LEAF: Learning to Explore and Accumulate Feedback for Agentic Machine Learning Engineering

Anonymous ACL submission

Abstract

Language-model agents are increasingly used to automate machine learning engineering, but existing frameworks often converge to semantically redundant solutions and fail to preserve reusable debugging and optimization knowledge across long search horizons. We introduce LEAF, a search-and-memory agentic framework for autonomous ML engineering. LEAF augments MCTS-style search with an LLM-adapted *Fireworks Algorithm* that allocates diverse code-generation "sparks" around promising nodes and triggers high-amplitude explosions to escape stagnation. It further combines cross-branch genetic merging with a dual-layer memory system: global semantic deduplication filters redundant code variants, while reflective short-term buffers distill recent improvement and debugging experience into persistent guidance. On MLE-bench, LEAF achieves a 66.7% overall medal rate, outperforming strong public leaderboard agents and controlled ablations. These results show that explicit diversity control and structured memory are critical for robust long-horizon agentic ML optimization.

1 Introduction

Agentic workflows have emerged as a transformative paradigm for automating complex machine learning pipelines, attracting intense research interest across academia and industry worldwide. From autonomous code generation (Li et al., 2023, 2022) to end-to-end scientific discovery, agent-based systems are increasingly deployed to navigate large, ill-structured solution spaces with minimal human supervision. Recent ML-oriented agents, including AIDE (Jiang et al., 2025), MLE-volve (Du et al., 2025; Feng et al., 2026), Pi-Evolve (Botla et al., 2025), and ML-Master (Liu et al., 2025; Zhu et al., 2026), together with benchmarks such as MLAgentBench (Huang et al., 2024) and MLE-bench (Chan et al., 2024), have shown that language-model agents can iteratively plan, implement, execute, debug, and refine machine learning pipelines under realistic environment feedback. These results suggest that agentic workflows can solve non-trivial end-to-end ML engineering tasks and accelerate the development of competitive modeling solutions.

Despite this momentum, prevailing agentic frameworks remain fundamentally constrained in two critical respects. First, many systems rely on vanilla Monte Carlo Tree Search (MCTS) (Coulom, 2006; Kocsis and Szepesvári, 2006) or simple single-path rollout strategies to guide solution search. Although UCT provides an explicit exploration term, this mechanism is hard to make effective in agentic ML programming: LLM-generated child nodes are often semantically correlated rather than independent actions, as syntactically different edits may still instantiate the same feature pipeline, model family, or tuning heuristic. Consequently, expanding more nodes does not necessarily expand the conceptual coverage of the search space, weakening the practical relevance of MCTS-style exploration guarantees. These systems therefore tend to revisit structurally similar high-reward trajectories and converge prematurely around local optima. Second, memory management in existing workflows is largely rudimentary: agents either maintain a flat context window that indiscriminately accumulates past interactions, or discard historical information entirely between episodes (Zhang et al., 2024; Shi et al., 2025). This absence of principled, structured memory design forces agents to repeatedly rediscover previously resolved bugs, overlook cross-branch insights, and fail to consolidate the incremental knowledge needed for sustained improvement.

To address these limitations, we introduce **LEAF** (**L**earning to **E**xplore and **A**ccumulate **F**eedback), a novel agentic ML workflow that

fundamentally rethinks both search strategy and memory architecture. At the search level, LEAF augments the standard MCTS backbone with a Fireworks Algorithm (Tan and Zhu, 2010) that triggers adaptive, amplitude-controlled explosions around promising nodes, dramatically widening exploration coverage while preserving exploitation efficiency; a genetic crossover mechanism further fuses high-quality ideas across distinct search branches, enabling the agent to synthesize solutions that no single trajectory would discover in isolation. At the memory level, LEAF introduces a dual-stage memory management module (Packer et al., 2023; Park et al., 2023) comprising a *Global Memory* for long-horizon semantic routing and deduplication across all branches, and a *Buffer Memory* implementing reflective sliding windows that distill actionable heuristics from recent execution feedback. Together, these innovations equip LEAF to generate more effective, diverse solutions and to accumulate structured knowledge that compounds across the search horizon.

Our contributions are threefold:

- We propose an LLM-adapted Fireworks search mechanism that maps explosion amplitude and spark allocation to code-generation diversity, enabling both local refinement and stagnation escape within an MCTS-style agentic workflow.

- We introduce a structured memory architecture that combines global semantic deduplication with short-term reflective buffers, reducing redundant solution cycling while preserving reusable debugging and optimization knowledge.

- We validate LEAF on MLE-bench and controlled ablations, showing state-of-the-art overall medal rate and demonstrating the separate benefits of memory control and Fireworks-based exploration.

2 Preliminary

2.1 Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search (Coulom, 2006; Kocsis and Szepesvári, 2006) iteratively builds a decision tree through four canonical phases—Selection, Expansion, Simulation, and Backpropagation—guiding node selection via the UCT criterion (Kocsis and Szepesvári, 2006):

$$\mathrm{UCT}(v) = \frac{Q(v)}{N(v)} + c\sqrt{\frac{\ln N(\mathrm{parent}(v))}{N(v)}}, \quad (1)$$

where $Q(v)$ is the cumulative reward of node v, $N(v)$ its visit count, and $c > 0$ a constant. In agentic ML workflows, nodes are code snapshots and rewards are task-specific evaluation metrics; standard MCTS faces a key practical limitation: LLM-generated edits are often semantically correlated, so the tree expands syntactically without covering new conceptual regions and converges around homogeneous local optima.

2.2 Fireworks Algorithm (FWA)

The Fireworks Algorithm (Tan and Zhu, 2010) is a swarm-intelligence metaheuristic inspired by the spectacular display of fireworks. Given a set of candidate solutions ("fireworks"), each firework $\mathbf{x}_i$ launches a burst of s_i sparks whose positions are sampled within an amplitude A_i around $\mathbf{x}_i$:

$$\hat{\mathbf{x}}_i^{(k)} = \mathbf{x}_i + A_i \cdot \boldsymbol{\delta}_k, \quad \boldsymbol{\delta}_k \sim \mathcal{U}(-1,1)^d, \quad (2)$$

where d is the dimensionality of the search space. The explosion amplitude and spark count are adaptively modulated by relative fitness:

$$A_i = A_{\max} \cdot \frac{f(\mathbf{x}_i) - f_{\min} + \varepsilon}{\sum_j (f(\mathbf{x}_j) - f_{\min}) + \varepsilon}, \quad (3)$$

$$s_i = m \cdot \frac{f_{\max} - f(\mathbf{x}_i) + \varepsilon}{\sum_j (f_{\max} - f(\mathbf{x}_j)) + \varepsilon}. \quad (4)$$

where $f(\cdot)$ denotes the objective function, $f_{\min}$ and $f_{\max}$ are the current minimum and maximum fitness values, $A_{\max}$ and m are user-defined upper bounds, and ε is a small constant preventing division by zero.

After each explosion, a selection operator retains a fixed population of N survivors from the union of fireworks and their sparks, balancing quality and spatial diversity. In LEAF, we lift the FWA metaphor to the discrete, language-conditioned search space of agentic ML: each MCTS node acts as a firework, its amplitude corresponds to the semantic temperature of LLM sampling, and sparks are diverse child hypotheses generated around a promising parent solution. This integration directly addresses the homogeneity bottleneck of vanilla MCTS by enforcing controlled yet broad divergence from high-reward nodes.

