`, `fusion`): "We are in the late exploitation phase. Top models are tightly clustered around 0.53–0.55 MAE. Focus on fusing the best techniques: Regression-via-Classification (Two-Hot Encoding), mixed precision with gradient accumulation, explicit expiration timing features, and multi-branch CNN-RNN architectures. Ensure robust residual connections to prevent gradient decay."

Steps 276–300. Decision type: `note` | **Config change:** yes (note only) | **Stagnation trigger:** routine note-mode check (top scores at ∼0.59).
Reasoning: "Scores have improved to ∼0.59, making the previous note's reference to 0.53–0.55 outdated. Updating the note to reflect the new state-of-the-art techniques (DeepResidualBiGRU, target normalization, pure inspiratory MAE loss) to guide the final exploitation phase."
Note update (targets: `improve`, `evolution`, `fusion`): "We are in the final exploitation phase. Top models have reached ∼0.58–0.59 proxy score. The most successful architectures use DeepResidualBiGRU or BiLSTM (4+ layers) with skip connections, target normalization, and pure inspiratory MAE loss. Fast vectorized feature engineering is crucial for efficiency. Focus on refining and fusing these top techniques."

Steps 301–325. Decision type: `note` | **Config change:** yes (note only) | **Stagnation trigger:** routine note-mode check (re-occurrence of code-truncation bugs at steps 263, 281).
Reasoning: "The search is in the final exploitation phase with top scores reaching ∼0.59. However, recent buggy nodes (e.g., steps 263, 281) failed due to code truncation caused by excessively long feature engineering blocks. I am updating the note to warn agents to keep feature engineering concise while continuing to refine the top-performing DeepResidualBiGRU/BiLSTM architectures."
Note update (targets: `improve`, `evolution`, `fusion`): "Final exploitation phase. Top models reach ∼0.59 proxy score using DeepResidualBiGRU/BiLSTM (4+ layers), skip connections, target normalization, and pure inspiratory MAE loss. WARNING: Excessively long feature engineering blocks are causing code truncation and node failures. Keep feature engineering concise and modular. Focus on refining and fusing these top techniques."

Steps 326–350 — Phase 3 → 4 transition. Decision type: `full` | **Config change:** yes (phase transition) | **Stagnation trigger:** 9-step stagnation in final 8% of budget; top families 0.59–0.61, Family 3 leading.
Reasoning: "The search is in the final 8% of the wall-clock budget with a 9-step stagnation. Top families are closely clustered between 0.59 and 0.61 proxy score, with Family 3 leading via EMA and deep LSTMs. To provide a more rigorous signal for final tie-breaking and aggregation, we will increase the proxy size slightly. By raising eval_fraction to 0.25 and max eval-grad-steps to 3000, we maintain the same local_epoch while increasing data diversity. The remaining time allows for ∼21 nodes at the new cost, which is sufficient for final fusion and aggregation."
Phase 4 configuration delta: `--eval-grad-steps` 800, 1600, 2400 → 1000, 2000, 3000; `eval_fraction`: 0.20 → 0.25; `exec_timeout` held at 600 s; `local_epoch` held at 11.3; `suggested_max_steps`: 400 → 350.
Note update (targets: `improve`, `evolution`, `fusion`, `aggregation`): "Final exploitation phase. Top score is ∼0.61 (Family 3) using EMA, Gaussian noise, and deep LSTMs. Families 1 and 5 are close at ∼0.59. Focus on fusing these top approaches (e.g., EMA + deep residual LSTMs). WARNING: Excessively long feature engineering blocks are still causing code truncation and node failures (e.g., steps 263, 281). Keep feature engineering strictly concise and modular."

Steps 351–375. Decision type: `note` | **Config change:** no | **Stagnation trigger:** routine note-mode check (budget 719.6 / 720 min reached).
Reasoning: "The experiment has reached the end of its wall-clock budget (719.6 / 720 minutes). The current note is still relevant for any final nodes that might execute, and no further guidance is necessary."
Delta: none.

Steps 376–400. Decision type: `note` | **Config change:** no | **Stagnation trigger:** routine note-mode check (budget exceeded, 779.2 / 720 min).

Reasoning: "The experiment has exceeded its wall-clock budget (779.2min / 720.0min). The current note is still relevant for any final nodes that might execute, and no further guidance is necessary."
Delta: none.

Step 401 — halt. **Decision type:** `full` | **Config change:** yes (halt request) | **Stagnation trigger:** wall-clock budget fully exhausted (836.8 / 720 min).
Reasoning: "The experiment has exceeded its wall-clock budget of 720 minutes (currently at 836.8 minutes elapsed). The search has completed 401 nodes, with top families reaching a proxy score of $\sim$0.647 using deep sequence models and EMA. Since the time budget is fully exhausted, continuing the run is wasteful. I am issuing a halt request to terminate the experiment."
Delta: `halt_run=true`; `halt_reason=time_budget_exhausted`.