OPT-BENCH: Evaluating the Iterative Self-Optimization of LLM Agents in Large-Scale Search Spaces

Anonymous ACL submission

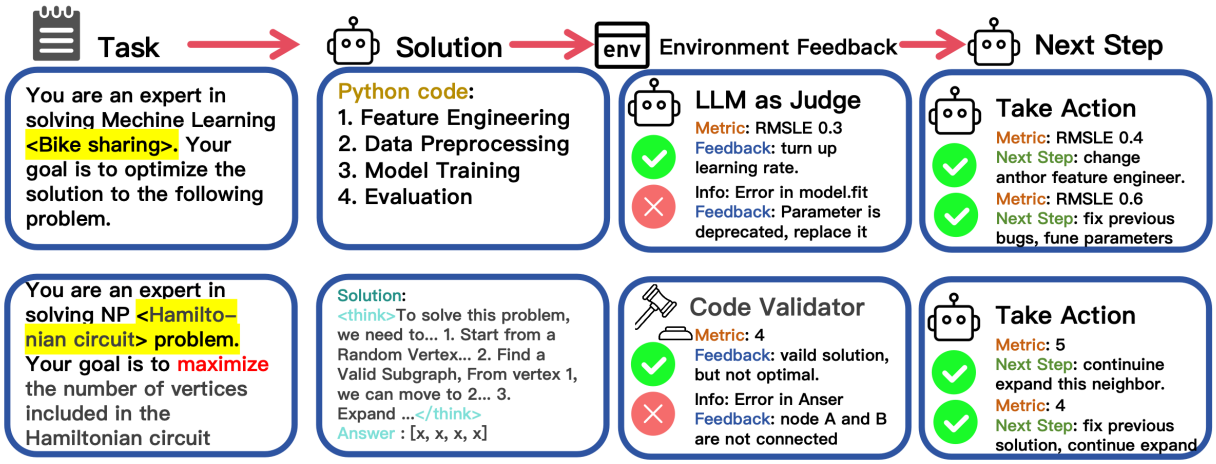

Figure 1: **OPT-BENCH framework.** This framework evaluates **Iterative Self-Optimization** by integrating two distinct reasoning modalities: **Continuous Parametric Optimization** for machine learning tasks (Top), and **Discrete Combinatorial Reasoning** for NP-hard problems (Bottom). In both settings, the agent leverages **Environmental Feedback** to guide its trajectory of improvement and debugging, progressively bridging the gap between initial hypotheses and optimal solutions through intrinsic reasoning.

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities in reasoning and tool use. However, the fundamental cognitive faculties essential for problem-solving—perception, reasoning, and memory—remain the stable core of intelligence. Unlike memorizing specific patterns, humans succeed in novel environments by applying these intrinsic faculties to adapt and optimize. Yet, whether LLMs possess this essential capacity—namely, the ability to continuously refine solutions in response to dynamic environmental feedback—remains underexplored. To address this challenge, we introduce **OPT-BENCH**, a benchmark for evaluating self-improvement capabilities in large-scale search spaces. By combining 20 machine learning tasks with 10 classic NP-hard problems, OPT-BENCH provides a rigorous setting to assess whether agents can adapt through intrinsic self-reflection rather than rote tool application. We further propose **OPT-Agent**, a framework that emulates human-like cognitive adaptation. It operates via a general perception–memory–reasoning loop, iteratively refining solutions based on environmental feedback. Through extensive experiments on 19 LLMs from 7 model families, including reasoning models, general models, and open-source models ranging from 3B to 235B parameters, we demonstrate stronger models are more effective at leveraging feedback signals for self-improvement. However, this upper-bound adaptability remains fundamentally constrained by the models' base capacity, and even the most advanced LLMs still fall short of human expert performance.

1 Introduction

The advent of Large Language Models (LLMs) (OpenAI, 2025; Grattafiori et al., 2024; Guo et al., 2025) has revolutionized artificial intelligence, demonstrating exceptional performance across a wide range of tasks (Brown et al., 2020; Ouyang et al., 2022; Achiam et al., 2023; Chowdhery et al., 2023; Touvron et al., 2023; Google, 2024). While specific interaction tools evolve rapidly, the fundamental cognitive faculties required for problem-solving—*perception*, *reasoning*, and *memory*—remain the stable core of intelligence. Humans succeed in novel environments not by memorizing specific interfaces, but by applying these intrinsic principles to adapt and optimize. However, determining whether LLMs possess this essential capacity for

iterative self-optimization—continuously refining solutions in response to **dynamic environmental feedback**—remains an underexplored frontier.