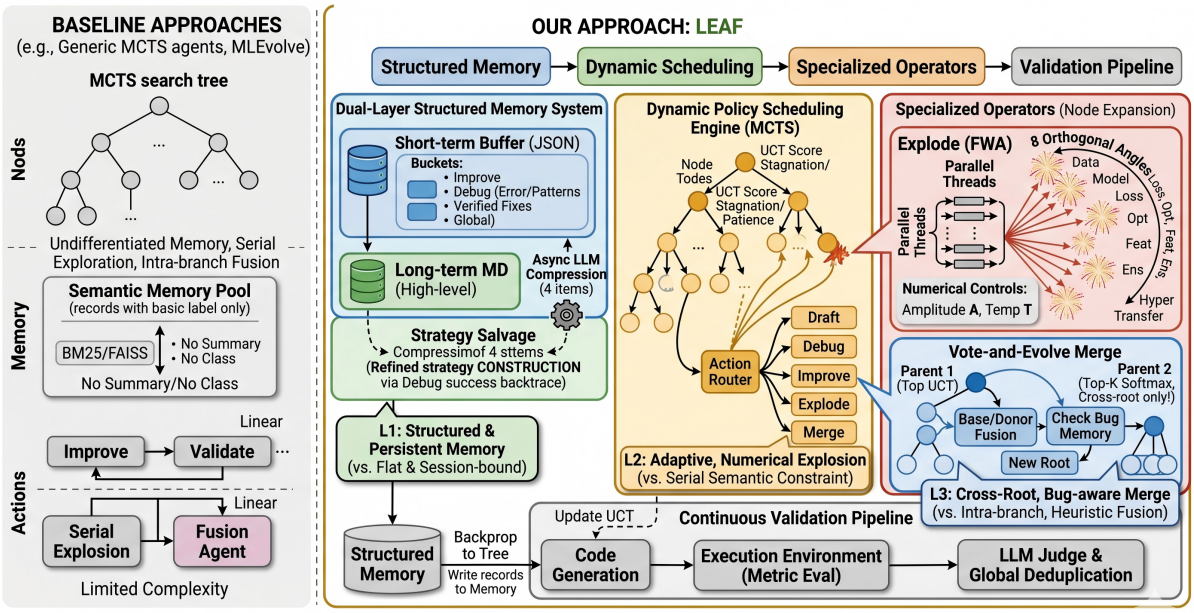


Figure 1: The left panel illustrates the standard baseline approach, while the right panel depicts our proposed LEAF framework. The system integrates natural language planning, coding, and continuous validation, driven by a dynamic policy scheduling engine. Key differences in LEAF include the systematic branching of the search tree through Draft, Debug, Improve, Explode, and Merge operations, which are supported by structured memory and explicit diversity control.

3 Methodology

3.1 Overall Framework Architecture

LEAF drives autonomous ML optimization through a dynamic MCTS loop (see Figure 1) whose search policy adaptively allocates probability across five actions—*Draft*, *Debug*, *Improve*, *Explode*, and *Merge*—as search progresses. Our primary reference baseline is ML-Master 2.0 (Zhu et al., 2026), a concurrent system that also combines MCTS-style tree search with LLM generation but lacks explicit diversity control and structured memory management. Each executed node yields a scalar reward that is backpropagated to update UCT scores:

$$\mathrm{UCT}(i) = \frac{R_i}{N_i} + c\sqrt{\frac{\ln(N_p + 1)}{N_i}}, \qquad (5)$$

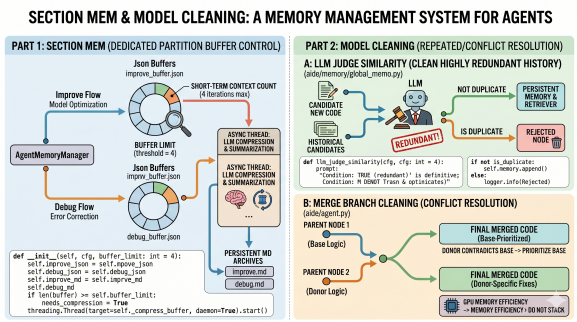
where R_i and N_i are the accumulated reward and visit count of node i, N_p is the maximum visit count among its logical parents, and $c = \sqrt{2}$.

Two core mechanisms address solution homogeneity and memory degradation. **(1) Dual-layer memory.** Short-term rolling buffers (*Improve* and *Debug*, capacity 4) are asynchronously distilled into persistent Markdown logs capturing proven strategies, error patterns, and defensive rules; a long-term Global Memory vector-indexes all accepted nodes and rejects redundant duplicates via a three-step LLM–FAISS–LLM pipeline (Section 3.2). **(2) Fireworks explosion and cross-branch merge.** When progress stagnates, a forced high-temperature explosion escapes the local optimum by generating a burst of diverse child hypotheses; cross-branch merge fuses solutions from distinct search lineages to transfer complementary strengths (Section 3.4).

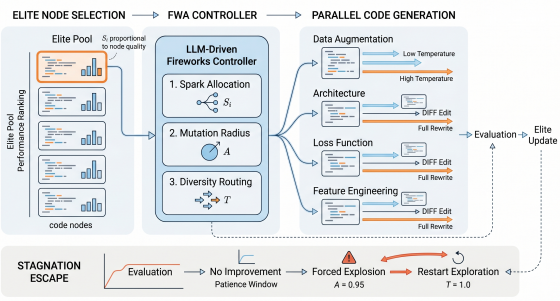
3.2 Decoupled Memory Architecture

To prevent information degradation across long search horizons (Wang et al., 2025; Zhang et al., 2024), LEAF organises agent memory into two temporal layers and three specialised streams.

Short-term Buffer Memory. Two independent JSON rolling buffers track recent execution history: the *Improve Buffer* collects every successfully executed optimisation node (i.e. nodes with `is_buggy = False`), while the *Debug Buffer* collects all repair attempts regardless of outcome. Each buffer enforces a capacity limit of four entries. When the limit is reached, an *asynchronous* LLM thread is launched to distil the batch—the main search loop is never blocked—and appends the resulting structured summary to a persistent



(a) Architecture of the Decoupled Memory.



(b) Fireworks Algorithm (FWA) Search.

Figure 2: Visualizing the core mechanisms of LEAF: (a) shows the decoupled memory architecture for context efficiency, and (b) illustrates the FWA exploration strategy.

Markdown log, following the reflective distillation paradigm of Park et al. (2023). For the Improve Buffer, the summary extracts: (1) *positive guidance*—proven strategies that raised the metric; (2) *negative constraints*—actions that froze or degraded performance; (3) *synergy observations*—technique combinations whose joint effect exceeds their individual contributions; and (4) *next-step recommendations* grounded in the current best baseline. For the Debug Buffer, the summary identifies: (1) *frequent error patterns* and their root causes; (2) *verified fixes*—exact plans and code changes where `is_buggy` became `False`; and (3) *defensive coding rules* to prevent recurrence.

A cross-stream recovery mechanism handles the case where an improve node initially fails but is subsequently repaired by a debug node: on successful repair, the system traces back to the original improve intent and synthesises a hybrid entry inserted into the Improve Buffer, ensuring that valid strategies are not lost due to transient runtime errors.

Long-term Persistent Memory. The Markdown logs accumulated by the asynchronous distillation passes constitute the long-term persistent layer. At every generation step, `get_memory_context` returns the concatenation of (i) the full Markdown history (long-term) and (ii) the current uncompressed JSON buffer (short-term), providing the LLM with both high-level strategic summaries and fine-grained recent records simultaneously. Concretely, IMPROVE and EXPLODE actions receive the improve context under the heading *Strategy Evolution Memory*, while DEBUG actions receive the debug context.

3.3 Semantic Deduplication via Global Memory

As the search tree grows, LLMs tend to regenerate structurally similar code under different surface forms—e.g., trivial variable renaming or negligible hyperparameter perturbations (Xu et al., 2025; Shi et al., 2025). To filter such redundancy at scale, LEAF maintains a *Global Memory* that operates synchronously on every newly generated node via a three-step pipeline:

1. **LLM Structural Compression.** The raw code is compressed into a structured descriptor capturing its summary, feature engineering steps, model architecture, training strategy, and a pseudocode-level `core_framework` skeleton. This representation abstracts away surface noise and exposes the underlying algorithmic identity.

2. **Dense Retrieval.** The compressed descriptor is encoded with a sentence transformer and queried against a FAISS index (Johnson et al., 2021) over all previously accepted nodes to retrieve the top-k most semantically similar candidates.

