REASONING AS GRADIENT: SCALING MLE AGENTS BEYOND TREE SEARCH

Yifei Zhang[1,2]*, **Xu Yang**[1]*, **Xiao Yang**[1]*,
Bowen Xian[1], **Qizheng Li**[1,3], **Shikai Fang**[4], **Jingyuan Li**[1,5],
Jian Wang[1], **Mingrui Xu**[6], **Yuge Zhang**[1], **Weiqing Liu**[1]†, **Jiang Bian**[1]
[1]Microsoft Research Asia, [2]Nanjing University, [3]Peking University,
[4]Zhejiang University, [5]Wuhan University, [6]Hong Kong University of Science and Technology
https://github.com/microsoft/RD-Agent

ABSTRACT

LLM-based agents for machine learning engineering (MLE) predominantly rely on tree search, a form of gradient-free optimization that uses scalar validation scores to rank candidates. As LLM reasoning capabilities improve, exhaustive enumeration becomes increasingly inefficient compared to directed updates, analogous to how accurate gradients enable efficient descent over random search. We introduce GOME, an MLE agent that operationalizes gradient-based optimization. GOME maps structured diagnostic reasoning to gradient computation, success memory to momentum, and multi-trace execution to distributed optimization. Under a closed-world protocol that isolates architectural effects from external knowledge, GOME achieves a state-of-the-art 35.1% any-medal rate on MLE-Bench with a restricted 12-hour budget on a single V100 GPU. Scaling experiments across 10 models reveal a critical crossover: with weaker models, tree search retains advantages by compensating for unreliable reasoning through exhaustive exploration; as reasoning capability strengthens, gradient-based optimization progressively outperforms, with the gap widening at frontier-tier models. Given the rapid advancement of reasoning-oriented LLMs, this positions gradient-based optimization as an increasingly favorable paradigm. We release our codebase and GPT-5 traces.

1 INTRODUCTION

Automating Machine Learning Engineering (MLE) remains a long-standing challenge. With rapid advances in code generation and reasoning capabilities (Jaech et al., 2024; Guo et al., 2025a), LLM-based MLE agents have recently emerged (Jiang et al., 2025; Liu et al., 2025), demonstrating the ability to autonomously diagnose data and implement predictive pipelines. However, despite these advances, the prevailing paradigm, pioneered by AIDE (Jiang et al., 2025) and adopted by successors like ML-Master (Liu et al., 2025), remains rooted in **tree search**. These gradient-free approaches exhibit two fundamental limitations. First, they rely on **score-based expansion**: rich execution feedback (e.g., error traces, detailed logs) is scalarized into a single metric to decide *which* node to expand, discarding the diagnostic information needed to determine *how* to update. Second, they operate over a **predefined action space**: agents select among fixed templates, failing to capture the effectively continuous nature of code modifications.

We argue that MLE inherently favors **gradient-based optimization** over search (Figure 1). Unlike domains punctuated by hard dead ends, MLE pipelines are typically *repairable* and effectively continuous, favoring methods that propose state-conditioned updates over fixed-action enumeration. In this analogy, execution feedback serves as the gradient signal, and LLM reasoning computes the update direction Δ. However, gradient methods require accurate signals: with weak reasoners, diagnostic reasoning produces noisy Δ (e.g., hallucinated edits), necessitating tree search as a hedge against unreliability. As reasoning capabilities advance, we claim that a **crossover** emerges:

*Equal contribution.
†Corresponding author.

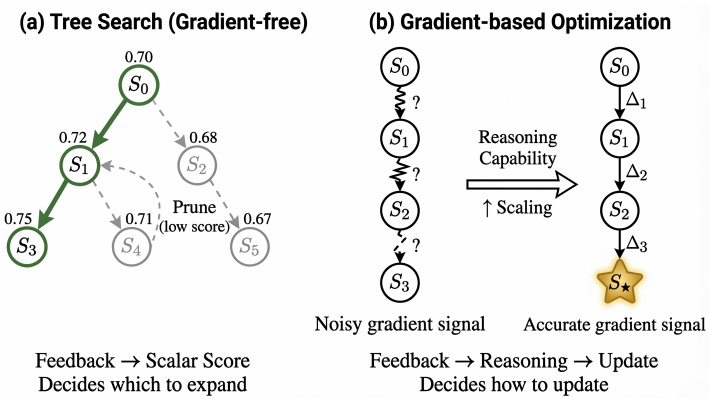


Figure 1: Comparison of search-based (gradient-free) and gradient-based optimization paradigms. (a) Tree search uses scalar scores to decide *which* branch to expand. (b) Gradient-based optimization uses reasoning to decide *how* to update; stronger reasoning yields more accurate gradient signals.

gradient-based optimization should increasingly outperform search, enabling direct convergence to optimal solutions s^* rather than relying on exhaustive enumeration.

To test this claim, we propose **GOME** (**G**radient-based **O**ptimization for **M**achine Learning **E**ngineering). GOME adopts an iterative framework where modular components mirror classical optimizers (Table 2): structured reasoning extracts improvement directions from execution feedback (gradient signals), success memory accumulates proven patterns (momentum), and parallel traces share updates (distributed optimization). We evaluate under a **closed-world protocol**, restricting agents to task-provided materials to isolate intrinsic optimization capability from external knowledge augmentation. Under this rigorous setting, GOME achieves a state-of-the-art **35.1%** any-medal rate on MLE-Bench with GPT-5 using a single **V100 GPU within 12 hours** , surpassing search-based baselines. Scaling analysis across 10 models ranging from general (GPT-4o) to frontier reasoning models (GPT-5) validates our claim: while search methods excel with weaker reasoners, GOME's advantage grows with reasoning capability, widening to **+7.1%** with frontier models. This confirms that as reasoning scales, gradient-based optimization becomes the superior paradigm.

In summary, this work makes the following key contributions: (1) We propose GOME, an MLE agent that adopts gradient-based optimization rather than gradient-free enumeration, establishing a functional mapping between agent components and classical optimizer modules. (2) GOME achieves state-of-the-art performance on MLE-BENCH under closed-world evaluation, with scaling analysis confirming that gradient-based optimization scales more effectively with reasoning capability than tree search. (3) We release our codebase and GPT-5 execution traces to support reproducibility.

2 RELATED WORK

LLM-based MLE Agents. Existing MLE agents predominantly treat code generation as a search problem over a fixed action space. As summarized in Table 1, approaches explore various topologies including tree search (Jiang et al., 2025; Liu et al., 2025; Kulibaba et al., 2025), graph-based exploration (Toledo et al., 2025; InternAgent-Team, 2025), and evolutionary strategies (Li et al., 2025). While some methods augment search with external retrieval (Nam et al., 2025), they share a fundamental limitation: execution feedback is used primarily for *ranking*. Rich signals (logs, errors) are scalarized into validation scores to select *which* node to expand, rather than diagnosing *how* to update. For instance, MLE-STAR (Nam et al., 2025) refines code blocks via ablation-based scoring, yet still relies on enumeration rather than deriving updates from execution dynamics (see Appendix D.2). In contrast, GOME shifts the paradigm from ranking to *update*: it integrates scalar scores with diagnostic feedback to generate state-conditioned hypotheses, effectively treating reasoning as a gradient signal.

Table 2: Analogy between GOME and gradient-based optimization.

Concept	GOME Component	Functional Role
Gradient ∇L	Structured Reasoning	Signal: decides *how* to update
Momentum	Success Memory	Acceleration via proven patterns
Distributed SGD	Multi-trace Optimization	Parallelism with knowledge sharing

Table 1: Comparison of LLM-based MLE agents.

Method	Structure	Feedback Role	Ext. Know.	Code
AIDE	Tree	Ranking	✗	✓
ML-Master	Tree	Ranking	✗	✓
AIRA	Graph	Ranking	✗	✓
MLE-STAR	Chain	Ranking	✓	✓
InternAgent-MLE	Graph	Ranking	✓	✗
KompeteAI	Tree	Ranking	✓	✗
FM Agent	Evolutionary	Ranking	✓	✗
GOME	Chain	Update	✗	✓

Reasoning as Optimization. Recent work frames LLM reasoning as an optimization signal, iteratively refining artifacts rather than enumerating candidates (Yang et al., 2023; Shinn et al., 2023). This paradigm has seen success in prompt optimization (Pryzant et al., 2023; Guo et al., 2025b) and general-purpose agents (Madaan et al., 2023), where textual "gradients" guide edits (Yuksekgonul et al., 2024; Cheng et al., 2024). However, this approach remains underexplored in MLE, likely due to the complexity of the code space compared to prompts. Existing MLE agents thus revert to search (Table 1) to hedge against generation failures. GOME bridges this gap by applying reasoning-as-optimization to the MLE domain, demonstrating that with sufficiently capable models, structured feedback can serve as a reliable gradient for complex code engineering.

3 GOME

We formulate MLE tasks as finding an optimal solution within the code space under resource constraints:

$$s^* = \arg\max_{s \in \mathcal{S}} h(\mathcal{T}, s) \quad \text{s.t.} \quad \texttt{cost}(s) \leq B, \tag{1}$$

where $\mathcal{S}$ is the space of valid ML pipelines, $h(\mathcal{T}, s)$ evaluates solution s on task $\mathcal{T}$ (e.g., accuracy), and B bounds the total computational budget.

Existing MLE agents typically organize $\mathcal{S}$ as a tree and use scalar scores to decide *which* node to expand, a form of gradient-free optimization. We propose GOME, which instead focuses on *how* to update. Inspired by gradient-based optimization, GOME extracts directional improvement signals from structured feedback via LLM reasoning. While true gradients are undefined in discrete code space, LLM reasoning serves a functionally analogous role: analyzing execution results to determine not just whether a solution improved, but *why* it improved and *what* to change next. Table 2 summarizes this analogy.

Figure 2 illustrates the framework. GOME initializes N parallel optimization traces, each beginning with a distinct hypothesis (i.e., an improvement direction for the ML pipeline) through forced diversification (§3.5). Each trace maintains local best solution $s^{*(i)}$ and experiment history $\mathcal{H}^{(i)}$, while synchronizing via a shared success memory $\mathcal{M}$. Each iteration proceeds through four stages: (1) **Execution**: run current solution and collect feedback (§3.1); (2) **Validation**: apply hierarchical checks to form structured feedback $f_t^{(i)}$ (§3.2); (3) **Memory update**: contribute successful hypotheses to $\mathcal{M}$ (Momentum accumulation) (§3.3); (4) **Reasoning**: generate the next hypothesis $\eta_{t+1}^{(i)}$ by combining local feedback with shared memory (Gradient computation) (§3.4). This local-global interplay enables collaborative optimization (§3.5), with full algorithmic details in Appendix C.