Current benchmarks primarily measure a model's ability to generate correct responses in a single pass (Fan et al., 2023; Lin et al., 2024), overlooking the essential human capability of learning from experience. Humans routinely refine reasoning by integrating feedback—scientists adjust hypotheses, students revise strategies, and chess players improve tactics based on past outcomes. Yet, existing evaluations lack a unified framework to assess this *trajectory* of self-improvement, leaving a critical gap in understanding how agents navigate complex search spaces through intrinsic reflection rather than rote memorization.

To address this limitation, we introduce **OPT-BENCH**, a comprehensive benchmark designed to probe the limits of LLM agents in large-scale search space self-optimization driven by environmental signals. OPT-BENCH juxtaposes two distinct feedback landscapes: (1) **20 Real-world Machine Learning (ML) tasks** (sourced from Kaggle), representing *continuous optimization spaces* where feedback provides noisy but directional gradients (e.g., improving validation accuracy); and (2) **10 Classical NP-hard problems**, representing *discrete combinatorial spaces* characterized by brittle constraints and local optima. This deliberate combination challenges agents to demonstrate versatile cognitive adaptability: employing **inductive reasoning** to tune hyperparameters in ML tasks, while switching to **deductive logic** to construct valid structures in NP problems. To evaluate performance relative to human cognition, we curate gold-standard solutions from Kaggle leaderboards for ML tasks and implement **Human-expert Heuristics** for NP tasks, establishing a baseline rooted in expert reasoning patterns rather than brute-force computation.

To facilitate rigorous evaluation, we present **OPT-Agent**, a framework designed to emulate **human-like cognitive adaptation**. Unlike complex agent architectures that rely heavily on engineered prompts or external solvers, OPT-Agent employs a **perception–memory–reasoning loop**. It generates solutions, validates them against the environment, and iteratively refines them by retrieving and analyzing *accumulated environmental signals*.

Using this benchmark, we conduct extensive experiments on 19 LLMs from 7 model families. Our analysis uncovers a "Scaling Law of Self-Improvement": stronger base models are significantly more effective at leveraging historical feedback to optimize solutions. However, we reveal a critical **cognitive divergence**: while historical context effectively drives optimization in continuous ML domains, its benefit is often constrained in discrete NP tasks. Even frontier models (e.g., GPT-4o) struggle with *incremental refinement* in combinatorial spaces, frequently resetting solutions rather than repairing them—highlighting a persistent gap compared to human expert performance.

In summary, our contributions are:

- We propose **OPT-BENCH**, a benchmark that juxtaposes continuous (ML) and discrete (NP) optimization tasks to evaluate the *intrinsic self-optimization* capabilities of LLMs, moving beyond rote tool application to assess dynamic adaptability.
- We introduce **OPT-Agent**, an evaluation framework that emulates the human perception-memory-reasoning loop, enabling a rigorous assessment of how models leverage historical environmental feedback to refine solutions iteratively.
- We provide a comprehensive analysis of 19 LLMs, revealing that while strong models excel at inductive optimization in continuous spaces, they face fundamental reasoning barriers in discrete combinatorial refinement, marking a key direction for future AGI research.

2 OPT-BENCH

To rigorously evaluate adaptive intelligence across diverse feedback landscapes, OPT-BENCH integrates 30 distinct tasks: 20 machine learning (ML) challenges and 10 classical NP-hard problems. This composition is intentionally designed to juxtaposition two fundamental modes of reasoning: (1) **Continuous Inductive Optimization** (ML tasks): utilizing noisy, directional feedback (e.g., accuracy metrics) to tune continuous hyperparameters in applications like sales forecasting and sentiment analysis. (2) **Discrete Deductive Reasoning** (NP tasks): navigating brittle, combinatorial search spaces with strict constraints, covering graph theory and resource allocation problems such as Hamiltonian cycle and TSP. These problems are selected for their computational intractability at scale, forcing agents to rely on heuristic planning rather than brute-force memorization. See Appendix A for the complete task list.