3. **LLM Semantic Judgement.** An LLM adjudicator compares the new code against the retrieved candidates in batch, deciding whether the new node constitutes a *redundant iteration* (only cosmetic or negligible changes) or a *valid novelty* (new architecture, data pipeline, training mechanism, or bug fix). Only nodes judged as novel are admitted to the search tree and persisted to `global_memory.json`; duplicates are discarded and the generation is retried up to three times.

3.4 LLM-Driven Fireworks Exploration and Stagnation Escape

While deduplication ensures structural uniqueness, effectively navigating the vast machine learning code space requires dynamically adjusting the magnitude and direction of code mutations. To this end, we formulate an LLM-adapted Fireworks Algorithm (FWA) (Tan and Zhu, 2010) as the primary generative engine.Diverging fundamentally from traditional LLM agents that rely on linear prompting or monolithic generation loopswhich often confine code evolution to narrow, localized trajectoriesour FWA mechanism exponentially expands the explorable search space. By translating continuous optimization mechanics (sparks and explosion radii) into discrete LLM generation parameters, it enables a fine-grained, dynamic balance between exploitation (fine-tuning) and exploration (radical refactoring).

When an elite node is selected for expansion, the FWA governs the generation of its descendants through three mathematical mappings:

- **Metric-Aware Spark Allocation:** The computational budgetdefined as the number of generated variants or "sparks" (S_i)is strictly proportional to the node's relative performance. For a selected node i, S_i is calculated based on its metric difference compared to the worst-performing node in the active elite pool. Consequently, higher-quality nodes receive exponentially more sparks (bounded to a hardware-constrained range, e.g., $[1, 8]$), concentrating generation resources on promising topological regions.

- **Adaptive Mutation Radius (Amplitude):** In traditional FWA, nodes closer to the global optimum exhibit a smaller explosion radius to facilitate local exploitation. We map this amplitude A to LLM generation behavior based on the distance between the node's metric and the global best.

  - *Exploitation ($A < 0.5$):* Nodes approaching the optimum receive low LLM sampling temperatures ($T \leq 0.6$). The generation format is restricted to localized Search/Replace DIFF edits, inherently enforcing micro-level hyper-parameter tuning or minor logic tweaks.
  - *Exploration ($A \geq 0.5$):* Sub-optimal nodes are assigned high temperatures ($T \geq 0.8$) and forced to output complete code structures, triggering macroscopic architectural rewrites.

- **Orthogonal Diversity Routing:** To maximize the conceptual coverage of the allocated sparks and prevent the LLM from generating semantically homogeneous code, each parallel spark is conditioned on a distinct optimization heuristic. These predefined angles (e.g., *Data Augmentation*, *Model Architecture Modification*, *Loss Function Adjustment*, *Feature Engineering*) act as directional vectors, ensuring the explosion spreads evenly across the high-dimensional feature space.

- **Patience-Based Forced Explosion for Stagnation Escape**A critical vulnerability of LLM-driven code generation is the tendency to collapse into local optima, repeatedly generating structurally similar, high-scoring but marginally plateaued code. The FWA mitigates this via a deterministic escape mechanism. The system continuously tracks the global best metric; if no relative improvement exceeding a strict threshold (e.g., 5%) is observed within a continuous patience window (e.g., 20 steps), a **Forced Explosion** is triggered.

Bypassing standard UCT selection, this mechanism targets the stagnant global best node and violently disrupts its logic by applying a near-maximum amplitude ($A = 0.95$) and the highest temperature ($T = 1.0$). This forces the LLM to abandon the established, safe algorithmic trajectory and hallucinate radical, high-entropy solutions, effectively kicking the search process out of the local optimum and into unexplored domains.

Cross-branch genetic merge. FWA explosions diversify exploration *within* a single branch; cross-branch merge provides the complementary operation by recombining solutions from *across* independent lineages. At each merge step the *Base parent* p_1 is the highest-UCT node in the tree; the *Donor parent* p_2 is drawn via UCT-weighted softmax over the top-k elite nodes under a hard cross-root constraint—p_2 must originate from a *different root* than p_1, preventing trivial within-branch blending and maximising complementary capability transfer. The LLM then performs a *Vote-and-Evolve* fusion: it elects a Base and a Donor by holistic comparison, grafts the Donor's high-value components (e.g., a stronger data pipeline

or model head) onto the Base template—deferring to the Donor only when it directly resolves a failure recorded in the Debug Buffer—and emits the merged plan–code pair through the Global Memory deduplication gate, retrying up to three times if a duplicate is detected.

4 Experiments

4.1 Main Results on MLE-bench

We evaluate LEAF on MLE-bench (Chan et al., 2024), a benchmark of 75 completed Kaggle competitions spanning tabular prediction, computer vision, NLP, audio, and scientific modelling, stratified into *Low* (22), *Medium* (38), and *High* (15) difficulty splits; *All* (75) is the primary leaderboard column. Because Kaggle does not release held-out test labels, MLE-bench reconstructs each task by splitting public training data into agent-visible and hidden test partitions graded by per-competition scripts. Each task requires the agent to autonomously inspect data, design and execute a modeling pipeline in an isolated sandbox, debug failures, and produce a valid CSV submission within a **24-hour** wall-clock budget. Submissions are scored against Kaggle percentile thresholds for Bronze, Silver, and Gold medals; the primary metric is *Any Medal Rate* (%)—the fraction of tasks where the agent earns at least a Bronze—averaged over ≥ 3 independent runs and reported as mean $\pm$ SEM.

We compare LEAF against the strongest public MLE-bench leaderboard agents available in our evaluation snapshot. These baselines include frontier-model agent systems such as Famou-Agent 2.0 (Li et al., 2025), AIBuildAI (Zhang et al., 2026), and CAIR MARS+ (Chen et al., 2026), as well as research systems closely related to our setting, including MLEvolve (Du et al., 2025; Feng et al., 2026), PiEvolve (Botla et al., 2025), and ML-Master 2.0 (Zhu et al., 2026). LEAF achieves a record-breaking **66.7%** overall medal rate (± 0.77), ranking first among all evaluated agents. The next strongest competitor, Famou-Agent 2.0, reaches 64.44% (± 1.18), followed by AIBuildAI at 63.11% (± 0.44) and CAIR MARS+ at 62.67% (± 0.77); MLEvolve and PiEvolve both achieve 61.33%, while ML-Master 2.0 trails at 56.44% (± 2.47). (Detailed per-difficulty statistics is provided in Table 3 in Appendix C.)

Mixed Routing. Rather than relying on a single backbone, LEAF employs a heterogeneous *Mixed Routing* strategy that allocates each action type to the model best suited to its cognitive demands (detailed in Appendix B): **Claude Sonnet 4.6** handles planning and code generation (DRAFT/IMPROVE), **Gemini 3 Pro** serves as the execution diagnostics and feedback distillation engine (DEBUG), and **Qwen3.6-Plus** drives the open-ended exploratory operations (EXPLODE/MERGE). This division of labour allows each phase to benefit from its most capable specialist while keeping total inference cost within budget, and we observe that no single frontier model consistently dominates across all three cognitive roles.

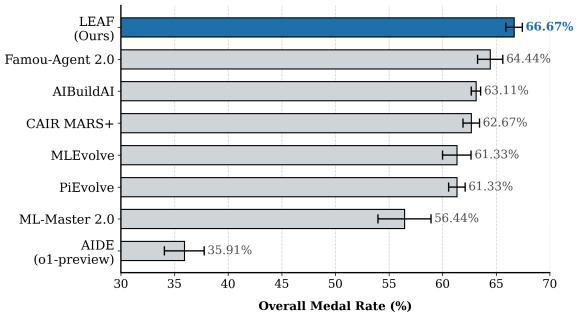


Figure 3: Performance level distribution across top-tier agents on MLE-bench. LEAF exhibits a consistently higher "water level" of engineering proficiency across diverse task categories. Detailed per-difficulty statistics and baseline breakdowns are provided in Table 3 in Appendix C.