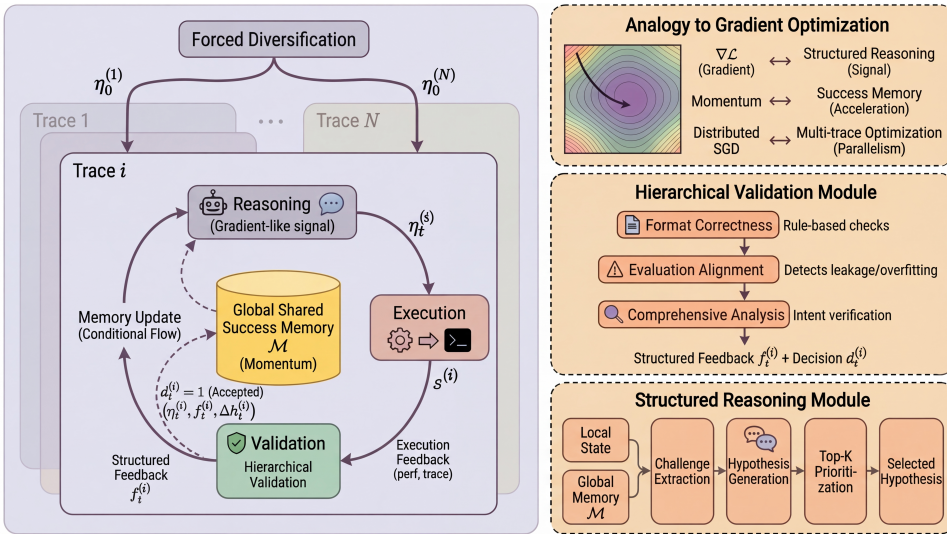


Figure 2: Overview of the GOME framework. Multiple traces optimize in parallel, synchronizing through a global shared success memory $\mathcal{M}$. Each trace iteratively executes solutions, validates improvements, updates shared memory, and reasons over local-global feedback to generate the next hypothesis. Right panels illustrate the gradient-based optimization analogy and key module details.

3.1 EXECUTION FEEDBACK

Each iteration begins with a trace executing its current solution and collecting local feedback. Executing solution $s_t^{(i)}$ on trace i produces execution feedback:

$$(\texttt{perf}_t, \texttt{trace}_t) = \texttt{Execute}(s_t^{(i)}, s^{*(i)}), \tag{2}$$

where $s^{*(i)}$ denotes the best solution found by trace i so far. The component $\texttt{perf}_t = (h_t, h^{*(i)}) \in \mathbb{R}^2$ contains current score and trace-local best score as scalar metrics; $\texttt{trace}_t$ captures execution logs(stdout/stderr,runtime logs) and code diffs relative to the trace's current best solution $s^{*(i)}$, enabling diagnosis of implementation issues.

3.2 HIERARCHICAL VALIDATION

A critical challenge in LLM-based optimization is distinguishing genuine improvements from deceptive ones. Solutions may achieve high validation scores through shortcuts such as data leakage or metric gaming. Based on execution feedback, hierarchical validation determines acceptance decision $d_t^{(i)} \in \{0, 1\}$ and diagnostic $\texttt{reason}_t$:

$$(d_t^{(i)}, \texttt{reason}_t) = \texttt{Validate}(s_t^{(i)}, \texttt{perf}_t, \texttt{trace}_t), \tag{3}$$

forming the structured feedback $f_t^{(i)} = (\texttt{perf}_t, \texttt{trace}_t, \texttt{reason}_t)$. The validation process enforces three sequential stages: (1) **Format correctness:** verifies output schema via rule-based checks; (2) **Evaluation alignment:** detects data leakage and overfitting risks via LLM analysis, e.g., accessing test labels or future information; (3) **Comprehensive analysis:** verifies whether the hypothesis achieved its intended effect and assessing code quality. The LLM makes the final accept decision $d_t^{(i)}$ based on this reasoning when $h(s_t^{(i)}) \gtrsim h(s^{*(i)})$, with $\texttt{reason}_t$ capturing the analysis from whichever stage determined the outcome, informing subsequent hypothesis generation.

3.3 SUCCESS MEMORY

Upon validation, the hypothesis and its feedback are committed to the shared success memory $\mathcal{M}$:

$$\mathcal{M}_{t+1} = \begin{cases} \mathcal{M}_t \cup \{(\eta_t^{(i)}, f_t^{(i)}, \Delta h_t^{(i)})\} & \text{if } d_t^{(i)} = 1 \\ \mathcal{M}_t & \text{otherwise} \end{cases} \tag{4}$$

where $\eta_t^{(i)}$ is the hypothesis that led to solution $s_t^{(i)}$, $f_t^{(i)}$ is the structured feedback (§3.2), and $\Delta h_t^{(i)} = h_t^{(i)} - h^{*(i)}$ records the score difference (may be negative if accepted for code quality merits). Each memory entry contains the complete context: *what* was tried (hypothesis), *what* happened (feedback), and *how much* it helped (score delta). $\mathcal{M}$ is distinct from the raw local history $\mathcal{H}^{(i)}$: in single-trace settings, it serves as a curated subset of verified successes; in multi-trace settings, it naturally extends to a global repository aggregating discoveries across all workers. Functionally analogous to momentum, this global memory biases future updates toward proven directions.

3.4 Structured Reasoning

The reasoning module generates the next improvement hypothesis, serving as the *gradient signal* that directs optimization. It first extracts challenges $\mathcal{C}_t^{(i)}$ from structured feedback $f_t^{(i)}$ and local history $\mathcal{H}^{(i)}$, shifting from exploratory analysis (early iterations) to targeted diagnosis (addressing specific execution failures). For each identified challenge c, the module generates a concrete hypothesis η_c by conditioning on the trace's complete local state $(s^{*(i)}, f_t^{(i)}, \mathcal{H}^{(i)})$.

Candidates are scored across dimensions $\mathcal{D}$ (impact, alignment, novelty, feasibility, risk-reward)[1]:

$$\texttt{score}(\eta) = \sum_{d \in \mathcal{D}} w_d \cdot \texttt{score}_d(\eta, \mathcal{M}), \tag{5}$$

where the success memory $\mathcal{M}$ (§3.3) modulates scoring: hypotheses resembling past successes gain confidence, while similar failures receive penalties. Rather than greedily selecting the top-scored hypothesis, we sample from top-k to maintain exploration diversity (in multi-trace settings, cross-trace mechanisms further guide selection, see §3.5). The selected hypothesis is finally implemented as solution $s_{t+1}^{(i)}$ via code generation and iterative refinement.

3.5 Multi-trace Optimization

Single-trace optimization risks local optima and cannot recover from poor early decisions. Gome employs N parallel traces that synchronize via $\mathcal{M}$, enabling *online knowledge sharing* where trace i's successful discoveries immediately inform traces $j \neq i$.

Forced Diversification at Initialization. To prevent redundant exploration (which renders memory sharing ineffective), we maximize initial search coverage by strictly enforcing diversity. Trace n's starting hypothesis is conditioned on all prior proposals to ensure orthogonality:

$$\eta_0^{(n)} = \texttt{GenerateHypothesis}\big(\mathcal{T}, \{\eta_0^{(j)}\}_{j<n}\big), \tag{6}$$

forcing the swarm to investigate distinct regions of the solution space from the outset.

Cross-trace Hypothesis Selection. In multi-trace settings, each trace constructs a candidate pool from three complementary sources:

$$\mathcal{H}_{\text{cand}} = \{\eta_c\}_{c \in \mathcal{C}_t^{(i)}} \cup \{\eta^\star\} \cup \texttt{Sample}(\mathcal{M}), \tag{7}$$

where $\{\eta_c\}$ are hypotheses generated by the current trace (§3.4), $\eta^\star$ is the hypothesis from the highest-scoring iteration in $\mathcal{M}$, and $\texttt{Sample}(\mathcal{M})$ retrieves hypotheses via a probabilistic kernel considering embedding similarity and score diffs. An LLM-based selector then produces the final hypothesis:

$$\eta_{t+1}^{(i)} = \texttt{LLMSelect}(\mathcal{H}_{\text{cand}}, \{f_\eta\}_{\eta \in \mathcal{M}}), \tag{8}$$

where $\{f_\eta\}$ denotes the associated feedback stored alongside each hypothesis. The selector has flexibility to *select*, *modify*, or *generate* based on the candidate pool and historical outcomes.

[1] Weights: $(0.4, 0.2, 0.2, 0.1, 0.1)$ respectively.

3.6 Robust Implementation

We employ several mechanisms to ensure reliable optimization: (1) **Development-evaluation separation**: initial iterations run on a small data subset with rapid debug loops, where failed code triggers iterative debugging that analyzes error messages and proposes targeted fixes; final validation executes on full data. (2) **Multi-seed selection**: for final submission, we rerun top-k candidates with multiple seeds and select the best-performing result, reducing variance from stochastic LLM outputs. (3) **Adaptive time management**: an LLM-based module extends time budget based on task complexity and failure patterns, avoiding premature termination on difficult tasks.

4 Experiments

We evaluate Gome on MLE-Bench under a rigorous closed-world protocol to validate our claim that gradient-based optimization scales more effectively than search as reasoning capability improves. Additional experiments including protocol comparison and cost analysis are provided in Appendix A.

4.1 Experiment Setup

Benchmark. All experiments are conducted on MLE-Bench (Chan et al., 2024), a comprehensive benchmark comprising 75 Kaggle competitions for evaluating AI agents on machine learning engineering tasks, categorized by complexity into low, medium, and high. MLE-Bench-Lite consists of 22 low-complexity tasks for efficient evaluation. We report any-medal rate as the primary indicator.

Implementation. We evaluate Gome using three frontier LLMs representing a spectrum of reasoning capabilities: DeepSeek-R1 (Guo et al., 2025a), o3 (OpenAI, 2025a), and GPT-5 (OpenAI, 2025b), ordered by increasing reasoning strength. All experiments run on 12 vCPUs, 220GB RAM, and 1 NVIDIA V100 GPU with a 12-hour budget, averaging results over three random seeds.

Baselines. We benchmark against leading closed-world agents: MLAB (Huang et al., 2023), OpenHands (Wang et al., 2024), AIDE (Jiang et al., 2025), AIRA (Toledo et al., 2025), and ML-Master (Liu et al., 2025). We re-evaluate ML-Master under identical constraints to ensure fairness, while adopting reported results for the other baselines.

4.2 Main Results

Gome establishes a new SOTA on MLE-Bench. As detailed in Table 3, Gome outperforms all closed-world baselines, achieving a peak any-medal rate of **35.1%** with GPT-5. Under identical constraints (12h, V100), Gome matches ML-Master on weaker reasoners but pulls ahead significantly with stronger models, widening the gap to **+11.1 percentage points** with GPT-5 and improving Gold rate. This widening gap is consistent with our hypothesis (validated across 10 models in Section 5). Furthermore, Gome matches the performance of AIRA despite operating under stricter resource constraints (**half the time**, weaker GPU), highlighting the efficiency of gradient-based optimization.