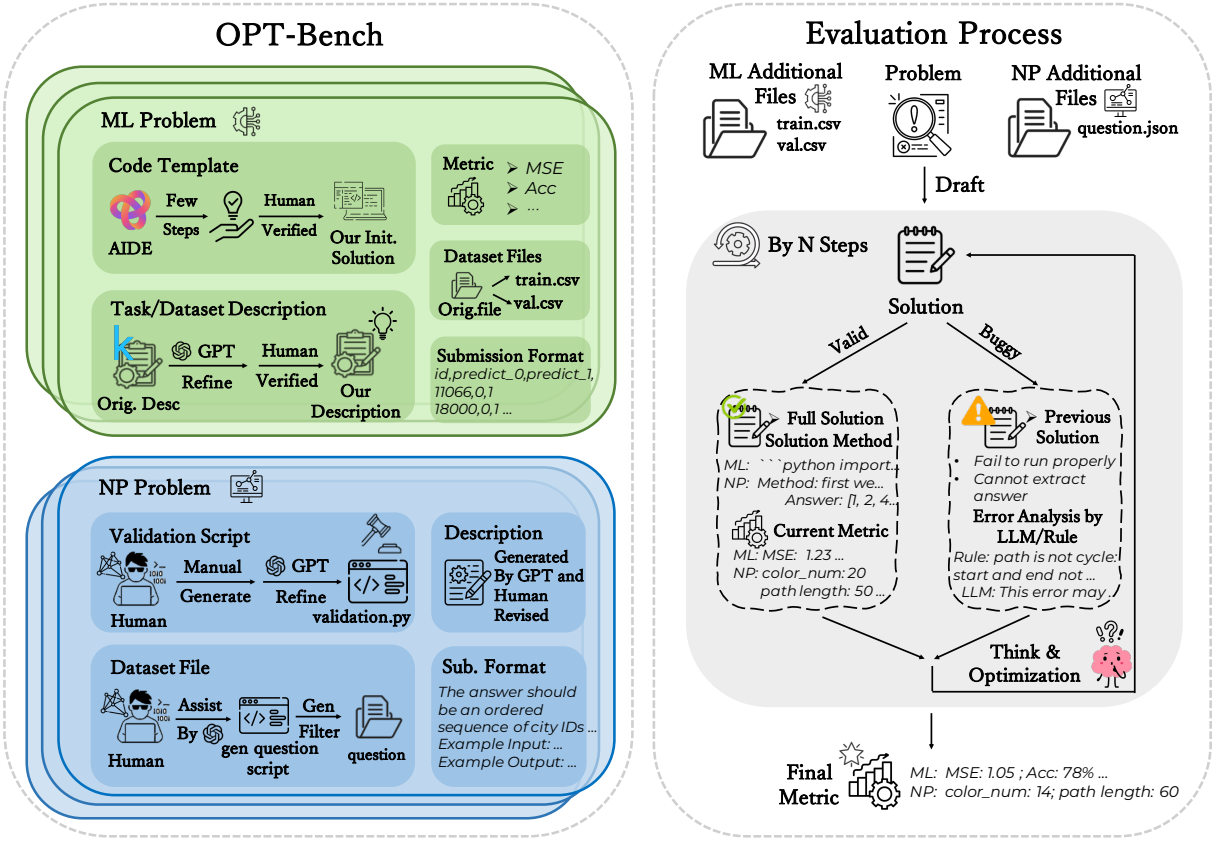

Figure 2: **Overview of the OPT-BENCH dataset and OPT-Agent Framework.** The left panel illustrates the data structure of OPT-Bench, encompassing ML and NP problems. Each module includes problem definitions, dataset files, validation script (NP), evaluation metrics, and submission formats, integrating human-verified initial solutions and LLM-assisted refinement. The right panel details the evaluation workflow, where solutions are iteratively generated and validated in steps. Valid solutions proceed to metric calculation, while buggy solutions trigger error analysis via LLM or rule-based diagnostics, followed by iterative optimization. This framework enables systematic and automated assessment of LLMs' optimization capabilities across diverse problem domains.

2.1 Dataset Curation and Analysis

As illustrated in Figure 2, the preparation of ML tasks ensures a rigorous testbed for real-world engineering capabilities. We collect task descriptions from Kaggle, refine them via GPT-4o, and verify them through human experts to ensure ambiguity-free instructions. Evaluation metrics are strictly defined to provide objective feedback signals. To simulate a realistic optimization starting point, we generate an initial solution using the AIDE agent (Jiang et al., 2025), followed by refinement from four PhD-level experts. This ensures the initial state is functional but suboptimal, providing ample headroom for iterative improvement. We utilize Kaggle leaderboard gold medal solutions as the **Human Expert Baseline**, representing the upper bound of domain-specific engineering.

For NP tasks, the benchmark focuses on 10 classical problems encapsulated in JSON format. Unlike ML tasks where the solution is a script, NP tasks require the construction of valid discrete structures (e.g., a specific node sequence). We provide detailed goal definitions and example I/O to guide the model's deductive process. Validity is enforced by a rule-based script `validation.py`, which acts as the "Environment," providing binary feedback (Valid/Invalid) and specific error messages (e.g., "Node visited twice"). Crucially, to establish a meaningful comparison for intrinsic reasoning, we implement **Human-Exper Heuristics** (e.g., simulated annealing algorithm) as the baseline. This serves as a proxy for expert human reasoning patterns, distinct from the theoretical optima found by industrial solvers.

Each sample in OPT-BENCH comprises the following elements (see Figure 3):

- **Task Description**: Defines the optimization landscape, including background context and specific objectives (e.g., minimize path length).
- **Dataset Specifications**: ML tasks provide CSVs with feature descriptions; NP tasks provide JSON

instances of varying topological complexity.