4.2 Ablation Study

To evaluate the impact of our core architectural components, we conduct an ablation study focusing on two primary dimensions: the memory management system and the evolution engine.

4.2.1 Ablation on Memory Control

We first evaluate the partitioned context control and model-based similarity arbitration across two representative tasks: *stanford-covid-vaccine* and *lmsys-chatbot-arena*. We compare four variants:

- **Full**: The complete system including all proposed mechanisms.
- **Model Judge**: The system without the LLM-driven semantic deduplication filter.
- **Context Control**: The system without the decoupled, summarizing memory buffer.
- **No Memory**: The system with both the memory buffer and semantic deduplication disabled.

The results are summarized in Table 1. Our findings indicate that the decoupled memory architecture is essential for handling tasks with extremely long reasoning horizons; for instance, the *lmsys-chatbot-arena* task fails to generate a valid submission entirely without context control due to context overflow during iterative refinement. Furthermore, the model-based judge significantly enhances solution quality by preventing the agent from getting trapped in redundant, low-value code modifications, as reflected by the superior scores in the full version.

Configuration	Covid (Score ↓)	LMSYS (Score ↓)
Full System	**0.231**	**1.036**
Model Judge	0.250	1.064
Context Control	0.238	N/A
No Memory	0.258	1.097

Table 1: Ablation results on representative benchmarks. Lower scores are better. Note that without context control, the *Chatbot Arena* task fails to produce a result due to context limit exhaustion.

4.2.2 Ablation on Fireworks-Based Exploration

To isolate the contribution of Fireworks-based exploration, we compare LEAF with a **No Explode** variant that disables both probabilistic explosions and patience-triggered forced explosions. This ablation preserves the remaining search and memory components, but restricts expansion to the standard *Improve* and *Debug* actions. The comparison therefore measures whether explicit FWA-style diversification contributes beyond ordinary iterative refinement and repair.

Configuration	Covid (Score ↓)	LMSYS (Score ↓)
Full System	$\mathbf{0.231 \pm 0.001}$	$\mathbf{1.036 \pm 0.007}$
No Explode	0.313	1.085 ± 0.023

Table 2: Ablation of the Fireworks Algorithm. Scores are averaged over three runs for the full system and nine valid runs for the No Explode variant (4 of 13 runs produced invalid submissions). Lower is better.

Note on metric provenance. Figure 4 depicts **validation-set** convergence trajectories recorded *during* the search process (i.e., scores observed on the local validation split at each step), whereas Tables 1 and 2 report the final **test-set** scores produced by submitting the best solution to the MLE-bench evaluation harness. The two sets of numbers are therefore not directly comparable: for example, the full-system validation MCRMSE of 0.184 (Figure 4b) corresponds to the test-set score of 0.231 reported in Table 1.

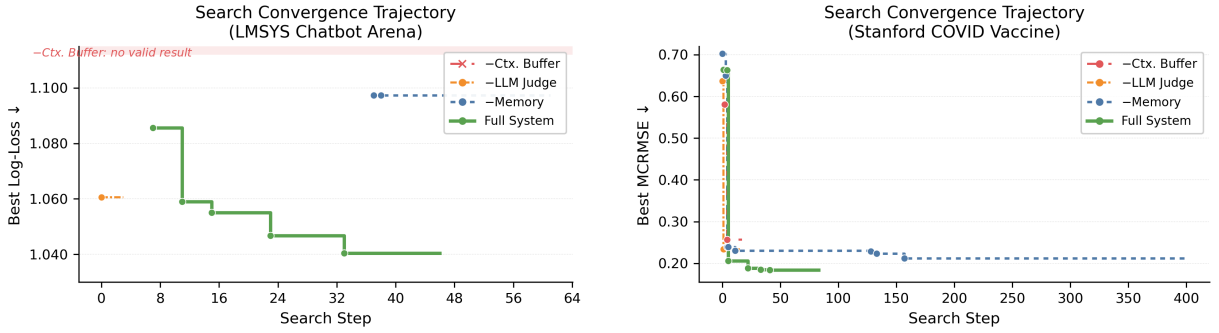
Removing FWA consistently degrades performance on both tasks. On *stanford-covid-vaccine*, the score increases from 0.231 to 0.313 (+35.5% relative). Although the No Explode variant still reaches the gold-medal threshold, it fails to approach the stronger solution found by the full system, suggesting that FWA is most useful for moving beyond competent local refinements. On *lmsys-chatbot-arena*, the mean score worsens from 1.036 to 1.085 (+4.2% relative), and the standard deviation more than triples ($0.007 \rightarrow 0.023$). In addition, 4 of 13 No Explode runs fail to produce a valid submission, whereas all full-system runs remain valid; among the 9 valid No Explode runs, only 2 surpass the median threshold, compared with all three full-system runs. These results indicate that FWA provides two complementary benefits: it broadens local search by generating diverse hypotheses around promising nodes, and it offers an explicit escape mechanism when ordinary refinement stalls. Notably, the Full System's scatter plot (Figure 5, blue) exhibits characteristic non-monotone step-jumps at multiple junctures where improvement had visibly plateaued—a signature of the Patience-Based Forced Explosion mechanism, which triggers high-amplitude diversification after 20 consecutive non-improving steps and steers search toward regions unreachable by ordinary IMPROVE iterations alone. The *No Explode* variant (orange) lacks this escape valve, explaining its higher variance and the elevated rate of invalid runs on LMSYS.

4.3 Case Study: Iterative Optimization on Chatbot Arena

A step-by-step trace of LEAF's optimization trajectory on *lmsys-chatbot-arena*—including a concrete instance of the cross-stream recovery mechanism—is provided in Appendix A.1.

5 Related Work

Agentic Search. Recent MLE automation frameworks have explored graph-based and tree-based search to navigate large solution spaces. MLE-volve (Du et al., 2025; Feng et al., 2026) addresses Kaggle-style competitions via Monte Carlo Graph Search (MCGS) with multi-agent collaboration, while aira-dojo (Toledo et al., 2025) systematically benchmarks diverse search policies—including



(a) LMSYS Chatbot Arena

(b) Stanford COVID Vaccine

Figure 4: **Validation-set** convergence trajectories for four memory ablation variants (not directly comparable to the test-set scores in Tables 1–2). **Left (LMSYS):** −Ctx. Buffer never produces a valid submission (red ×); the full system reaches log-loss 1.040 by step 34 while −Memory plateaus at ~ 1.097. **Right (Stanford):** the full system converges to MCRMSE 0.184 by step 41; −Memory requires 157 steps to reach 0.212 and −Ctx. Buffer stagnates at 0.257.

MCTS and evolutionary strategies—across MLE-bench tasks. Compact single-agent systems such as The FM Agent (Li et al., 2025) and Operand Quant (Sahney et al., 2025) demonstrate that streamlined pipelines remain competitive at scale, while AlphaEvolve (Novikov et al., 2025) extends the evolutionary coding paradigm to scientific and algorithmic discovery.

Iterative Planning and Memory. LoongFlow structures agent behaviour around a Plan–Execute–Summarize (PES) loop, accumulating engineering experience across iterations. ML-Master (Liu et al., 2025) pairs exploratory search with chain-of-thought reasoning for MLE tasks, and its successor ML-Master 2.0 (Zhu et al., 2026) introduces Hierarchical Cognitive Caching (HCC) to distil multi-day research experience into structured context. For longer horizons, Pi-Evolve (Botla et al., 2025) and DeepScientist (Weng et al., 2025) apply evolutionary and progressive refinement strategies to autonomous scientific discovery.

Positioning of LEAF. While the above works make substantial progress in planning and memory structuring, they share two recurring limitations: solution homogenisation under standard tree search and flat or monolithic context management. LEAF directly targets both by augmenting MCTS with amplitude-controlled FWA explosions and cross-branch genetic crossover for diversity, and by decoupling memory into a global semantic store and a reflective sliding-window buffer for precision.