Accurate gradient signals drive rapid convergence. On MLE-Bench-Lite (Table 4), Gome achieves a **68.2%** medal rate, matching the SOTA open-world method (Leeroo) despite lacking external retrieval (see Appendix A.3 for full comparison). This confirms that for tractable tasks, internal diagnostics provide high-fidelity gradients sufficient for rapid convergence (Nesterov et al., 2018). Conversely, the plateau on high-complexity tasks across all methods indicates a "noisy gradient" regime where reasoning limits are reached, suggesting that further gains require restoring signal accuracy through stronger models or external knowledge.

Beyond benchmark evaluation, we validate Gome on a live Kaggle competition, demonstrating its ability to discover non-trivial feature engineering strategies autonomously (Appendix D.3).

4.3 Ablation Study

To validate Gome's design, we ablate its three core components: (1) **w/o Structured Reasoning**, replacing diagnostic analysis with simple error-correction prompts; (2) **w/o Success Memory**, re-

Table 3: Percentage of achieving any medals across different ML task complexity levels (left) and other evaluation dimensions (right) on MLE-Bench. Reporting results are mean ± SEM over 3 seeds. Valid, Median+, and Gold indicate the percentage of submissions with valid score, above median score, and gold medal. Best performances are marked in **bold**; second best are underlined. †GPU configuration from official MLE-Bench evaluation setup.

			Medal rate in different complexity (%)				Other evaluation dimensions (%)		
Agent	**Time**	**GPU**	Low	Medium	High	All	Valid	Median+	Gold
MLAB									
gpt-4o-24-08	24h	A10†	4.2±1.5	0.0±0.0	0.0±0.0	1.3±0.5	44.3±2.6	1.9±0.7	0.8±0.5
OpenHands									
gpt-4o-24-08	24h	A10†	11.5±3.4	2.2±1.3	1.9±1.9	5.1±1.3	52.0±3.3	7.1±1.7	2.7±1.1
AIDE									
gpt-4o-24-08	24h	A10†	19.0±1.3	3.2±0.5	5.6±1.0	8.6±0.5	54.9±1.0	14.4±0.7	5.0±0.4
o1-preview	24h	A10†	34.3±2.4	8.8±1.1	10.0±1.9	16.9±1.1	82.8±1.1	29.4±1.3	9.4±0.8
AIRA									
o3	24h	H200	55.0±1.5	**22.0±1.2**	21.7±1.1	31.6±0.8	97.5±0.3	**45.5±0.8**	**17.3±0.4**
ML-Master									
DeepSeek-R1	12h	V100	47.0±6.6	9.7±2.3	20.0±3.8	22.7±0.8	91.6±0.4	32.9±0.4	12.4±0.4
o3	12h	V100	42.4±4.0	12.3±1.8	20.0±0.0	22.7±1.3	94.2±0.4	31.5±2.2	13.3±0.8
GPT-5	12h	V100	51.5±3.0	10.5±2.6	17.8±2.2	24.0±2.3	**98.2±0.4**	34.7±0.4	12.4±1.6
Gome (ours)									
DeepSeek-R1	12h	V100	50.0±0.0	9.7±0.9	20.0±0.0	23.4±0.4	88.9±0.4	35.1±0.4	12.4±0.9
o3	12h	V100	59.1±4.5	20.2±0.9	**22.2±2.2**	32.5±1.2	94.2±0.4	43.1±0.9	**17.3±0.8**
GPT-5	12h	V100	**68.2±2.6**	21.1±1.5	**22.2±2.4**	**35.1±0.4**	96.0±0.0	45.3±0.0	16.4±0.8

Table 4: Performance comparison on MLE-Bench-Lite. Best performances are marked in **bold**. †Methods using open-world protocol with external resource access.

Agent	**Any-Medal (%)**
KompeteAI† (Gemini-2.5-flash,6h)	51.5±1.5
AIRA (o3,24h)	55.0±1.5
InternAgent-MLE† (DeepSeek-R1,12h)	62.1±3.0
FM Agent† (Gemini-2.5-pro,24h)	62.1±1.5
Thesis† (GPT-5-Codex,24h)	65.2±1.5
Leeroo† (Gemini-3-pro,24h)	**68.2±2.6**
Gome (GPT-5,12h)	**68.2±2.6**

moving the global repository of proven patterns; and (3) **w/o Multi-trace Optimization**, reducing to single-trajectory optimization without cross-trace knowledge sharing.

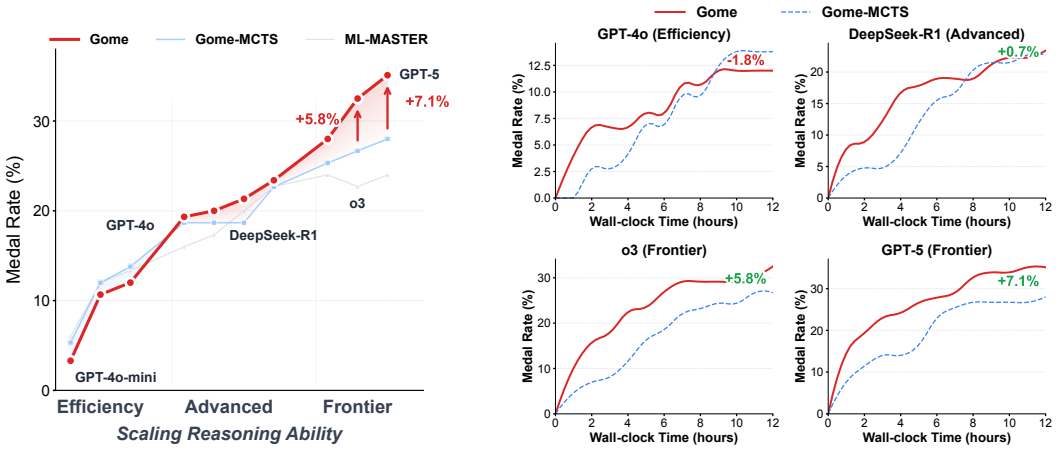
As shown in Table 5, removing any component compromises the system's overall robustness, particularly the primary Any-Medal rate. The **Structured Reasoning** ablation causes the most severe degradation, with improvement rate dropping from 41.1% to 22.6%, indicating that without diagnostic analysis, most optimization steps fail to produce valid improvements. Removing **Success Memory** shows a different failure mode: while per-iteration success remains decent, the lack of momentum from proven patterns leads to redundant exploration, reducing medal rate by 6.2%. Similarly, **Multi-trace Optimization** is essential for diversity: its removal degrades final performance despite high local improvement rates, suggesting that cross-trace sharing helps the swarm escape the local optima that trap single-trace agents. All three components contribute meaningfully (Table 2).

5 Scaling Analysis

To validate our claim that gradient signal quality depends on reasoning capability, we evaluate Gome across three tiers: Efficiency (e.g., GPT-4o-mini), Advanced (e.g., DeepSeek-R1), and Frontier (e.g.,

Table 5: Ablation study on GOME components (GPT-5, 12h budget, averaged over 3 runs). **Any Medal**: percentage achieving any medal; **Gold**: gold medal rate; **Impr.**: improvement rate, the percentage of iterations where the proposed solution passes hierarchical validation. **IC**: Spearman correlation between validation and test improvements.

Configuration	Any Medal (%)	Gold (%)	Impr. (%)	IC
GOME (full)	35.1	16.4	41.1	0.92
w/o Structured Reasoning	25.8	13.3	22.6	0.83
w/o Success Memory	28.9	16.9	36.2	0.87
w/o Multi-trace Optimization	32.4	15.1	41.3	0.88



(a) **Reasoning Scaling Laws.** As model capability increases from Efficiency to Frontier tiers, GOME's advantage over search-based baselines widens significantly.

(b) **Convergence Dynamics.** GOME (solid red) exhibits rapid early convergence, while MCTS (dashed blue) starts slower but catches up on weaker models. On Frontier-tier models, GOME maintains its advantage throughout.

Figure 3: **Scaling analysis.** (a) As reasoning capability improves, gradient-based optimization increasingly outperforms search baselines. (b) On weaker models, rapid early convergence is followed by a plateau due to noisy feedback; stronger reasoning enables sustained improvement throughout the 12h budget.

o3, GPT-5)[2]. We compare against *GOME-MCTS*, a controlled variant replacing gradient updates with tree search within identical infrastructure, isolating the optimization strategy impact.

Figure 3a reveals a distinct phase transition. On Efficiency-tier models, GOME lags behind: noisy gradients from weak reasoning favor exhaustive search. However, a **crossover** emerges as capability increases—search gains plateau while GOME accelerates, widening the gap to +5.8% on o3 and +7.1% on GPT-5. Figure 3b **traces this divergence temporally**. GOME exhibits rapid early convergence across all tiers, but sustainability differs: on weaker models, unreliable gradients cause premature **stagnation**, allowing MCTS to eventually catch up; on Frontier models, accurate signals enable GOME to maintain its lead throughout, achieving both faster convergence and superior final performance.

These results validate our central claim: gradient-based optimization increasingly outperforms tree search as reasoning capability improves. More broadly, they reveal a fundamental divergence in scaling properties: tree search scales with *inference compute* (traversing more nodes), while gradient-based optimization scales with *model capability* (reasoning better). Given the rapid advancement of reasoning-oriented LLMs, this positions gradient-based optimization as the increasingly favorable paradigm for MLE agents.

[2]See Appendix A.1 for complete tier definitions.

6 Conclusion

We presented Gome, reframing MLE agents through gradient-based optimization. By treating LLM reasoning as gradient signals, success memory as momentum, and multi-trace execution as distributed optimization, Gome establishes a principled correspondence between classical optimization and agentic ML engineering.

Our experiments demonstrate state-of-the-art performance on MLE-Bench, while scaling analysis reveals a more fundamental finding: tree search scales with inference compute through exhaustive exploration, while Gome scales with model capability through increasingly accurate gradient signals. As foundation models advance in reasoning ability, this positions gradient-based optimization as an increasingly favorable paradigm.

This work opens a new design axis for MLE agents: rather than engineering more sophisticated search strategies, future systems may benefit from investing in gradient quality through richer feedback signals and stronger diagnostic reasoning. We release our codebase and GPT-5 execution traces to support reproducibility and future research.

Reproducibility Statement

To ensure reproducibility, we provide comprehensive implementation details throughout the paper and supplementary materials. The complete source code for Gome, including all modular components (structured reasoning, success memory, multi-trace optimization), is available as anonymous supplementary material. Detailed algorithmic procedures are described in Section 3 and Appendix C. Our experimental setup is fully specified in Section 4 and Appendix B, including hardware environment (V100 GPU, 12-hour budget), and evaluation protocol on MLE-Bench. All reported results represent mean $\pm$ SEM across three independent runs with different random seeds. Upon acceptance, we will publicly release the complete codebase with documentation and execution traces.

Ethics Statement

Our work on Gome aims to augment machine learning engineering practice while upholding scientific integrity. We conduct rigorous evaluation on MLE-Bench, a publicly available benchmark of appropriately licensed Kaggle competitions. We do not use sensitive or personally identifiable data. We acknowledge that autonomous code generation carries risks of misuse; we encourage human oversight for any deployment and design safeguards against unauthorized credential usage or data leakage. We report results transparently, including limitations and negative findings, and ensure proper attribution to original dataset creators and prior work.