- **Submission Formats**: Stipulates the required output structure, ensuring consistent parsing for automated evaluation.
- **Initial Solution (Cold Start)**: A functional but suboptimal baseline (script for ML, valid path for NP) provided to the agent to kickstart the optimization trajectory.
- **Environmental Feedback Mechanism**: For ML, this involves execution on a hold-out test set; for NP, `validation.py` verifies constraints and computes metrics. This mechanism simulates dynamic environmental signals.
- **Human/Expert Baseline**: Kaggle Gold Solutions for ML, and Human-Proxy Heuristics for NP, providing a "reasoning ceiling" for comparison.

2.2 OPT-Agent Workflow

To better evaluate the intrinsic self-optimization capabilities of LLMs, we introduce **OPT-Agent**, a framework inspired by AlphaEvolve (Novikov et al., 2025) and cognitive Chain-of-Thought theories. Instead of relying on elaborate prompt engineering, OPT-Agent implements a fundamental **Perception–Memory–Reasoning Loop**. As illustrated in Figure 2, it iteratively refines solutions by accumulating and analyzing environmental signals.

- **Drafting (Initialization)**: The agent enters the environment by generating an initial hypothesis. For ML tasks, it synthesizes a Python training script; for NP tasks, it uses **deductive reasoning** to directly construct a solution structure (e.g., a path sequence). This direct construction forces the model to rely on internal planning rather than offloading logic to a code interpreter.
- **Improving (Refinement)**: Triggered when a valid solution is found. The LLM acts as an optimizer, retrieving *historical context*—including past code/paths, performance metrics, and feedback trends. For ML, it performs **inductive reasoning** to adjust architecture or hyperparameters. For NP, it attempts to refine the discrete structure (e.g., shortening a TSP path) while maintaining validity constraints. This step tests the agent's ability to learn from the trajectory of past successes.
- **Debugging (Correction)**: Invoked when the environment returns a failure signal (e.g., runtime error or constraint violation). The agent analyzes the error logs provided by the environment to diagnose the fault. This step evaluates the agent's **self-correction** capability and its ability to recover from invalid states in the search space.

3 Experiments and Results

3.1 Experimental Setup

Environments. OPT-BENCH-ML experiments utilized a standard CPU environment (4 cores, 32GB RAM), while OPT-BENCH-NP required minimal resources (2 cores, 16GB RAM). We accessed proprietary models via API and open-source models (3B–72B) via LMDeploy.

Baselines and Metrics. To rigorous quantify the **Self-Optimization** capability—defined as the ability to improve over time using environmental feedback—we establish the following evaluation protocols:

3.1.1 Evaluation Metrics and Baselines

To rigorously quantify the **Self-Optimization** capability—defined as the agent's ability to evolve from a crude initial state to an expert-level solution using environmental feedback—we establish the following evaluation protocols:

- **Baselines (The Optimization Bounds):**
  1. *Lower Bound (Random Rollout):* The performance of the agent generating solutions independently without historical context (memoryless). This represents "blind" trials and serves as the baseline for assessing whether the agent is truly learning or merely guessing.
  2. *Upper Bound (Human Expert):* The Gold Medal solution (for ML) or heuristic optimal (for NP), representing the functional ceiling of task performance.
- **Win Count (Feedback Utility):** Defined as the number of tasks in which the OPT-Agent outperforms the *Random Rollout* baseline. This metric quantifies the model's ability to perform self-optimization. A high Win Count indicates that the model is successfully interpreting environmental feedback and using it to guide performance improvement.
- **Improvement Rate (IR):** A quantitative measure of **learning efficiency**, primarily used to