6 Conclusion

LEAF advances agentic ML automation by combining adaptive search, semantic deduplication, and structured memory. It achieves state-of-the-art results on challenging benchmarks and provides a robust foundation for future autonomous agentic research in machine learning.

Limitations

First, system efficiency remains a bottleneck. LEAF prioritizes high-level code intelligence and search decisions, but its sandbox execution, state journaling, and I/O routines are not yet hardware-aware. This limits throughput when scaling to larger search budgets or deployment settings with strict latency constraints.

Second, LEAF can favor structurally simple, robust solutions over complex stacking or large ensembling strategies. This bias improves reliability but may leave marginal performance gains unexplored in competitions where expert systems rely heavily on elaborate model combinations.

Ethics Statement

LEAF is designed strictly for research and educational purposes. Care has been taken to avoid bias amplification within the search space and to prioritize reproducibility in experimental environments. Users are encouraged to apply the framework responsibly and openly acknowledge its autonomous limitations when considering deployment in high-stakes or production applications. Our code and evaluation scripts will be released upon acceptance.

References

Anonymous. 2025. mle-skill: Agent skills for MLE-bench. https://anonymous.4open.science/r/mle-skill-DAA1. GitHub repository; author anonymized for review.

Sai Kiran Botla, Kirubanath Sankar, Abhishek Chopde, and Fardeen Pettiwala. 2025. Pi-evolve: Long-horizon evolutionary optimization for autonomous scientific discovery.

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 Mdry. 2024. Mle-bench: Evaluating machine learning agents on machine learning engineering.

Jiefeng Chen, Bhavana Dalvi Mishra, Jaehyun Nam, Rui Meng, Tomas Pfister, and Jinsung Yoon. 2026. MARS: Modular agent with reflective search for automated AI research. *Preprint*, arXiv:2602.02660.

Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, pages 72–83. Springer.

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

Shiyang Feng, Runmin Ma, Xiangchao Yan, Yue Fan, Yusong Hu, Songtao Huang, Shuaiyu Zhang, Zongsheng Cao, Tianshuo Peng, Jiakang Yuan, Zijie Guo, Zhijie Zhong, Shangheng Du, Weida Wang, Jinxin Shi, Yuhao Zhou, Xiaohan He, Zhiyin Yu, Fangchen Yu, and 39 others. 2026. Internagent-1.5: A unified agentic framework for long-horizon autonomous scientific discovery. *arXiv preprint arXiv:2602.08990*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating language agents on machine learning experimentation. *arXiv preprint arXiv:2310.03302*.

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. Aide: Ai-driven exploration in the space of code.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2021. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML)*, pages 282–293. Springer.

Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, Shenghao Feng, Mo Zhao, Fengzhi Qiu, Rui Yang, Mengmeng Zhang, Wenyi Zhu, Yingying Sun, Quan Sun, Shunhao Yan, and 3 others. 2025. The fm agent. *Preprint*, arXiv:2510.26144.

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2023. Structured chain-of-thought prompting for code generation. *Preprint*, arXiv:2305.06599.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097.

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

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. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *Preprint*, arXiv:2506.13131. ArXiv preprint arXiv:2506.13131.

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

Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, and Michael S. Bernstein. 2023. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST)*.

Arjun Sahney, Ram Gorthi, Cezary astowski, and Javier Vega. 2025. Operand quant: A single-agent architecture for autonomous machine learning engineering. *Preprint*, arXiv:2510.11694.

Zihao Shi and 1 others. 2025. Rethinking memory in llm based agents: Representations, operations, and emerging topics. *arXiv preprint arXiv:2505.00675*.

Ying Tan and Yuanchun Zhu. 2010. Fireworks algorithm for optimization. In *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE.

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, Kelvin Niu, Tatiana Shavrina, Jean-Christophe Gagnon-Audet, Michael Shvartsman, Shagun Sodhani, Alexander H. Miller, Abhishek Charnalia, and 6 others. 2025. Ai research agents for ma-

chine learning: Search, exploration, and generalization in mle-bench. *arXiv*.

Yinzhu Wang and 1 others. 2025. Memory for autonomous LLM agents: Mechanisms, evaluation, and emerging frontiers. *arXiv preprint arXiv:2603.07670*.

Yixuan Weng, Minjun Zhu, Qiujie Xie, Qiyao Sun, Zhen Lin, Sifan Liu, and Yue Zhang. 2025. Deepscientist: Advancing frontier-pushing scientific findings progressively. *Preprint*, arXiv:2509.26603.

Wujiang Xu and 1 others. 2025. A-Mem: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*.

Ruiyi Zhang, Peijia Qin, Qi Cao, Li Zhang, and Pengtao Xie. 2026. AIBuildAI: An AI agent for automatically building AI models. *Preprint*, arXiv:2604.14455.

Zeyu Zhang, Xiaohe Bo, Chao Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Wen, and Ji-Rong Wen. 2024. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*.

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

A Case Studies

A.1 Iterative Optimization on Chatbot Arena

This appendix traces LEAF's full optimization trajectory on the *lmsys-chatbot-arena* task—a preference-prediction problem requiring long-context reasoning and positional bias mitigation—and documents a concrete activation of the cross-stream recovery mechanism described in §3.2.

Debugging and Error Recovery. The agent initially encountered several engineering hurdles. Mixed-precision training (`float16`) caused floating-point drift, leading to submission rejections because prediction rows did not sum exactly to 1.0; the framework reflected on this failure and implemented explicit logit casting to `float32` with row-wise normalization. It further identified and resolved library-specific issues, including a deprecated argument in `scikit-learn`'s `log_loss` and a security-related block on `torch.load` (CVE-2025-32434).

These recoveries also illustrate the *cross-stream recovery* mechanism in action. At step 13, an IMPROVE node proposed Test-Time Augmentation (TTA) with an A/B-swap inference pass, but was immediately blocked by the `torch.load` CVE before any training could begin—the TTA strategy was therefore never recorded as a success in the memory buffer. After a DEBUG node (step 14) diagnosed and partially mitigated the CVE issue, the system traced back to the original TTA proposal, stripped the entangled DeBERTa-large upgrade that had compounded the failure, and re-issued the TTA strategy as a clean IMPROVE node (step 15). This salvaged submission achieved a validation log-loss of **1.0550**—a 0.004 gain over the prior best—demonstrating that the cross-stream recovery mechanism successfully rescued a high-value innovation that would otherwise have been permanently discarded as a failed attempt.

Iterative Optimization Trajectory. Starting from a baseline DistilBERT model (validation log-loss 1.0857), LEAF executed a series of targeted improvements:

- **Heuristic Refinement:** Training-time A/B response swapping and unfreezing the final transformer layers reduced the loss to 1.0590.
- **Post-processing and Augmentation:** Test-Time Augmentation (TTA) with probability inversion—recovered via the cross-stream mechanism above—improved the metric to 1.0550, neutralizing positional artifacts at inference time.
- **Optimization Tuning:** AdamW weight decay (0.01), gradient clipping, and early stopping further drove the log-loss to 1.0468.
- **Context Management:** Token-aware "Head+Tail" extraction (preserving the beginning and end of long responses) with markdown stripping yielded a final validation score of **1.0366**.

This progression illustrates LEAF's ability to resolve low-level runtime failures, salvage interrupted improve intents through memory-guided recovery, and systematically stack high-level heuristics to close the performance gap on complex MLE tasks.

A.2 Cross-Branch Genetic Merge: A Concrete Instance

To complement the linear trajectory above, we document a concrete activation of the cross-branch genetic merge mechanism on the *stanford-covid-vaccine* task (MCRMSE ↓), where two independently evolved solution lineages were recombined into a strictly superior hybrid.

The two parent branches. By step 19, Branch 1 had developed a three-layer BiGRU backbone trained with standard MSE loss, reaching MCRMSE 0.306. Independently, Branch 2 evolved a Conv1D+BiGRU architecture with a target-weighted SmoothL1 loss that explicitly down-weights the two unscored degradation columns, achieving MCRMSE 0.252 by step 42. The two branches had complementary strengths: Branch 2 held a better-aligned loss surface, while Branch 1 had developed richer input representations via learnable nucleotide-position embeddings.