References

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

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

Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. Aide: Ai-driven exploration in the space of code. *arXiv preprint arXiv:2502.13138*, 2025.

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

Stepan Kulibaba, Artem Dzhalilov, Roman Pakhomov, Oleg Svidchenko, Alexander Gasnikov, and Aleksei Shpilman. Kompeteai: Accelerated autonomous multi-agent system for end-to-end pipeline generation for machine learning problems. *arXiv preprint arXiv:2508.10177*, 2025.

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

InternAgent-Team. Internagent-mle: Navigating fine-grained optimization for coding agent. https://openreview.net/pdf?id=sNjos7jJxb, 2025.

Annan Li, Chufan Wu, Zengle Ge, Yee Hin Chong, Zhinan Hou, Lizhe Cao, Cheng Ju, Jianmin Wu, Huaiming Li, Haobo Zhang, et al. The fm agent. *arXiv preprint arXiv:2510.26144*, 2025.

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

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652, 2023.

Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. Automatic prompt optimization with" gradient descent" and beam search. *arXiv preprint arXiv:2305.03495*, 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Evoprompt: Connecting llms with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2025b.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024.

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

OpenAI. Introducing openai o3 and o4-mini, 2025a. URL https://openai.com/index/introducing-o3-and-o4-mini/. Accessed: 2025-12-22.

OpenAI. Introducing gpt-5, 2025b. URL https://openai.com/index/introducing-gpt-5/. Accessed: 2025-12-22.

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

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

Yurii Nesterov et al. *Lectures on convex optimization*, volume 137. Springer, 2018.

Artificial Analysis Team. Artificial Analysis: Independent Benchmarks and Performance Landscape of AI Models, 2025. URL https://artificialanalysis.ai/. Accessed: 2025-12-28.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*, 2023.

Paul Gauthier. Aider LLM Leaderboards: Code Editing and Refactoring Benchmarks. `https://github.com/Aider-AI/aider`, 2025. URL `https://aider.chat/docs/leaderboards/`. Accessed: 2025-12-28.

Akira Isihara. *Statistical physics*. Academic Press, 2013.

APPENDIX

A MORE EXPERIMENTAL ANALYSIS

A.1 EXTENDED SCALING ANALYSIS

This section provides detailed per-model results supporting the scaling analysis in Section 5. We categorize models into three capability tiers following the stratification established by the Artificial Analysis Intelligence Index[3], SWE-bench, and Aider leaderboard[4] (Artificial Analysis Team, 2025; Jimenez et al., 2023; Gauthier, 2025): **Efficiency** models are optimized for throughput and cost; **Advanced** models represent balanced capability-efficiency trade-offs; **Frontier** models define current state-of-the-art reasoning.

Table 6: Complete per-model scaling results. All values are any-medal rates (%) on MLE-Bench. **Bold** indicates the best performance per model. Runs indicates the number of independent trials.

Model	GOME	GOME-MCTS	ML-Master	Runs
Efficiency Tier Models				
GPT-4o-mini	3.3	5.3	**6.0**	2
GPT-5-nano	10.7	**12.0**	**12.0**	2
GPT-4o	12.0	**13.8**	13.3	3
Advanced Tier Models				
GPT-5-mini	**19.3**	18.7	16.0	2
Grok-4	**20.0**	18.7	17.3	2
Grok-4.1	**21.3**	18.7	20.0	2
DeepSeek-R1	**23.4**	22.7	22.7	3
Frontier Tier Models				
DeepSeek-v3.2	**28.0**	25.3	24.0	2
o3	**32.5**	26.7	22.7	3
GPT-5	**35.1**	28.0	24.0	3

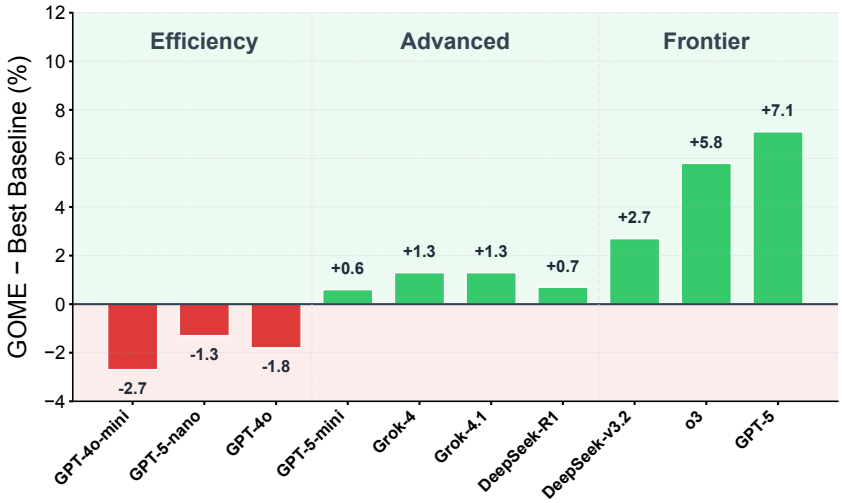


Figure 4: Performance gap between GOME and the best baseline across model capability tiers. Negative values (red) indicate GOME underperforms; positive values (green) indicate GOME outperforms. The crossover occurs at the Advanced tier, with the gap widening progressively in the Frontier tier.

[3]https://artificialanalysis.ai/

[4]https://aider.chat/docs/leaderboards/

Table 6 and Figure 4 report the complete results for all ten models evaluated in our scaling analysis. On Efficiency-tier models, GOME underperforms the best baseline consistently, with gaps ranging from -1.3% to -2.7%. This confirms that weaker models produce unreliable gradient signals. The high sensitivity of gradient-based optimization causes it to be confidently misguided by hallucinated diagnostics, while search-based methods remain robust through exhaustive exploration. The crossover emerges in the Advanced tier, where all four models show positive gaps ranging from $+0.6\%$ to $+1.3\%$. The advantage is modest but consistent, suggesting that mid-capability models begin producing sufficiently accurate gradient signals to offset the exploration benefits of tree search. At the Frontier tier, GOME's advantage becomes substantial and the gap widens progressively from $+2.7\%$ to $+7.1\%$, demonstrating that stronger reasoners amplify GOME's relative gains rather than merely maintaining them.

These results validate our central claim: gradient-based optimization exhibits fundamentally different scaling properties from tree search. The performance gap transitions from negative to positive and continues to widen as model capability increases, implying that gradient-based optimization will become increasingly favorable as foundation models advance.

A.2 Additional Ablations

Our main experiments use a 12-hour time budget (half of MLE-Bench's standard 24-hour allocation) for computational efficiency. To examine whether GOME continues to improve with the full time budget, we select o3 and GPT-5 as representative Frontier-tier models and extend to 24 hours.

Table 7: GOME performance under half-time (12h) and full-time (24h) budgets (averaged over 3 runs). All values are any-medal rates (%) on MLE-Bench. See Table 8 for full results.

Model	12h	24h	Improvement
o3	32.5	36.4	+3.9 (+12.0%)
GPT-5	35.1	40.4	+5.3 (+15.1%)

Table 7 shows that GOME continues to benefit from additional optimization time without plateauing. Both models achieve substantial gains when doubling the time budget, with o3 improving by 12.0% and GPT-5 by 15.1% relatively. Notably, the stronger model (GPT-5) exhibits larger improvements, consistent with our scaling claim: higher-quality gradient signals enable more sustained optimization progress. This suggests that gradient-based optimization has not yet saturated its potential on current Frontier models, and further gains may be achievable with increased compute budgets.

A.3 Impact of External Knowledge Access

Table 3 presents results under two evaluation protocols. Methods marked with ✓ operate in a closed-world setting, where agents cannot access external knowledge sources during evaluation but may employ carefully designed prompts, multi-stage pipelines, and framework-level optimizations. Methods marked with ✗ operate in an open-world setting, where agents can additionally leverage external resources such as web search, documentation retrieval, and curated solution libraries. We include both for comprehensive comparison.

Rationale for Closed-World Evaluation. Our main experiments (Section 4) adopt the closed-world protocol for three reasons. First, it isolates architectural contributions from confounding factors, ensuring that performance differences can be attributed to the optimization framework rather than retrieval quality or knowledge source coverage. Second, it enables fair comparison across methods, as open-world results depend heavily on which external APIs are available, how retrieved content is filtered, and the quality of curated knowledge bases. Third, it directly tests our core claim that improved LLM reasoning translates to better gradient signals, whereas external knowledge access would obscure this relationship by providing an alternative path to performance gains.

Performance Analysis. External knowledge access shows varying impact across task complexity. On low-complexity tasks, GOME achieves the highest medal rate across both protocols, suggesting that strong reasoning can fully compensate for the absence of external resources on well-defined

problems. The gap widens on medium and high-complexity tasks, where open-world methods benefit from domain-specific knowledge retrieval for specialized competitions. Despite this inherent disadvantage, GOME remains highly competitive, achieving top-three overall performance while operating under strictly closed-world conditions with shorter time budgets and weaker hardware than leading open-world methods.

Table 8: Percentage of achieving any medals across different ML task complexity levels on MLE-Bench. Reporting results are mean ± SEM over 3 seeds. Best performances are marked in **bold**; second best are underlined; third best are marked with ⋆. "–" indicates results not reported in the original work. ‡GPU configuration from official MLE-Bench evaluation setup.

				Medal rate in different complexity (%)			
Agent	**w/o Ext. Know.**	**Time**	**GPU**	Low	Medium	High	All
MLAB							
gpt-4o-24-08	✓	24h	A10‡	4.2±1.5	0.0±0.0	0.0±0.0	1.3±0.5
OpenHands							
gpt-4o-24-08	✓	24h	A10‡	11.5±3.4	2.2±1.3	1.9±1.9	5.1±1.3
AIDE							
gpt-4o-24-08	✓	24h	A10‡	19.0±1.3	3.2±0.5	5.6±1.0	8.6±0.5
o1-preview	✓	24h	A10‡	34.3±2.4	8.8±1.1	10.0±1.9	16.9±1.1
AIRA							
o3	✓	24h	H200	55.0±1.5	22.0±1.2	21.7±1.1	31.6±0.8
ML-Master							
DeepSeek-R1	✓	12h	V100	47.0±6.6	9.7±2.3	20.0±3.8	22.7±0.8
o3	✓	12h	V100	42.4±4.0	12.3±1.8	20.0±0.0	22.7±1.3
GPT-5	✓	12h	V100	51.5±3.0	10.5±2.6	17.8±2.2	24.0±2.3
KompeteAI							
Gemini-2.5-flash	✗	6h	A100	51.5±1.5	–	–	–
Neo							
Multi Frontier-Tier Models	✗	36h	A100	48.5±1.5	29.8±2.3⋆	24.4±2.2	34.2±0.9
InternAgent-MLE							
DeepSeek-R1	✗	12h	A800	62.1±3.0	26.3±2.6	24.4±2.2	36.4±1.2
MLE-STAR							
Gemini-2.5-pro	✗	12h	8*V100	63.6±6.0	–	–	–
FM Agent							
Gemini-2.5-pro	✗	12h	A800	62.1±1.5	36.8±1.5	**33.3±0.0**	43.6±0.8
Thesis							
GPT-5-Codex	✗	24h	H100	65.2±1.5⋆	**45.6±7.2**	31.1±2.2	**48.4±3.6**
GOME (ours)							
DeepSeek-R1	✓	12h	V100	50.0±0.0	9.7±0.9	20.0±0.0	23.4±0.4
o3	✓	12h	V100	59.1±4.5	20.2±0.9	22.2±2.2	32.5±1.2
o3	✓	24h	V100	63.6±2.6	27.2±2.3	24.4±2.2	36.4±0.4
GPT-5	✓	12h	V100	68.2±2.6	21.1±1.5	22.2±2.4	35.1±0.4
GPT-5	✓	24h	V100	**71.2±1.5**	28.1±0.9	26.7±0.0⋆	40.4±0.9⋆

A.4 COST ANALYSIS

Table 9 reports API pricing and average cost per competition for Frontier-tier models under the 12-hour evaluation budget.