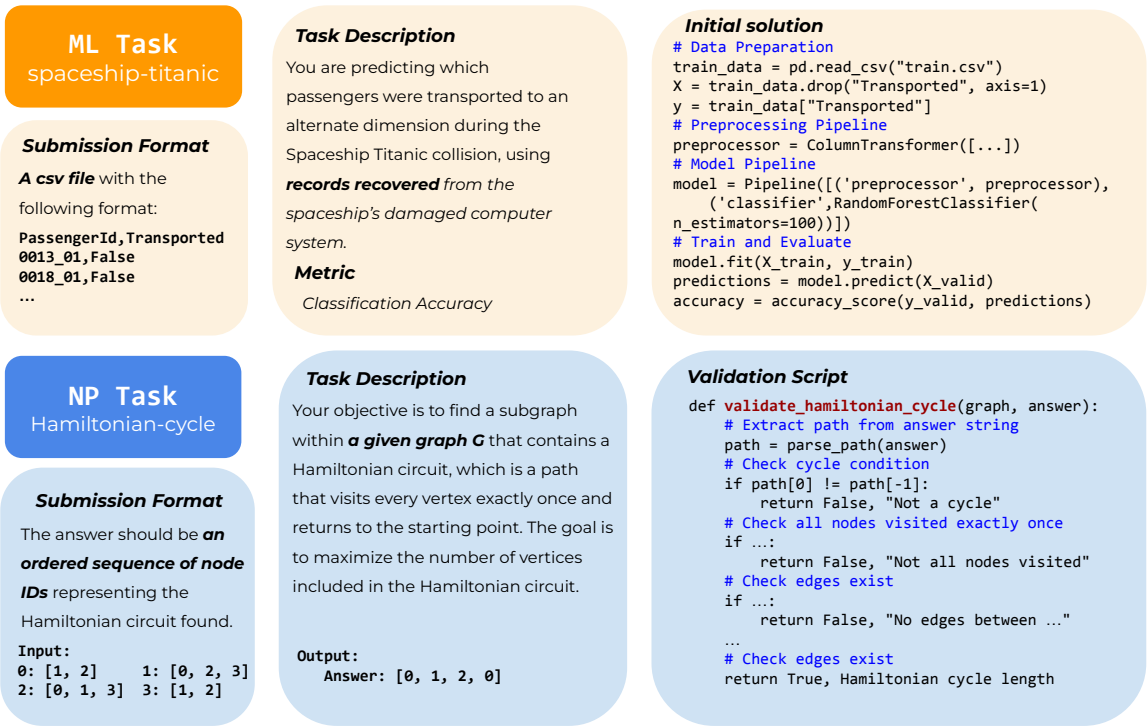

Figure 3: **Specific cases from OPT-BENCH**. Take the spaceship titanic classification task and the Hamiltonian cycle optimization problem as representative examples.

capture relative performance gains in ML tasks when compared with the Random Rollout baseline. It is defined as:

$$\mathrm{IR}(\alpha, \beta) = \frac{1}{n} \sum_{i=1}^{n} \frac{\alpha_i}{\beta_i},$$

where α_i denotes the metric value after optimization, and β_i denotes the corresponding baseline value (either the initial solution or a memory-less generation). This metric reflects the *magnitude of progress* achieved along the optimization trajectory; values of $\mathrm{IR} > 1.0$ indicate that the LLM is effectively leveraging environmental feedback to improve performance.

- **Expert Gap (EG):** To address the scale heterogeneity across 30 diverse tasks (e.g., MSE vs. Accuracy), we compute the *Normalized Gap* relative to the human expert. For a task with metric M, the score is normalized as:

$$\mathrm{Score}_{norm} = \frac{M_{current} - M_{initial}}{M_{expert} - M_{initial}}$$

This serves as the definitive measure of **solution quality**. It quantifies the extent to which the agent's self-optimization loop bridges the gap between the initial 'Draft' and the 'Expert' solution, enabling fair comparisons across diverse domains.

- **Buggy Rate:** The proportion of invalid solution attempts (e.g., syntax errors or constraint violations). A decreasing buggy rate over iterations indicates that the agent is not only optimizing for task performance but also progressively learning to satisfy the environment's structural constraints—particularly in NP tasks.

3.2 Main Results: Dynamics of Self-Optimization

Continuous Parametric Optimization (ML). Table 1 validates our hypothesis: in continuous search spaces, strong LLMs effectively function as *inductive optimizers*, interpreting numerical feedback as directional gradients to guide self-optimization. Proprietary models (e.g., `gpt-4o`) achieve a dominant 18/2 Win Count over the random baseline at 20 steps. This confirms that performance gains stem from purposeful, **history-driven refinement**—effectively pruning the search space based on accumulated environmental signals—rather than from stochastic fluctuations.

Scaling Law of Model Size. We observe a sharp divergence in performance based on model scale (e.g., Expert Gap 0.45 for `Qwen2.5-72B` vs. 0.20 for `7B`). This suggests a potential **cognitive threshold**: smaller models tend to perceive complex error traces as noise due to limited semantic working memory, whereas larger models exhibit emergent