Vote-and-Evolve fusion. At step 45, the merge action selected Branch 2 as the *Base* (higher UCT score) and Branch 1 as the *Donor*. Following the Vote-and-Evolve protocol, it grafted Branch 1's learnable embedding layer onto Branch 2's Conv1D+BiGRU+target-weighted-loss framework, replacing the rigid one-hot encoding while preserving the superior loss alignment. The resulting hybrid achieved MCRMSE $\mathbf{0.239}$—lower than both Parent 1 (0.306) and Parent 2 (0.252)—demonstrating that cross-branch recombination can access solutions outside the reach of either independent lineage.

B Experimental Setup and Implementation Details

Computational Infrastructure All sandbox evaluations and environment execution steps for LEAF were conducted on an isolated server equipped with **1 NVIDIA h100** GPUs and **8 vCPUs**.

Heterogeneous Multi-Model Routing Strategy To maximize semantic throughput while maintaining strategic decoupling across ultra-long horizons, LEAF shifts away from traditional monolithic LLM dependencies. Instead, we implement a Dynamic Cognitive Lifecycle Routing strategy, allocating complementary LLMs as highly specialized agents based on the specific computing and reasoning demands of each execution phase:

1. **Solution Planning & Code Generation (DRAFT / IMPROVE):** All planning and code-writing actions—both initial drafting and iterative improvement—are handled by **Claude Sonnet 4.6**. Its strong instruction-following and long-context reasoning capabilities allow the framework to produce complete, runnable training pipelines and to synthesise incremental refinements from accumulated memory context with high fidelity.

2. **Execution Diagnostics & Performance Feedback (DEBUG / Feedback Distillation):** When the sandbox returns runtime errors or performance stagnation signals, **Gemini 3 Pro** is invoked as the specialized diagnostic auditor. Its robust long-context window enables the framework to digest massive raw execution logs, identify root-cause failure patterns, and distil precise self-reflection memories into the Debug Buffer without losing critical context.

3. **Exploratory Synthesis (EXPLODE / MERGE):** Diversity-seeking operations—both FWA-triggered explosions that generate high-entropy solution variants and cross-branch merge steps that recombine elite solutions from distinct lineages—are routed to **Qwen3.6-Plus**. Its high sampling throughput and broad generalization make it well-suited for the combinatorial, open-ended generation demands of these two actions.

MLE-bench requires agents to continuously process massive environment observations, lengthy execution logs, and iterative code blocks. Across the complete benchmarking runs, our system consumed an estimated total of **1 billion tokens**. Given the heavily state-dependent nature of the framework, this comprised approximately **850 million input tokens** (e.g., sandbox feedback, dataset schemas, and error logs) and **150 million output tokens** (e.g., generated Python scripts and bash commands). The total API expenditure for these evaluations was approximately **$4,000 USD**. To ensure rigorous and fair evaluation, the hyperparameters governing our dynamic Monte Carlo Tree Search (MCTS) and Fireworks Algorithm (FWA) were kept strictly constant across all tasks in MLEBench. We explicitly did not perform task-specific hyperparameter tuning on the evaluation set. The core hyperparameters are summarized as follows:

- **MCTS Exploration Constant (c):** Set to $\sqrt{2} \approx 1.414$ to balance exploration and exploitation.
- **FWA Spark Allocation (S_i):** Dynamically bounded within $[1, 8]$ based on relative node performance.
- **Adaptive Mutation Temperature (T):** $T \leq 0.6$ for local exploitation (Search/Replace DIFFs) and $T \geq 0.8$ for global exploration (full architectural rewrites).
- **Stagnation Escape (Forced Explosion):** Triggered after a patience window of 20 steps if the global best metric improvement is less than 5%. During the explosion, the temperature is maximized to $T = 1.0$ with an amplitude of $A = 0.95$.

Agent	LLM(s)	Low (22)	Med. (38)	High (15)	All (75)
LEAF (Ours)	Mixed Routing†	**81.82**‡	**68.42**‡	**40.00**‡	**66.67** ±0.77
Famou-Agent 2.0	Gemini-3-Pro	80.30±1.52	64.04±2.32	42.22±2.22	64.44±1.18
AIBuildAI	Claude-Opus-4.6	77.27±0.00	61.40±0.88	46.67±0.00	63.11±0.44
CAIR MARS+	Gemini-3-Pro	78.79±1.52	60.53±1.52	44.44±2.22	62.67±0.77
MLEvolve	Gemini-3-Pro	80.30±1.52	57.89±1.52	42.22±2.22	61.33±1.33
PiEvolve	Gemini-3-Pro	80.30±1.52	58.77±0.88	40.00±0.00	61.33±0.77
ML-Master 2.0	DeepSeek-V3.2	75.76±1.51	50.88±3.51	42.22±2.22	56.44±2.47
AIDE	o1-preview	35.91±1.86	8.45±0.43	11.67±1.27	17.12±0.61

†Mixed Routing: Claude Sonnet 4.6 (DRAFT/IMPROVE), Gemini 3 Pro (DEBUG), Qwen3.6-Plus (EXPLODE/MERGE).
‡LEAF per-split: 18/22 (Low), 26/38 (Med.), 6/15 (High) medals from the full 75-task evaluation.

Table 3: Per-difficulty Any Medal Rate (%) for agents in Figure 3. Baselines: mean ± SEM, ≥3 runs, best submission per agent.

C Full MLE-bench Leaderboard Breakdown

Table 3 reports the per-difficulty Any Medal Rate for the eight agents featured in Figure 3, complementing the overall discussion in §4.1. All baseline figures are sourced from the public MLE-bench leaderboard; we list each agent's best-performing submission for clarity.

LEAF's Mixed-Routing configuration. The **66.7%** overall medal rate reported for LEAF is produced exclusively by the **Mixed Routing** setup detailed in Appendix B: **Claude Sonnet 4.6** drives all planning and code-generation steps (DRAFT/IMPROVE), **Gemini 3 Pro** handles execution diagnostics and feedback distillation (DEBUG), and **Qwen3.6-Plus** powers the open-ended exploratory operations (EXPLODE/MERGE). No single backbone is used throughout; the 66.7% figure thus reflects the joint contribution of all three specialist models operating within the LEAF framework.

D Prompt Templates

In this section, we provide the core prompt templates used in LEAF for agent planning, code generation, and memory management.

D.1 Agent Planning and Code Generation

The primary prompt used to guide the agent in designing and implementing ML solutions.

Introduction: You are a Kaggle Grandmaster. Your goal is to design and implement a highly competitive ML solution that learns from data and generates real predictions. Treat this with professionalism.

Execution Constraints: 1. CRITICAL: Set `num_workers=8` in all DataLoaders. 2. Hugging Face mirror: Configure via `HF_ENDPOINT` if downloading models. 3. NO PLACEHOLDERS: The generated Python code must be 100% complete and runnable. 4. Print ONLY the final validation score at the very end of the execution.

ATTENTION: 1. MULTI-EPOCH TRAINING: You MUST implement a full training loop with multiple epochs (e.g., 5-20 depending on task). 2. ITERATIVE LOGIC: The `train_one_epoch` in Kaggle Blueprint is a sub-component; ensure it is called within a proper `for epoch in range(N)` loop.

Response Format: Your response should be a brief outline/sketch of your proposed solution in natural language (MUST be strictly 3-5 sentences), followed by a single markdown code block which implements this solution and prints out the evaluation metric. There should be no additional headings or text in your response. Just natural language text (3-5 sentences total) followed by a newline and then the markdown code block.

D.2 Memory Management: Model/Code Optimization (Improve)

The prompt used to summarize and compress historical improvement nodes into persistent memory.