Model	**Input**	**Output**	**Cost/Task**	**Medal Rate**
DeepSeek-v3.2	$0.28 ($0.028)	$0.42	$1.89	28.0%
GPT-5	$1.25	$10.00	$20.74	35.1%
o3	$2.00	$8.00	$26.53	32.5%

Table 9: API pricing (per million tokens) and average cost per competition for Frontier-tier models. Cost/Task is averaged over 3 independent runs. Parenthetical value indicates cache read price.

DeepSeek-v3.2 achieves 28.0% medal rate at $1.89 per task, offering the best cost-efficiency among Frontier models. Its pricing is an order of magnitude lower than alternatives, with cache read further reducing input cost to $0.028/M tokens for repeated context. GPT-5 achieves the highest performance (35.1%) at $20.74 per task, where the high output price ($10/M tokens) dominates the cost. o3 incurs the highest total cost ($26.53) while underperforming GPT-5, likely due to extended reasoning traces that increase token consumption without proportional performance gains. These results suggest that model selection should balance performance requirements against budget constraints, with DeepSeek-v3.2 as a strong choice for cost-sensitive deployments.

A.5 Task-type Breakdown

Table 10 presents detailed score comparison across 9 representative MLE-Bench tasks, grouped by data modality. We evaluate Gome with three frontier-tier backbone models: DeepSeek-R1 (Guo et al., 2025a), o3 (OpenAI, 2025a), and GPT-5 (OpenAI, 2025b).

Table 10: Score comparison on 9 MLE-Bench tasks across three data modalities. Best result for each task is highlighted in **bold**. ↑: higher is better; ↓: lower is better.

Task	Metric	DeepSeek-R1	o3	GPT-5
Image Tasks				
aptos2019-blindness-detection	QWK ↑	0.8938	**0.9243**	0.9158
inaturalist-2019-fgvc6	Top-1 Error ↓	0.2951	0.3201	**0.2261**
plant-pathology-2021-fgvc8	F1 ↑	**0.9100**	0.8949	0.9044
Text Tasks				
detecting-insults-in-social-commentary	AUC ↑	0.9384	**0.9452**	0.9176
jigsaw-toxic-comment-classification	AUC ↑	0.9848	0.9856	**0.9865**
spooky-author-identification	Logloss ↓	0.3291	0.2880	**0.2293**
Tabular Tasks				
nomad2018-predict-transparent-conductors	RMSLE ↓	0.0615	**0.0593**	0.0618
stanford-covid-vaccine	MCRMSE ↓	0.2289	**0.2263**	0.3365
tabular-playground-series-dec-2021	Accuracy ↑	0.9615	0.9631	**0.9632**

While GPT-5 achieves significantly higher overall medal rate (35.1%) compared to o3 (32.5%) and DeepSeek-R1 (23.4%), the task-level scores reveal a more nuanced picture. GPT-5 achieves the best score on 4 out of 9 tasks, but o3 also wins 4 tasks, and DeepSeek-R1 achieves the top score on plant-pathology despite having the lowest overall medal rate. This discrepancy arises because medal rate measures the proportion of tasks exceeding competition-specific thresholds, not absolute score superiority. GPT-5's advantage lies in consistently crossing medal thresholds across more tasks, rather than dominating every individual competition. The results suggest that different backbone models bring complementary strengths, and Gome's framework effectively harnesses these diverse capabilities regardless of the underlying LLM.

B Implementation Details

B.1 Gome Hyperparameters

Table 11 summarizes the hyperparameter configuration used in Gome experiments. Parameters are organized according to the framework components described in Section 3.

B.2 ML-Master Reproduction

We reproduce ML-Master (Liu et al., 2025) as a representative state-of-the-art tree search baseline. We obtain the official implementation from the authors' public repository[5] and adapt it to our evaluation protocol.

[5] https://github.com/sjtu-sai-agents/ML-Master

Table 11: GOME Hyperparameter Configuration.

Hyperparameter	Description	Default
LLM Configuration		
chat_model	Base LLM for reasoning	GPT-5
temperature	LLM decoding temperature	1.0
max_retry	Maximum API retry attempts	12000
retry_wait_seconds	Wait time between retries (s)	5
full_runtime	Total Running timeout (h)	12
Success Memory (§3.3)		
enable_global_memory	Enable shared success memory $\mathcal{M}$ across traces	True
memory_save_type	Content saved to memory (Full: hypothesis + feedback + score)	Full
Structured Reasoning (§3.4)		
llm_select_hypothesis	Use LLM-based hypothesis selector	True
simple_hypothesis	Enable simplified hypothesis format after initialization	True
unique_hypothesis	Enforce unique hypothesis generation within each trace	True
enable_cross_trace_sharing	Allow hypothesis sampling from other traces via $\mathcal{M}$	True
Multi-trace Optimization (§3.5)		
max_trace_num	Number of parallel traces N	4
merge_hours	Hours between trace synchronization	3
debugging_semaphore	Max concurrent debugging operations	3
running_semaphore	Max concurrent running operations	3
feedback_semaphore	Max concurrent feedback operations	1
cross_trace_diversity	Enforce diversity across traces at initialization	True
Robust Implementation (§3.6)		
coder_timeout_multiplier	Upper bound for coder timeout scaling	4×
runner_timeout_multiplier	Upper bound for runner timeout scaling	4×
timeout_increase_stage	Initial timeout increase stage	1
timeout_stage_patience	Patience before timeout escalation	2
llm_decide_longer_runtime	Use LLM to decide whether to extend execution time	True
fix_seed_and_split	Fix random seed and data split for reproducibility	True
enable_multi-seed_selection	Rerun top candidates with multiple seeds for final submission	True

Environment Configuration. The original ML-Master experiments were conducted on NVIDIA A100 GPUs. We adapt the implementation to run on NVIDIA V100 GPUs within our standardized Docker environment, ensuring fair comparison across all methods. All experiments use the same MLE-Bench Docker image used for GOME, eliminating potential confounds from environment differences. We apply a 12-hour wall-clock time budget per task, consistent with our closed-world evaluation protocol.

Model Substitution. The original ML-Master implementation uses DeepSeek-R1 and GPT-4o as backbone models. To enable scaling analysis across model capability tiers, we modify the inference module to support configurable LLM backends while preserving all other components (tree construction, node selection, and expansion strategies). This allows us to evaluate ML-Master with models ranging from GPT-4o-mini to GPT-5.

Hyperparameters. We adopt the default hyperparameters from the original implementation, including search depth, branching factor, and selection criteria. No task-specific tuning is performed to ensure a fair comparison with GOME, which also uses fixed hyperparameters across all tasks.

Reproduction Validation. Table 12 compares our reproduced results with those reported in the original paper. Minor performance differences are attributable to hardware differences (V100 vs. A100), Docker environment variations, stochastic variation across runs, and our stricter closed-world protocol that prohibits external knowledge retrieval.

Table 12: ML-Master reproduction validation on DeepSeek-R1. Original results are from Liu et al. (2025) on A100 GPUs; ours are averaged over 3 independent runs under closed-world protocol on V100 GPUs. All values are any-medal rates (%) on MLE-Bench with 12-hour budget.

Model	Hardware	Medal Rate	Source
DeepSeek-R1	A100	29.3	Official
DeepSeek-R1	V100	22.7	Our Rerun

B.3 GOME-MCTS IMPLEMENTATION

Our modular framework naturally supports diverse exploration strategies beyond gradient-based optimization. To provide a fair comparison between optimization paradigms within the same framework, we implement Monte Carlo Tree Search (MCTS), a widely-adopted algorithm in existing MLE agents such as ML-Master (Liu et al., 2025). This implementation, called GOME-MCTS, serves two purposes: (1) it validates that our framework is a general-purpose platform capable of accommodating different search algorithms, and (2) it isolates the effect of the optimization strategy from other architectural differences when comparing against external baselines.

Architectural Mapping. GOME-MCTS reuses GOME's execution feedback (§3.1), hierarchical validation (§3.2), and robust implementation modules (§3.6). The key difference lies in how optimization decisions are made: GOME-MCTS replaces structured reasoning (§3.4) with PUCT-based node selection, and replaces success memory (§3.3) with Q-value estimates accumulated through tree search. This substitution isolates the comparison to the core optimization mechanism while controlling for other architectural factors.

PUCT Selection. We adopt the PUCT (Predictor + UCT) variant with uniform prior $P(n, a) = 1$. The selection score balances exploitation and exploration:

$$U(n, a) = Q(n, a) + c_{\text{puct}} \cdot \frac{\sqrt{N(n)}}{1 + N(n, a)}, \tag{9}$$

where $Q(n, a)$ represents the average reward of taking action a from node n, $N(n, a)$ counts how many times action a has been selected from node n, and $N(n)$ is the total visit count for node n. The parameter $c_{\text{puct}} \in [0, \infty)$ controls the exploration-exploitation trade-off: $c_{\text{puct}} = 0$ yields pure exploitation (greedy selection), while larger values increasingly favor exploration of less-visited actions. We set $c_{\text{puct}} = 1.0$ in all experiments.

Reward Design. We experiment with two reward formulations. The binary reward R_{bin} simply assigns 1 for validated submissions and 0 otherwise:

$$R_{\text{bin}} = \begin{cases} 1, & \text{if validated,} \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

When a feedback score v is available, score-based reward R_{score} provides richer signals:

$$R_{\text{score}} = \begin{cases} +\tanh(v), & \text{if validated and higher is better,} \\ -\tanh(v), & \text{if validated and lower is better,} \\ -1, & \text{if rejected.} \end{cases} \tag{11}$$

The $\tanh$ transformation bounds rewards in $[-1, 1]$ while preserving score ordering. We use R_{score} in all experiments.