Model	5 steps			10 steps			20 steps		
	Win Count	IR(w,w.o)	EG	Win Count	IR(w,w.o)	EG	Win Count	IR(w,w.o)	EG
Proprietary LLMs									
gpt-4o-2024-08-06	13/7	1.28	0.53	13/7	1.80	0.58	**18/2**	1.89	0.61
gpt-4.1-2025-04-14	12/8	1.11	0.40	14/6	1.67	0.50	14/6	**2.15**	0.58
gpt-o3-mini	12/8	1.25	**0.55**	14/6	1.77	**0.63**	13/7	1.35	**0.65**
gemini-2.0-flash	13/7	1.08	0.45	14/6	**2.08**	0.48	14/6	1.95	0.53
claude-3-5-sonnet-20241022	11/9	1.15	0.30	12/8	1.45	0.40	12/8	1.83	0.48
claude-3-7-sonnet-20250219	14/6	1.00	0.43	14/6	1.11	0.48	14/6	1.35	0.53
grok-3	12/8	1.16	0.50	14/6	1.36	0.55	15/5	1.29	0.63
Deepseek-V3.1-Thinking	12/8	1.13	0.50	12/8	1.18	0.58	11/9	1.03	0.60
Qwen3-235B-Thinking	12/8	1.18	0.52	13/7	1.13	0.55	14/6	1.13	0.62
Qwen3-235B-Instruct	12/8	1.13	0.48	13/7	1.18	0.54	13/7	1.03	0.60
Open-Source LLMs									
Internlm3-8b-instruct	8/12	0.94	0.05	9/11	0.98	0.23	10/10	1.01	0.30
Qwen2.5-3B-Instruct	8/12	0.98	0.05	7/13	0.85	0.13	6/14	0.63	0.15
Qwen2.5-7B-Instruct	9/11	0.99	0.03	7/13	0.82	0.15	6/14	0.63	0.20
Qwen2.5-14B-Instruct	10/10	1.39	0.18	11/9	1.11	0.25	12/8	1.13	0.30
Qwen2.5-32B-Instruct	9/11	1.38	0.25	10/10	1.21	0.33	11/9	1.19	0.45
Qwen2.5-72B-Instruct	**15/5**	**1.58**	0.40	**15/5**	1.47	0.43	15/5	1.61	0.45
Qwen3-8B	9/11	0.99	0.28	10/10	1.24	0.33	12/8	1.16	0.38
Qwen3-30B-A3B	10/10	1.06	0.45	11/9	1.08	0.53	12/8	1.14	0.54
Qwen3-32B	10/10	1.07	0.18	11/9	1.19	0.48	12/8	1.28	0.58

Table 1: Evaluation Results of LLMs on OPT-BENCH-ML, comparing both closed-source and open-source models, including general and reasoning models.

capabilities for multi-step inductive reasoning.

Reasoning vs. General Models. Reasoning models (e.g., `gpt-o3-mini`) consistently outperform general instruction-tuned counterparts, achieving the highest Expert Gap closure (0.65). This indicates that the *Chain-of-Thought (CoT)* paradigm is critical for *fine-grained optimization*, enabling agents to deduce the causal mechanism of failure rather than resorting to random parameter guessing.

Optimization Horizon. Extending the optimization trajectory to 20 steps yields substantial gains for proprietary models (e.g., `gpt-4.1` achieves an IR of 2.15), while weaker models plateau early. This highlights that **long-horizon self-optimization** requires strong contextual retention: without it, agents lose focus and regress to near-random exploration.

Discrete Combinatorial Domains (NP). Table 2 presents a contrasting landscape: in combinatorial search spaces, environmental feedback yields diminishing returns for self-optimization.

The Feedback Efficiency Paradox. Unlike the dominance observed in ML tasks, Win Counts here are more balanced (e.g., `gpt-4o`: 4/6), revealing a **feedback misinterpretation** phenomenon. Discrete error signals (e.g., "Invalid Cycle") lack directional gradients, making it difficult for standard models to translate them into structural repairs. As a result, self-optimization often degenerates into "random search with memory."

Reasoning vs General Models. Reasoning models (e.g., `Deepseek-V3.1-Thinking`) achieve a 0.00 buggy rate and the highest Expert Gap ($\sim$0.79), confirming that Chain-of-Thought (CoT) reasoning is *essential* for maintaining global topological consistency. These models can verify solution validity in ways that instruction-tuned models often fail to do.

The Validity Bottleneck. Smaller open-source models encounter a **feasibility threshold** (buggy rates $\sim$0.80). Unlike soft failure modes in ML domains, NP tasks impose a hard "valid/invalid" constraint. Failure to cross this threshold renders historical feedback ineffective, as the agent never establishes a valid baseline from which to optimize.

Optimization Horizon and the Deductive Ceiling. Larger general models (e.g., `Qwen2.5-72B`) exhibit moderate gains (EG $0.33 \rightarrow 0.47$) over 20 steps, while smaller models plateau early due to insufficient planning depth for complex topological transformations. Interestingly, reasoning models also show limited improvement despite strong initial performance, possibly due to **reinforcement learning alignment constraints** that inhibit long-horizon exploration.