You are a top-tier Machine Learning Expert and AI Architect. Below are the detailed records of nodes generated by an AI Agent during recent "Model/Code Optimization (Improve)" iterations. Your task is to act as the "Long-term Memory & Search Policy Engine". Analyze these attempts and call the provided function to generate a highly concise, quantitative, and instructive summary.

Analysis Requirements & Constraints: 1. **Quantitative Traceability (CRITICAL)**: NEVER use vague qualitative words like "moderate", "slightly better", or "large". You MUST cite specific metric changes and exact parameter values. 2. **Performance Comparison & Hybrid Evaluation**: Compare the metric of each node against

its parent. Evaluate the combined algorithmic intent, not just the bugs. 3. **Algorithmic Focus for Pitfalls**: Only list fundamental ML strategy failures (e.g., incorrect architectures, harmful data augmentations). DO NOT list basic Python syntax errors. 4. **Method Induction & Synergy**: Extract proven combinations. Observe if there are "synergy benefits". 5. **Strategic Next Steps**: Propose 2-4 concrete, incremental ML experiments based strictly on the current successful baselines.

D.3 Memory Management: Code Repair (Debug)

The prompt used to audit failed attempts and extract verified code fixes.

You are a top-tier Senior Software Engineer acting as a strict Code Auditor. Below are the detailed records of historical attempts extracted from an AI Agent's Debug Memory Buffer during a "Code Repair" task.

CRITICAL AUDIT CONSTRAINTS: 1. **ANTI-HALLUCINATION**: You MUST base your analysis STRICTLY on the provided Memory Buffer. Do NOT invent, guess, or pull external "best practices". 2. **TRUTH OF `is_buggy`**: If `is_buggy = true`, the plan and code represent a FAILED attempt. A bug is ONLY considered "Resolved" if there is a specific node where `is_buggy = false`.

Analysis Requirements: 1. **Root Cause Tracking**: Analyze the terminal output across the buffer to summarize the most frequent errors. 2. **Verified Fixes (If Any)**: Identify specific nodes where the bug was successfully fixed and summarize the EXACT changes. 3. **Failed Attempts & Symptomatic Patches**: Explicitly identify the bugs that were NEVER resolved and what was tried that did not work. 4. **Defensive Programming Guidance**: Extract coding standards to prevent future similar bugs based ONLY on validated fixes and identified failures.

D.4 Code Review Prompt

The prompt used for autonomous safety and integrity checks.

You are a Senior Data Science Code Reviewer. Your goal is to ensure the submission is legally valid and logically sound.

CRITICAL INSTRUCTION: You must strictly follow the Code Review Guidelines. Do NOT rely on your general knowledge if it conflicts with the Environment Facts. Your output must be a structured review focusing ONLY on Data Leakage and Critical Integrity.

STRICTLY FORBIDDEN: Do NOT replace the user's model architecture with other backbones just to make code executable. Do not question or change the user's model choice.

Environment Facts: - Paths: `./input/`, `./working/`, `./submission/` ALL EXIST. - Submission File Location: Must save to `./submission/submission.csv`. - Bleeding Edge: Assume the latest library versions and model availability.

E Ablation Study details

This section provides granular statistics supporting the ablation findings discussed in §4.2. We report step-level diagnostics for two representative tasks: **LMSYS Chatbot Arena** (logloss, ↓) and **Stanford COVID-19** (MCRMSE, ↓), across four variants: Full System (*total*), −LLM Judge (*no_model_judge*), −Ctx. Buffer (*no_context_ctr*), and −Memory (*no_mem*).

Step-level performance statistics. Table 4 summarises the total search budget consumed, the number of steps that produced a valid (executable, non-leaking) submission, the index of the first valid step, the best metric achieved, and the step at which it was reached.

Table 4: Step-level search statistics for two ablation tasks. *First valid* gives the step index of the first executable submission; *Best metric* is the optimum score achieved; *Step@best* is when it was reached.

Task	Variant	Steps	Valid	First valid	Best metric	Step@best
LMSYS	Full System	47	15	7	**1.040**	33
	−LLM Judge	4	1	0	1.061	0
	−Ctx. Buffer	4	0			
	−Memory	62	5	37	1.097	38
Stanford	Full System	84	44	1	**0.184**	41
	−LLM Judge	17	15	0	0.234	1
	−Ctx. Buffer	18	12	2	0.257	4
	−Memory	401	282	0	0.212	157

Action-type distribution and bug rates. Table 5 breaks down each variant's steps by node type (*draft*, *improve*, *debug*) and reports the number of buggy (non-executable or integrity-failing) steps within each type. Several patterns stand out. On LMSYS, the −Ctx. Buffer and −LLM Judge variants never advance beyond the draft stage: all four of their steps are draft attempts that fail, consistent with the context-bloat hypothesis above. The −Memory variant on LMSYS devolves into a debug loop: 47 of 62 steps are debug nodes (76%), and 44 of those 47 are still buggy (94%), indicating that without accumulated global experience the agent cannot escape repeated failure patterns. By contrast, the Full System on LMSYS allocates 34% of its budget to improve nodes (16/47) with a 69% success rate, reflecting productive refinement rather than remediation.

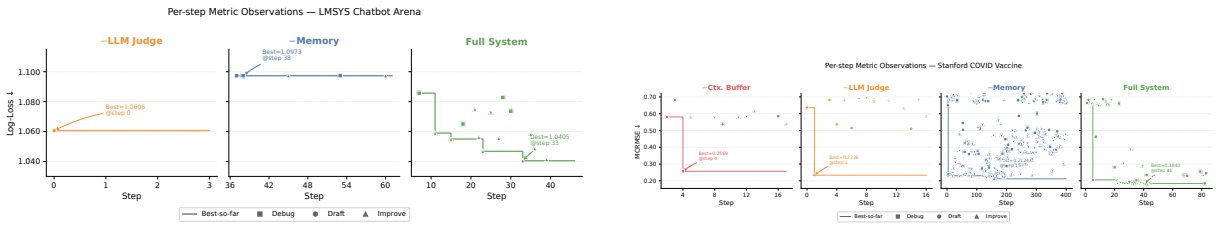
Table 5: Action-type distribution and buggy-step counts per variant per task. Parenthetical figures indicate steps where the generated code failed execution or integrity checks.

Task	Variant	Draft (buggy)	Improve (buggy)	Debug (buggy)
LMSYS	Full System	5 (5)	16 (5)	26 (22)
	−LLM Judge	4 (3)	0 (0)	0 (0)
	−Ctx. Buffer	4 (4)	0 (0)	0 (0)
	−Memory	3 (3)	12 (10)	47 (44)

Per-step metric observations. Figure 5 plots every valid metric observation against its step index for each variant on both tasks. The Full System's trajectory (blue) consistently achieves lower loss / error earlier and continues to improve to a better final value. The −Memory variant (red) on Stanford generates hundreds of data points but plateaus well above the Full System's optimum, illustrating search breadth without depth.

F Detailed MLE-bench Results

Implementation Details for Full-Scale Evaluation. To assess the generalization of the LEAF framework across a comprehensive set of tasks, we conducted a full-scale evaluation over all 50 MLE-bench competitions reported in Table 6. Due to the substantial computational and budget constraints imposed by running 50 long-horizon agentic experiments—each of which may consume hundreds of LLM API calls across multi-hour search sessions—this large-scale evaluation uniformly employs **Qwen-3.5-Plus** as the backbone model in place of the stronger **Qwen-3.6-Max** used in the core ablation experiments (§4.2). As a result, the absolute scores reported in this appendix (e.g., 0.2589 for *Stanford COVID Vaccine*) will differ from the scores in the main-body tables (e.g., 0.231 in Table 1), which were obtained with the stronger backbone. Readers should interpret the full-scale results as a demonstration of LEAF's



(a) LMSYS Chatbot Arena (logloss ↓)

(b) Stanford COVID-19 (MCRMSE ↓)

Figure 5: Per-step metric observations for each ablation variant. Each point is a valid submission; the dashed line traces the running best. Axes are clipped for readability (Stanford −Memory extends to step 401).

breadth and generalization across diverse task types, rather than as a direct numerical comparison with the controlled core experiments. Table 6 presents the comprehensive evaluation results of LEAF across

50 diverse Kaggle competitions from the MLE-bench suite. The competitions are grouped by the medal achieved by our system. The metric direction (↑ for higher-is-better, ↓ for lower-is-better) is indicated for clarity.