Backpropagation. After each rollout, rewards are backpropagated to update Q values along the trajectory using standard averaging:

$$Q(n, a) \leftarrow \frac{N(n, a) \cdot Q(n, a) + R}{N(n, a) + 1}. \tag{12}$$

Tree Expansion. Each node expansion generates $k = 3$ candidate solutions using the same LLM backbone as GOME. Unlike GOME's sequential refinement guided by structured reasoning, GOME-MCTS explores multiple branches in parallel and selects based on accumulated Q-values. The maximum tree depth is set to 10, with early termination upon achieving a medal-qualifying score.

C ALGORITHM DETAILS

C.1 ALGORITHM FOR EXECUTION FEEDBACK AND HIERARCHICAL VALIDATION

The detailed procedure for generating execution feedback and performing hierarchical validation is presented in Algorithm 1. This process ensures that the solution s_t is rigorously evaluated against the trace-local best solution s^* through a multi-stage gate mechanism before being accepted.

Algorithm 1 Execution Feedback and Hierarchical Validation

Input: current solution s_t, trace-local best solution s^*, task specification $\mathcal{T}$
Output: structured feedback f_t, accept decision d_t

▷ **Stage 1: Execution Feedback** (§3.1)
$h_t \leftarrow \texttt{Evaluate}(s_t, \mathcal{T})$ ▷ current validation score
$\texttt{perf}_t \leftarrow (h_t, h^*)$ ▷ scalar metrics: current and best scores
$\texttt{trace}_t \leftarrow (\texttt{CodeDiff}(s_t, s^*), \texttt{Logs}(s_t))$ ▷ code diffs and execution logs

▷ **Stage 2: Hierarchical Validation** (§3.2)
▷ Gate 1: Format correctness (rule-based)
if $\neg\, \texttt{FormatValid}(s_t, \mathcal{T})$ **then**
    $\texttt{reason}_t \leftarrow \texttt{DiagnoseFormat}(s_t, \mathcal{T})$
    return $((\texttt{perf}_t, \texttt{trace}_t, \texttt{reason}_t),\ 0)$
end if
▷ Gate 2: Evaluation alignment (LLM-based)
$\texttt{leakage} \leftarrow \texttt{DetectLeakage}(s_t, \mathcal{T})$ ▷ test label access, future info, etc.
if $\texttt{leakage} \neq \emptyset$ **then**
    $\texttt{reason}_t \leftarrow \texttt{DiagnoseAlignment}(s_t, \texttt{leakage})$
    return $((\texttt{perf}_t, \texttt{trace}_t, \texttt{reason}_t),\ 0)$
end if
▷ Gate 3: Comprehensive analysis (when $h_t \gtrsim h^*$)
$\texttt{reason}_t\texttt{.hyp\_verify} \leftarrow \texttt{VerifyHypothesis}(s_t, s^*, \texttt{trace}_t)$ ▷ intended effect achieved?
if $h_t \gtrsim h^*$ **then**
    $\texttt{reason}_t\texttt{.code\_quality} \leftarrow \texttt{AnalyzeCode}(s_t, s^*)$ ▷ best practices, efficiency
end if

▷ **Stage 3: Accept Decision** (LLM-based)
$f_t \leftarrow (\texttt{perf}_t, \texttt{trace}_t, \texttt{reason}_t)$
$d_t \leftarrow \texttt{LLMJudge}(f_t)$ ▷ accept based on full structured feedback
return $(f_t,\ d_t)$

C.2 ALGORITHM FOR STRUCTURED REASONING

The structured reasoning process, which guides the generation of the next hypothesis η_{t+1}, is outlined in Algorithm 2. This algorithm utilizes an adaptive weighting mechanism and leverages the global success memory $\mathcal{M}$ to prioritize hypotheses based on historical success and scenario analysis.

C.3 MULTI-TRACE OPTIMIZATION DETAILS

This section provides implementation details for the multi-trace optimization described in §3.5.

Algorithm 2 Structured Reasoning

Input: feedback f_t, local history $\mathcal{H}$, global success memory $\mathcal{M}$, best-so-far s^*, task $\mathcal{T}$
Output: selected hypothesis η_{t+1}, updated solution s_{t+1}

▷ **Stage 1: Adaptive Weighting**
$n_{\text{succ}} \leftarrow |\{e \in \mathcal{H} : d_e = 1\}|$
$n_{\text{fail}} \leftarrow |\mathcal{H}| - n_{\text{succ}}$
$\lambda \leftarrow \max(0, 3 - \lfloor (3n_{\text{succ}} + 2n_{\text{fail}})/8 \rfloor)$ ▷ decays with experience

▷ **Stage 2: Challenge Extraction**
$\mathcal{C}_t \leftarrow \emptyset$
if $\lambda > 0$ **then** ▷ early: scenario-driven
    $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \texttt{AnalyzeScenario}(s^*, \mathcal{T})$ ▷ SOTA alignment, gap identification
end if
if $\lambda < 3$ **then** ▷ later: feedback-driven
    $\mathcal{C}_t \leftarrow \mathcal{C}_t \cup \texttt{AnalyzeHistory}(f_t, \mathcal{H})$ ▷ explicit issues, persistent errors
end if

▷ **Stage 3: Hypothesis Generation**
for $c \in \mathcal{C}_t$ **do**
    $\eta_c \leftarrow \texttt{Hypothesize}(c, s^*, f_t, \mathcal{T})$ ▷ target component + reasoning
end for

▷ **Stage 4: Prioritization**
$\mathcal{D} \leftarrow \{\texttt{impact}, \texttt{alignment}, \texttt{novelty}, \texttt{feasibility}, \texttt{risk}\}$
$\mathbf{w} \leftarrow (0.4, 0.2, 0.2, 0.1, 0.1)$
for $(c, \eta) \in \mathcal{C}_t$ **do**
    $\texttt{score}(\eta) \leftarrow \sum_{d \in \mathcal{D}} w_d \cdot \texttt{score}_d(\eta, \mathcal{M})$
end for

▷ **Stage 5: Hypothesis Selection**
if single-trace **then**
    $\mathcal{H}_{\text{cand}} \leftarrow \texttt{TopK}(\{(c, \eta_c)\}, k)$
    $\eta_{t+1} \leftarrow \texttt{Sample}(\mathcal{H}_{\text{cand}})$ ▷ stochastic selection
else ▷ multi-trace (§3.5)
    $\eta^\star \leftarrow \arg\max_{\eta \in \mathcal{M}} \Delta h_\eta$ ▷ best from shared memory
    $\mathcal{H}_{\text{sim}} \leftarrow \texttt{SimilaritySample}(\mathcal{M}, f_t)$ ▷ embedding-based retrieval
    $\mathcal{H}_{\text{cand}} \leftarrow \{\eta_c\}_{c \in \mathcal{C}_t} \cup \{\eta^\star\} \cup \mathcal{H}_{\text{sim}}$
    $\eta_{t+1} \leftarrow \texttt{LLMSelect}(\mathcal{H}_{\text{cand}}, \{f_\eta\}_{\eta \in \mathcal{M}})$ ▷ select / modify / generate
end if

▷ **Stage 6: Solution Update**
$\tau \leftarrow \texttt{Sketch}(\eta_{t+1}, s^*)$ ▷ code modification plan
$s_{t+1} \leftarrow \texttt{Implement}(\tau, s^*)$ ▷ iterative refinement until executable
return (η_{t+1}, s_{t+1})

C.3.1 Candidate Hypothesis Construction

As described in §3.5, each trace constructs a candidate pool from three complementary sources:

$$\mathcal{H}_{\text{cand}} = \{\eta_c\}_{c \in \mathcal{C}_t^{(i)}} \cup \{\eta^\star\} \cup \mathcal{H}_{\text{sim}}, \tag{13}$$

where:

- $\{\eta_c\}_{c \in \mathcal{C}_t^{(i)}}$: Hypotheses generated by the current trace based on local challenges (§3.4)
- $\eta^\star$: The hypothesis from the highest-scoring iteration in the shared success memory $\mathcal{M}$
- $\mathcal{H}_{\text{sim}}$: Hypotheses sampled via the probabilistic interaction kernel

Probabilistic Interaction Kernel. Inspired by statistical physics (Isihara, 2013), we introduce a probabilistic interaction mechanism that enables controlled information exchange between traces. For each hypothesis η_c generated by the current trace, we compute an interaction potential with all hypotheses η_j in the shared memory $\mathcal{M}$:

$$U_{cj} = \alpha \cdot S_{cj} \cdot e^{-\gamma L} + \beta \cdot \tanh(\Delta_{cj}) \in [-2, 2], \tag{14}$$

where U_{cj} is the interaction potential between hypothesis η_c and historical hypothesis η_j. The parameters α and β are weights controlling the relative importance of embedding similarity S_{cj} (cosine similarity between embeddings of η_c and η_j) and score difference Δ_{cj}. The parameter γ is a decay factor based on the iteration count L.

The score difference Δ_{cj} is defined as:

$$\Delta_{cj} = \begin{cases} h_j - h^{*(i)}, & \text{if higher score is better} \\ h^{*(i)} - h_j, & \text{if lower score is better} \end{cases} \tag{15}$$

where $h^{*(i)}$ is the trace-local best score and h_j is the score associated with hypothesis η_j. The sampling distribution is computed via softmax normalization:

$$p_{cj} = \frac{\exp(U_{cj})}{\sum_k \exp(U_{ck})}, \quad \eta_{\text{sim}} \sim \text{Categorical}(p_{cj}). \tag{16}$$

This interaction potential integrates both semantic information from hypothesis text and empirical information from scores. The decay factor $e^{-\gamma L}$ applied to the similarity term reflects that the optimization trajectory is not a Markov process: the generation of later hypotheses depends on multiple previous steps. Therefore, in later stages of exploration, the weight of this component decays rapidly, so that score information plays a more dominant role, biasing selection toward empirically proven strategies.

C.3.2 Adaptive Hypothesis Selection

After constructing the candidate pool, an LLM-based selector produces the final hypothesis. The selector is not constrained to choose from the candidates; instead, it can take one of three actions:

- **Select**: Choose the most promising hypothesis from $\mathcal{H}_{\text{cand}}$
- **Modify**: Revise an existing candidate to improve it (e.g., adjust hyperparameters, refine approach)
- **Generate**: Create a new hypothesis by synthesizing insights from multiple candidates

This design aims to reduce hallucinations and stabilize outcomes across different traces, as the provided candidates serve as grounded references rather than strict constraints.

Algorithm 3 presents the cross-trace hypothesis selection process.

This adaptive selection mechanism, combined with the probabilistic interaction kernel, enables efficient cross-trace learning without sacrificing exploration diversity. When a trace falls behind the global best, it can leverage successful strategies from other traces via $\eta^\star$ and $\mathcal{H}_{\text{sim}}$; when leading, it continues local exploration while still benefiting from the shared memory. This design ensures that the system leverages collective discoveries from all parallel traces while maintaining independent optimization trajectories.