Model	5 steps					10 steps					20 steps				
	Win Count	Buggy Rate		EG		Win Count	Buggy Rate		EG		Win Count	Buggy Rate		EG	
		w	w.o	w	w.o		w	w.o	w	w.o		w	w.o	w	w.o
Proprietary LLMs															
gpt-4o-2024-08-06	5/5	0.30	0.34	0.41	0.40	4/6	0.26	0.30	0.44	0.44	4/6	0.22	0.18	0.47	0.54
gpt-4.1-2025-04-14	4/6	0.10	0.10	0.45	0.47	4/6	0.08	0.08	0.47	0.48	4/6	0.04	0.02	0.53	0.55
gpt-o3-mini	5/5	**0.00**	**0.00**	0.72	0.68	6/4	**0.00**	**0.00**	0.72	0.69	5/5	**0.00**	0.00	0.73	0.69
gemini-2.0-flash	5/5	0.16	0.20	0.40	0.42	6/4	0.06	0.12	0.44	0.43	**7/3**	0.06	0.10	0.45	0.40
claude-3-5-sonnet-20241022	7/3	0.20	0.32	0.49	0.44	7/3	0.18	0.30	0.51	0.46	**7/3**	0.18	0.28	0.52	0.47
claude-3-7-sonnet-20250219	**8/2**	0.10	0.24	0.62	0.52	**8/2**	0.08	0.22	0.64	0.55	**7/3**	0.06	0.12	0.66	0.58
grok-3	5/5	0.16	0.14	0.52	0.55	5/5	0.14	0.12	0.57	0.56	6/4	0.12	0.11	0.62	0.58
Deepseek-V3.1-Thinking	6/4	**0.00**	**0.00**	**0.76**	0.74	5/5	**0.00**	**0.00**	**0.78**	0.75	5/5	0.00	0.00	**0.79**	**0.76**
Qwen3-235B-Thinking	6/4	**0.00**	**0.00**	0.74	**0.76**	5/5	**0.00**	**0.00**	0.75	**0.76**	5/5	0.00	0.00	**0.76**	**0.76**
Qwen3-235B-Instruct	6/4	0.16	0.15	0.48	0.50	6/4	0.10	0.10	0.53	0.52	6/4	0.06	0.10	0.57	0.55
Open-Source LLMs															
Internlm3-8b-Instruct	4/6	0.52	0.54	0.12	0.14	6/4	0.30	0.42	0.20	0.18	6/4	0.18	0.20	0.24	0.20
Qwen2.5-3B-Instruct	6/4	0.64	0.80	0.12	0.03	7/3	0.46	0.68	0.18	0.09	8/2	0.38	0.58	0.21	0.14
Qwen2.5-7B-Instruct	6/4	0.54	0.70	0.22	0.14	7/3	0.38	0.54	0.31	0.23	6/4	0.24	0.28	0.35	0.33
Qwen2.5-14B-Instruct	5/5	0.40	0.60	0.31	0.22	5/5	0.24	0.28	0.41	0.39	6/4	0.18	0.18	0.47	0.43
Qwen2.5-32B-Instruct	5/5	0.42	0.44	0.33	0.35	5/5	0.30	0.28	0.41	0.43	6/4	0.20	0.20	0.52	0.46
Qwen2.5-72B-Instruct	5/5	0.40	0.50	0.33	0.30	5/5	0.32	0.38	0.39	0.38	4/6	0.22	0.32	0.47	0.44
Qwen3-8B	5/5	0.28	0.22	0.51	0.56	5/5	0.20	0.20	0.57	0.58	5/5	0.12	0.20	0.63	0.60
Qwen3-30B-A3B	5/5	0.04	0.12	0.60	0.59	5/5	**0.00**	0.06	0.65	0.65	6/4	**0.00**	**0.00**	0.69	0.68
Qwen3-32B	5/5	0.14	0.12	0.52	0.57	4/6	0.10	0.10	0.57	0.59	5/5	0.06	0.10	0.60	0.60

Table 2: Evaluation Results of LLMs on OPT-BENCH-ML, comparing both closed-source and open-source models, including general and reasoning models.

3.3 Ablation Study: Stability vs. Exploration

To decouple the effects of sampling randomness from intrinsic reasoning, we investigate how Temperature (T) modulates the **Self-Optimization** trajectory (Tables 3 and 4).

- **Continuous Domains (ML):** For strong models like `gpt-4o`, performance peaks at $T = 0$ (13/7 Win Count) and degrades significantly at $T = 0.8$ (10/10). *Insight:* In continuous spaces, environmental feedback acts as a specific directional gradient (e.g., "decrease learning rate"). Setting $T = 0$ ensures the agent strictly adheres to this signal, maximizing **Exploitation**. Higher temperatures introduce "adversarial noise," disrupting the precise numerical fine-tuning required for convergence and effectively diluting the value of historical context.