Table 6: Detailed performance of LEAF across 50 MLE-bench competitions.

Competition Name	Metric	LEAF Score	Medal Threshold	Medal
Gold Medals (19 Tasks)				
Histopathologic Cancer Detection	↑	0.9976	0.9835	Gold
MLSP 2013 Birds	↑	0.9584	0.9352	Gold
Detecting Insults In Social Commentary	↑	0.9548	0.8332	Gold
Predict Volcanic Eruptions INGV OE	↓	1445305.0	3971366.0	Gold
Tensorflow Speech Recognition Challenge	↑	0.9609	0.9048	Gold
Seti Breakthrough Listen	↑	0.8278	0.7980	Gold
iWildCam 2019 FGVC6	↑	0.3641	0.2120	Gold
Dogs Vs Cats Redux Kernels Edition	↓	0.0052	0.0388	Gold
Denoising Dirty Documents	↓	0.0066	0.0179	Gold
LMSYS Chatbot Arena	↓	0.9787	0.9839	Gold
Plant Pathology 2021 FGVC8	↑	0.9203	0.8612	Gold
Tabular Playground Series Dec 2021	↑	0.9623	0.9566	Gold
Aerial Cactus Identification	↑	1.0000	1.0000	Gold
VinBigData Chest X-ray Abnormalities	↑	0.3248	0.2890	Gold
ICML 2013 Whale Challenge Redux	↑	0.9905	0.9896	Gold
Learning Agency Lab Essay Scoring 2	↑	0.8421	0.8358	Gold
Plant Pathology 2020 FGVC7	↑	0.9989	0.9783	Gold
Tabular Playground Series May 2022	↑	0.9982	0.9982	Gold
Stanford Covid Vaccine	↓	0.2589	0.3472	Gold
Silver Medals (16 Tasks)				
Google Quest Challenge	↑	0.4064	0.3959	Silver
Petfinder Pawpularity Score	↓	17.0455	17.0663	Silver
Nomad2018 Predict Transparent Conductors	↓	0.0595	0.0622	Silver
Leaf Classification	↓	0.0016	0.0079	Silver
HubMAP Kidney Segmentation	↑	0.9468	0.9455	Silver
iWildCam 2020 FGVC7	↑	0.7286	0.6880	Silver
US Patent Phrase To Phrase Matching	↑	0.8688	0.8630	Silver
Tweet Sentiment Extraction	↑	0.7177	0.7175	Silver
RSNA MICCAI Brain Tumor Radiogenomic	↑	0.5923	0.5815	Silver
Spooky Author Identification	↓	0.2477	0.2699	Silver
Jigsaw Toxic Comment Classification	↑	0.9867	0.9866	Silver
Cassava Leaf Disease Classification	↑	0.9002	0.8992	Silver
iMet 2020 FGVC7	↑	0.6529	0.6490	Silver
Whale Categorization Playground	↑	0.4614	0.4485	Silver
Statoil Iceberg Classifier Challenge	↓	0.1313	0.1375	Silver
iNaturalist 2019 FGVC6	↓	0.2376	0.2605	Silver
Bronze Medals (15 Tasks)				
NFL Player Contact Detection	↑	0.6976	0.6965	Bronze
H and M Personalized Fashion Recs	↑	0.0248	0.0239	Bronze

Continued on next page

Table 6 continued from previous page

Competition Name	Metric	LEAF Score	Medal Threshold	Medal
Multi Modal Gesture Recognition	↓	0.1885	0.2492	Bronze
Hotel ID 2021 FGVC8	↑	0.3215	0.0216	Bronze
Chaii Hindi And Tamil QA	↑	0.7373	0.7372	Bronze
Text Normalization Challenge English	↑	0.9912	0.9903	Bronze
Herbarium 2020 FGVC7	↑	0.2033	0.0533	Bronze
Freesound Audio Tagging 2019	↑	0.6996	0.6923	Bronze
Text Normalization Challenge Russian	↑	0.9798	0.9759	Bronze
Random Acts Of Pizza	↑	0.6962	0.6921	Bronze
Kuzushiji Recognition	↑	0.7253	0.6580	Bronze
Herbarium 2022 FGVC9	↑	0.6035	0.5965	Bronze
Herbarium 2021 FGVC8	↑	0.1856	0.1302	Bronze
Aptos2019 Blindness Detection	↑	0.9176	0.9144	Bronze
Cdiscount Image Classification	↑	0.7236	0.7089	Bronze

G Full-Scale System Comparison on MLE-bench

LEAF is built upon AIDE (Jiang et al., 2025) as its starting codebase, extending it with MCTS-guided search, dual-layer memory, and Fireworks-based exploration. We evaluate LEAF against two agentic ML baselines—ML-Master (Zhu et al., 2026) and MLEvolve (Du et al., 2025; Feng et al., 2026)—using the official MLE-bench evaluation harness. Each system was run under its published default configuration; we report the best-scoring valid submission per competition per system. Table 7 summarises aggregate outcomes and Table 8 gives a per-competition breakdown for eight representative tasks.

Table 7: Aggregate performance across all MLE-bench competitions with valid submissions. All evaluated systems were run on 75 MLE-bench tasks using a unified foundational model (Qwen). The **#Comps** column denotes the number of competitions that successfully generated valid submission files and non-null scores. **G/S/B** denote Gold, Silver, and Bronze medals respectively; *Above Med.* counts competitions where the submission surpassed the human-median score (medals included).

System	#Comps	Above Med.	G	S	B	Total
LEAF (ours)	**38**	**13**	**6**	**1**	2	**9**
MLEvolve	25	10	2	1	**3**	6
ML-Master	26	6	1	1	1	3

Table 8: Score comparison on eight representative MLE-bench competitions. ↑ = higher is better; ↓ = lower is better. Superscripts $^{G}/^{S}/^{B}$ denote Gold/Silver/Bronze medals; $^{\checkmark}$ = above human median. **Bold** = best score in that row.

Competition	Dir.	LEAF (ours)	ML-Master	MLEvolve
Aerial Cactus Identification	↑	**1.000**^{G}	0.9999	0.99999
Leaf Classification	↓	**0.010**^{B}	0.069	0.096
Whale Categorization	↑	**0.343**$^{\checkmark}$	0.119	0.280
US Patent Phrase Matching	↑	**0.847**	0.842	0.825
Google Quest Challenge	↑	0.390^{B}	**0.415**^{S}	0.306
Jigsaw Toxic Comments	↑	0.986	0.986	**0.987**^{B}
MLSP-2013 Bird Detection	↑	0.744	0.789	**0.886**^{B}
NYC Taxi Fare Prediction	↓	6.269	11.650	**5.148**

On the full evaluation suite, LEAF achieves the highest total medal count (9, including 6 Golds) across 38 competitions, reflecting broad coverage and consistent solution quality. Across the eight representative tasks, LEAF records the best score in the top four competitions and leads on *Aerial Cactus* (Gold), *Leaf Classification* (Bronze), *Whale Categorization* (the sole above-median result), and *US Patent Phrase Matching*. ML-Master achieves itits strongest result on *Google Quest Challenge* (Silver), where its language-understanding strengths are most relevant. MLEvolve excels on the remaining three tasks: *Jigsaw Toxic Comments* (Bronze, by a 0.001 margin), *MLSP-2013 Bird Detection* (Bronze), and *NYC Taxi Fare Prediction*—tasks that favour broad pattern recognition over deep iterative refinement. Across both the full and the common subset, LEAF's dual-layer memory and MCTS-guided firework

and merge algorithm provide a consistent advantage on tasks requiring long-horizon optimisation, while each baseline demonstrates complementary domain strengths.