Algorithm 3 Cross-trace Hypothesis Selection

Input: local hypotheses $\{\eta_c\}_{c \in \mathcal{C}_t^{(i)}}$, success memory $\mathcal{M}$, trace-local best score $h^{*(i)}$
Output: selected hypothesis $\eta_{t+1}^{(i)}$

▷ **Step 1: Construct Candidate Pool** (§3.5)
$\eta^\star \leftarrow \arg\max_{\eta \in \mathcal{M}} \Delta h_\eta$ ▷ best from shared memory
$\mathcal{H}_{\text{sim}} \leftarrow \texttt{SampleByKernel}(\mathcal{M}, \{\eta_c\})$ ▷ probabilistic interaction
$\mathcal{H}_{\text{cand}} \leftarrow \{\eta_c\}_{c \in \mathcal{C}_t^{(i)}} \cup \{\eta^\star\} \cup \mathcal{H}_{\text{sim}}$

▷ **Step 2: LLM-based Selection**
$\eta_{t+1}^{(i)} \leftarrow \texttt{LLMSelect}(\mathcal{H}_{\text{cand}}, \{f_\eta\}_{\eta \in \mathcal{M}})$
▷ Selector can: **Select** from candidates, **Modify** existing, or **Generate** new
return $\eta_{t+1}^{(i)}$

C.4 Algorithm for Gome Main Loop

Algorithm 4 orchestrates the overall optimization process. It begins with "Initialization with Forced Diversification", where starting hypotheses are sequentially conditioned on prior proposals to maximize the initial coverage of the solution space.

The core "Parallel Optimization" then coordinates multiple concurrent traces. In each iteration, the system: (1) invokes the feedback mechanism (Algorithm 1) to evaluate the current state; (2) updates the Global Success Memory $\mathcal{M}$ upon acceptance, allowing successful strategies to be shared across traces; and (3) calls the structured reasoning module (Algorithm 2) to generate the next solution. The process concludes with Final Selection, which employs multi-seed evaluation on top candidates to mitigate variance and ensure the robustness of the final output s^*.

D Qualitative Analysis and Case Studies

We provide detailed case studies to illustrate Gome's reasoning capabilities, contrasting it with scalar-driven baselines and demonstrating its performance in real-world scenarios.

D.1 Preventing Overfitting via Hierarchical Validation

A critical limitation of gradient-free optimization is the reliance on validation scores as the sole proxy for solution quality. This often leads to overfitting or the acceptance of technically flawed solutions that happen to score well on a specific split.

We analyze Gome's overfitting detection capability on the *Stanford COVID Vaccine* task from MLE-Bench (High tier), where the goal is to predict RNA degradation rates (MCRMSE, lower is better). Across 90 optimization iterations, we identified 9 cases where validation improved but test performance degraded, which is the hallmark of overfitting.

Detection Performance. Table 13 summarizes Gome's overfitting detection results. The hierarchical validation correctly rejected 6 out of 9 overfitting cases (66.7% detection rate), preventing potentially harmful updates from being committed.

Table 13: Overfitting detection performance on Stanford COVID Vaccine.

Metric	Value
Total optimization iterations	90
Overfitting cases identified	9
Correctly rejected	6 (66.7%)
Incorrectly accepted	3 (33.3%)

Algorithm 4 GOME: Gradient-based Optimization for Machine Learning Engineering

Input: task specification $\mathcal{T}$, dataset $\mathcal{D}$, number of traces N, time budget B
Output: best solution s^*

▷ **Phase 1: Initialization with Forced Diversification** (§3.5)
$\mathcal{M} \leftarrow \emptyset$ ▷ global success memory
for $n = 1$ to N **do**
  $\mathcal{H}^{(n)} \leftarrow \emptyset$ ▷ local history for trace n
  $\eta_0^{(n)} \leftarrow \texttt{GenerateInitialHypothesis}(\mathcal{T}, \{\eta_0^{(j)}\}_{j<n})$ ▷ condition on earlier proposals
  $s_0^{(n)} \leftarrow \texttt{Implement}(\eta_0^{(n)}, \mathcal{T})$ ▷ initial solution for trace n
  $s^{*(n)} \leftarrow s_0^{(n)}$ ▷ trace-local best
end for

▷ **Phase 2: Parallel Optimization**
while $\texttt{RemainingBudget}(B) > 0$ **do**
  for each trace $n \in \{1, \ldots, N\}$ **in parallel do**

▷ Step 1: Execute and Validate (Algorithm 1)
    $(f_t^{(n)}, d_t^{(n)}) \leftarrow \texttt{FeedbackAndValidation}(s_t^{(n)}, s^{*(n)}, \mathcal{T})$

▷ Step 2: Update Success Memory (§3.3)
    if $d_t^{(n)} = 1$ **then** ▷ accepted
      $\Delta h_t^{(n)} \leftarrow h_t^{(n)} - h^{*(n)}$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{(\eta_t^{(n)}, f_t^{(n)}, \Delta h_t^{(n)})\}$
      if $h_t^{(n)} > h^{*(n)}$ **then**
        $s^{*(n)} \leftarrow s_t^{(n)}$ ▷ update trace-local best
      end if
    end if
    $\mathcal{H}^{(n)} \leftarrow \mathcal{H}^{(n)} \cup \{(s_t^{(n)}, f_t^{(n)}, d_t^{(n)})\}$

▷ Step 3: Structured Reasoning (Algorithm 2)
    $(\eta_{t+1}^{(n)}, s_{t+1}^{(n)}) \leftarrow \texttt{StructuredReasoning}(f_t^{(n)}, \mathcal{H}^{(n)}, \mathcal{M}, s^{*(n)}, \mathcal{T})$
  end for

▷ Adaptive Time Management (§3.6)
  $B \leftarrow \texttt{AdjustBudget}(B, \mathcal{M}, \{f_t^{(n)}\}_{n=1}^N)$
end while

▷ **Phase 3: Final Selection** (§3.6)
$\mathcal{S}_{\text{top}} \leftarrow \texttt{TopK}(\{s^{*(n)}\}_{n=1}^N, k)$
for $s \in \mathcal{S}_{\text{top}}$ **do**
  $\bar{h}(s) \leftarrow \texttt{MultiSeedEval}(s, \mathcal{T})$ ▷ reduce variance
end for
$s^* \leftarrow \arg\max_{s \in \mathcal{S}_{\text{top}}} \bar{h}(s)$
return s^*

Case: Overfitting Detection on Stanford COVID Vaccine (Node 32)

Task: Predict RNA degradation rates (MCRMSE, lower is better)

Hypothesis η_t: Align training objective with evaluation by redefining loss weights: for reactivity use clipped inverse-variance weights $w = \text{clip}(1/(\text{err}^2 + 10^{-6}), 0.1, 10)$; for deg_Mg_pH10 and deg_Mg_50C use constant $w = 1.0$. Replace hard SN_filter gating with soft factor (1.0 for SN=1, 0.5 for SN=0). Per-batch renormalize so each scored channel has equal total weight mass.

Metrics:

- Previous (Parent 27): $h_{\text{val}}^{*(i)} = 0.6948$, $h_{\text{test}}^{*(i)} = 0.2688$
- Current (Node 32): $h_{t,\text{val}} = 0.2945$ ($\downarrow$ 57.6% better), $h_{t,\text{test}} = 0.6382$ ($\uparrow$ 137.5% worse)

Code Changes: Kept PairAwareBiGRU architecture, features, split, inference identical to SOTA. Replaced training weights with channel-specific weighting; introduced per-batch channel renormalization; left validation metric as unweighted MCRMSE.

Observations from f_t: Current MCRMSE is 0.2945 vs. SOTA 0.2559; performance degraded by $+0.0385$. EDA indicates weak target–error association for reactivity (Pearson ≈ -0.042, Spearman ≈ 0.280), suggesting inverse-variance weighting may not benefit learning.

Hypothesis Evaluation: **REFUTED**. The redesigned loss weighting and per-batch renormalization worsened validation MCRMSE. Likely causes: weak correlation between reactivity and its error; soft SN scaling introducing noisier samples; renormalization potentially overemphasizing harder channels.

Decision: $d_t = 0$ **(REJECT)** — Despite validation appearing to improve, the structured analysis identified that the "improvement" stemmed from metric misalignment rather than genuine generalization gains.

Figure 5: Overfitting detection on Stanford COVID Vaccine. The validation score appeared to improve by 57.6%, but Gome's hierarchical validation analyzed the code changes and training dynamics to correctly reject the solution, avoiding a catastrophic 137.5% test regression.

Case Study: Loop 32. Figure 5 presents a representative case where Gome correctly rejected a deceptive improvement. The agent proposed a loss reweighting strategy that showed dramatic validation improvement (-57.6%) but would have caused catastrophic test degradation ($+137.5\%$).

Detailed Overfitting Cases. Table 14 lists all identified overfitting cases and Gome's decisions. The three incorrectly accepted cases (Nodes 40, 49, 80) all exhibited marginal validation improvements ($< 6\%$) with small test regressions ($< 6.3\%$), making them harder to distinguish from noise. In contrast, the correctly rejected cases often showed larger validation-test divergence patterns.

Table 14: All overfitting cases in Stanford COVID Vaccine. Δh_{val}: validation change (negative is improvement). Δh_{test}: test change (positive is degradation).

Loop	Component	Δh_{val}	Δh_{test}	Decision	Correct
23	Model	-63.8%	$+0.3\%$	Reject	✓
26	FeatureEng	-5.1%	$+1.4\%$	Reject	✓
32	Model	-57.6%	$+137.5\%$	Reject	✓
40	Model	-0.2%	$+6.3\%$	Accept	✗
47	Ensemble	-11.9%	$+0.4\%$	Reject	✓
49	Model	-5.8%	$+1.2\%$	Accept	✗
57	Model	-69.8%	$+0.3\%$	Reject	✓
77	FeatureEng	-0.1%	$+0.03\%$	Reject	✓
80	Model	-0.1%	$+0.03\%$	Accept	✗

Analysis. This case demonstrates why gradient-free methods that rely solely on scalar ranking are fundamentally limited: they cannot distinguish genuine improvements from deceptive ones. A score-driven agent would have accepted Loop 32's solution based on the 57.6% validation "improvement,"

leading to catastrophic test degradation. GOME's structured feedback, which analyzes code changes, training dynamics, and feature correlations, provides the semantic information necessary to detect such patterns. The 66.7% detection rate, while not perfect, represents a significant advantage over purely score-driven approaches that would accept all 9 overfitting cases (0% detection).

D.2 COMPARISON WITH MLE-STAR

MLE-STAR (Nam et al., 2025) represents a recent advancement in MLE agents, achieving 63.6% medal rate on MLE-Bench Lite through web retrieval and targeted code-block refinement. While MLE-STAR and GOME share a chain-based structure (as opposed to tree or graph topologies), they differ fundamentally in how feedback guides optimization. We analyze these differences across three dimensions.

Feedback Utilization: Ranking vs. Update. MLE-STAR employs a two-level selection process: (1) an ablation study identifies which code block has the greatest performance impact, and (2) multiple refinement plans are generated and evaluated, with the highest-scoring plan selected. Formally, given K candidate plans $\{p_k\}_{k=0}^{K-1}$ for code block c_t, MLE-STAR selects $k^* = \arg\max_k h(s_t^k)$ where $h(\cdot)$ is the validation score. This is fundamentally a *ranking* operation: scalar scores determine which pre-generated candidate survives.

In contrast, GOME uses structured reasoning to generate *directional* updates. Rather than generating multiple candidates and selecting via scores, the LLM analyzes execution feedback to produce a single improvement hypothesis with explicit direction (what to change) and magnitude (how much). The validation score serves as a stopping criterion and acceptance gate, not a selection mechanism among alternatives.

Code Block Selection: Ablation vs. Reasoning. MLE-STAR's ablation study agent $\mathcal{A}_{\text{abl}}$ generates code that systematically disables or modifies components (typically 2–3 per iteration), then compares resulting scores to identify the most impactful block. This requires additional exploratory executions per iteration and provides only *which* block matters most, not *why* it underperforms or *how* to improve it.

GOME's structured reasoning operates on execution feedback directly: training dynamics, validation curves, error patterns, and code behavior. The LLM reasons about *why* current performance is limited and *what modification* would address the underlying issue. This reasoning-first approach generates actionable hypotheses without requiring additional ablation executions.

Illustrative Comparison. Consider the overfitting case from §D.1 where validation improved 57.6% but test degraded 137.5%. This represents *subtle* overfitting that does not involve explicit data leakage detectable by rule-based checkers. Under MLE-STAR's framework, once the data leakage checker passes, the ablation study would identify the model component as impactful based on score changes. The planner would then generate multiple refinement plans, and plan selection would rely solely on validation scores, ultimately *accepting* the overfitting solution due to its apparent 57.6% improvement.

Under GOME's framework, structured reasoning analyzes training dynamics, feature correlations, and the semantic nature of the code change. This analysis identifies weak target-error correlation (Pearson ≈ -0.042), recognizing that the validation improvement stems from spurious fitting rather than genuine generalization. The hierarchical validation gate consequently rejects the update based on this reasoning rather than the misleading validation score.

Summary. Table 15 summarizes the key distinctions. While MLE-STAR advances beyond AIDE's tree search through targeted refinement, it remains within the score-driven paradigm: validation scores ultimately determine which solutions propagate. GOME shifts to reasoning-driven optimization, where LLM analysis of execution signals generates improvement direction rather than merely selecting among candidates. This distinction becomes increasingly important as model reasoning capabilities improve, enabling more accurate "gradient" estimation.

Table 15: Key distinctions between MLE-STAR and GOME. Overfitting detection rate is based on the case study in §D.1, where score-driven methods would accept all 9 overfitting cases.

Aspect	MLE-STAR	GOME
Feedback role	Ranking	Update
Plan generation	Multiple candidates	Single hypothesis
Selection mechanism	$\arg\max$ score	Reasoning gate
Block identification	Ablation study	Structured analysis
Subtle overfitting detection	0% (score-driven)	66.7% (reasoning)

D.3 ONLINE KAGGLE VALIDATION: STORE SALES FORECASTING

While MLE-Bench provides standardized evaluation, real-world data science requires navigating uncurated data, ambiguous constraints, and complex domain logic. To validate GOME's effectiveness in this setting, we deployed the framework on an active Kaggle competition: *Store Sales - Time Series Forecasting*.[6] This competition requires forecasting unit sales for thousands of product families sold at Corporación Favorita stores in Ecuador over a 15-day horizon. The dataset includes store metadata, oil prices (Ecuador's economy is oil-dependent), promotional information, and a calendar of holidays with complex transfer rules. The evaluation metric is Root Mean Squared Logarithmic Error (RMSLE, lower is better). GOME achieved a final leaderboard RMSLE of 0.431, ranking in the **top 15%** of all participants.

Optimization Trajectory. Table 16 summarizes the optimization process. Starting from an initial baseline (validation RMSLE 0.485), GOME progressively refined the solution over 48 iterations. The best validation score (0.318) was achieved at Loop 37, which translated to a leaderboard RMSLE of 0.431, outperforming approximately 85% of human participants.

Table 16: **Optimization trajectory on the Store Sales competition.** All intermediate scores are internal validation RMSLE (15-day holdout). The trajectory shows steady improvement from 0.485 to 0.318 over 37 iterations, with the gate mechanism rejecting degrading modifications (Loops 30, 43). The final submission achieved a leaderboard RMSLE of 0.431. Note that the human leaderboard leader (0.378) employs multi-level blending of other participants' submissions.

Loop	Val RMSLE	Key Modification	Outcome
0	0.485	Initial baseline	Accepted
2	0.354	Two-stage hurdle architecture	Accepted
7	0.327	Rolling statistics & lag features	Accepted
28	0.318	Store-local holiday processing	Accepted
37	**0.318**	Horizon bucketing & calibration	**Accepted (best)**
30	(0.547)	Naive ensemble strategy	Rejected
43	(0.559)	Unstable pipeline modification	Rejected
–	*0.378*	*Human best*	–

This trajectory demonstrates GOME's effective gate mechanism. At Loop 30, an ensemble strategy caused severe metric degradation (0.318→0.547), which was correctly rejected. Similarly, Loop 43's workflow modification destabilized the pipeline (0.559) and was rejected. The system preserved the robust solution from Loop 37 throughout subsequent iterations.

Architectural Comparison. To contextualize GOME's performance, we analyzed the codebases of high-ranking public notebooks. As shown in Figure 6, a striking distinction emerges. Many top-tier public solutions employ *second-level blending*—scripts that optimally weight prediction files submitted by other competitors using hand-tuned coefficients and rank-based corrections. These function as meta-optimization tools for existing outputs rather than building models from raw data.

[6] https://www.kaggle.com/competitions/store-sales-time-series-forecasting

(a) Top-Tier Human Solution: Post-Hoc Blending

```
# 1. Ingest pre-calculated prediction files from other participants (No training)
def read(dk, i):
    FiN = dk["path"] + dk["subm"][i]["name"] + ".csv"  # e.g., "0.37982.csv"
    return pd.read_csv(FiN)
# 2. Hard-coded weights for linear combination (manually tuned)
params = {
    'subwts': [+14, -1, -5, -8] / 100,  # correction weights by rank
    'subm': [{'name': '0.37982', 'weight': 0.30}, {'name': '0.37984', 'weight': 0.10},
             {'name': '0.38006', 'weight': 0.30}, {'name': '0.38040', 'weight': 0.30}]
}
# 3. Compute weighted sum (Pure ensembling of existing outputs)
def correct(x, cols, weights, subwts):
    rank_indices = [x['alls'].index(c) for c in cols]  # rank-based reweighting
    return sum([
        x[cols[j]] * (weights[j] + subwts[rank_indices[j]])
        for j in range(len(cols))
    ])
```

(b) Gome Solution: Ab Initio Modeling

```
# 1. Feature Engineering from Raw Data (Reasoning-driven)
buckets = {'H1': [1,2,3], 'H2': [4,5,6,7], 'H3': [8,9,10,11,12,13,14,15]}
# 2. Two-Stage Hurdle Model Training (Learning from scratch)
for bname, horizons in buckets.items():
    clf = LGBMClassifier(n_estimators=4000, num_leaves=384, learning_rate=0.03)
    clf.fit(X_clf_tr, y_clf_tr, sample_weight=w_recency)

    reg = LGBMRegressor(n_estimators=6000, num_leaves=384, learning_rate=0.03)
    reg.fit(X_reg_tr, y_reg_tr, sample_weight=w_recency)

    calibrator = PooledPlattCalibrator(max_h=max(horizons))
    calibrator.fit(clf.predict_proba(X_val)[:,1], h_val, y_val)
# 3. Model Inference: P(nonzero) * E[sales | nonzero]
yhat = calibrator.transform(clf.predict_proba(X)[:,1], h) * np.expm1(reg.predict(X))
```

Figure 6: **Codebase comparison.** (a) High-ranking public notebooks often use blending scripts that aggregate prediction CSVs submitted by other participants, a form of meta-optimization that, while effective for leaderboard climbing, involves no model training. Gome constructs the full pipeline from raw relational tables under the closed-world protocol without any external knowledge, including feature engineering, horizon-bucketed model training, and Platt calibration.

In contrast, Gome functions as an autonomous data scientist, constructing the entire pipeline from raw relational tables: (1) performing extensive feature engineering including horizon bucketing, rolling statistics, and store-local holiday processing; (2) training two-stage hurdle models (classifier for zero/non-zero, regressor for magnitude) with recency-weighted sampling; and (3) implementing pooled Platt calibration for probability adjustment. This confirms that Gome achieves expert-level performance through genuine architectural reasoning rather than aggregation of existing solutions.

D.4 Error Analysis

To understand Gome's limitations, we conduct a detailed case study on the *stanford-covid-vaccine* task (a high-complexity task in MLE-Bench) using GPT-5 as the backbone model.

Table 17 summarizes the failure mode distribution across 90 total iterations. We classify unsuccessful iterations into four categories: (1) **Gradient Hallucination**, where the reasoning module produces confident but incorrect improvement directions; (2) **Implementation Failures**, where correct hypotheses fail during code generation due to API misuse or syntax errors; (3) **Resource Constraints**, where solutions exceed memory limits or time budget B; and (4) **Validation Blindspots**, where validation score improves but test performance degrades.

Gradient hallucination dominates at 38.9%, where the reasoning module generates plausible but incorrect optimization directions. This rate is expected to decrease as LLM reasoning capabilities improve, consistent with our scaling claim (§5). Resource constraints account for 23.3% of failures

Table 17: Failure mode distribution on stanford-covid-vaccine (90 iterations, GPT-5).

Category	**Count**	**Percentage**
Successful Updates	*31*	*34.4%*
Gradient Hallucination	35	38.9%
Resource Constraints	21	23.3%
Validation Blindspots	3	3.3%
Total Failures	59	65.6%

due to hardware limitations (single V100 GPU, 32GB memory) and time budget restrictions. Many proposed solutions exceeded available memory or the time budget, particularly for this RNA sequence modeling task with computationally intensive architectures. Allocating more powerful hardware or extending time budgets would likely reduce this failure mode. Validation blindspots (3.3%) are rarer but harder to address, as they evade the gate mechanism entirely: the proposed modification improves validation metrics but degrades test performance due to distribution shifts. More sophisticated validation strategies may help mitigate this issue in future work.

E Code and Artifacts

We release our codebase and solution traces to facilitate reproducibility and future research. The repository includes the full implementation of Gome with all prompt templates, as well as a representative subset of 40 optimization trajectories (in JSON format) from GPT-5 experiments due to file size constraints.

Github repository: `https://github.com/microsoft/RD-Agent`