- **Discrete Domains (NP):** The dynamics shift in combinatorial spaces. While $T = 0$ ensures consistency, a moderate temperature ($T = 0.2$) often yields lower Buggy Rates for models like `gpt-4o` (0.18 vs. 0.28 at $T = 0$). *Insight:* Unlike the smooth landscape of ML, NP problems are characterized by "rugged" local optima. Deterministic decoding can cause the agent to fixate on a specific invalid structural pattern. Slight stochasticity ($T = 0.2$) provides the necessary **Exploration** to escape these local traps. However, excessive randomness ($T = 0.8$) shatters the logical coherence required to maintain constraints, causing Win Counts to drop (e.g., `gpt-4o` falls to 4/6) as the optimization degenerates into random guessing.

3.4 Discussion: The Limits of Intrinsic Self Optimization

Our findings highlight a critical boundary in LLM self-optimization cognition. **Signal Interpretation Divergence:** Self-optimization hinges on accurately mapping *Environmental Feedback* to *Action Updates*. We observe that LLMs excel in ML domains where feedback is semantic or numerical, facilitating coherent, incremental refinement. Conversely, they falter in NP tasks due to the challenge of maintaining complex structural dependencies. For instance, in the Hamiltonian cycle problem, while a human would locally repair a disconnected edge, LLMs often discard the entire history to generate a completely new solution, as shown in Figure 5. This inability to perform *incremental structural editing* explains why historical feedback yields diminishing returns in discrete combinatorial spaces compared to continuous ML landscapes.

The Human Gap: Even with self-optimization, a significant gap persists compared to the Hu-

Model	Temperature=0			Temperature=0.2			Temperature=0.8		
	Win Count	IR(w,w.o)	EG	Win Count	IR(w,w.o)	EG	Win Count	IR(w,w.o)	EG
gpt-4o-2024-08-06	**13/7**	**1.58**	0.58	9/11	1.19	0.50	10/10	1.10	0.40
grok-3	9/11	1.01	0.65	**11/9**	**1.29**	0.45	8/12	1.04	0.43
Qwen2.5-72B-Instruct	10/10	1.05	0.43	11/9	1.04	0.40	**11/9**	**1.11**	0.44

Table 3: Evaluation Results of LLMs on OPT-BENCH-ML Across Different Temperature Settings.

Model	Temperature=0					Temperature=0.2					Temperature=0.8				
	Win Count	Buggy Rate		EG		Win Count	Buggy Rate		EG		Win Count	Buggy Rate		EG	
		w	w.o	w	w.o		w	w.o	w	w.o		w	w.o	w	w.o
gpt-4o-2024-08-06	**5/5**	0.28	0.24	0.48	0.44	**5/5**	0.18	0.28	0.51	0.47	4/6	0.18	0.22	0.47	0.49
grok-3	**5/5**	**0.14**	**0.18**	**0.54**	**0.52**	**5/5**	**0.12**	**0.14**	**0.56**	**0.53**	4/6	**0.00**	**0.00**	**0.60**	**0.61**
Qwen2.5-72B-Instruct	**5/5**	0.30	0.34	0.47	0.46	4/6	0.24	0.26	0.44	0.46	**5/5**	0.25	0.26	0.46	0.44

Table 4: Evaluation Results of LLMs on OPT-BENCH-NP across Different Temperature Settings.

man Expert Upper Bound. While agents can improve over their initial drafts, they rarely reach the global optima that humans find through principled heuristic derivation, suggesting that current "Self-Optimization" is essentially a *local search* mechanism bounded by the model's inherent reasoning depth.

4 Related Work

LLM Evaluation Early benchmarks like MMLU (Hendrycks et al., 2020) and BIG-bench (Srivastava et al., 2022) assessed broad knowledge but remained confined to static, multiple-choice formats. While subsequent reasoning-focused evaluations—ranging from math (MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021)) and code (HumanEval (Chen et al., 2021), MBPP (Austin et al., 2021)) to combinatorial problems (NP-Engine (Li et al., 2025))—have advanced the field, they predominantly rely on **one-shot, open-loop paradigms**. Even with Chain-of-Thought prompting (Wei et al., 2022), these benchmarks evaluate *instantaneous deduction* rather than *adaptive learning*. Notably, NPHardEval (Fan et al., 2023) introduces complexity classes but remains a single-pass solvability test, failing to capture the **iterative self-correction** dynamics essential for autonomous optimization.

LLM Agents Recent frameworks have empowered LLMs with tool use (ReAct (Yao et al., 2022a), Toolformer (Schick et al., 2023)) and feedback loops (Reflexion (Shinn et al., 2023)). However, existing agent benchmarks primarily focus on **task completion** rather than **solution refinement**. Benchmarks like AgentBench (Liu et al., 2023), WebShop (Yao et al., 2022b), and WebArena (Zhou et al., 2023) evaluate whether an agent can *execute* a sequence of actions to finish a task, not whether it can *optimize* a metric over time. MLE-Bench (OpenAI, 2024) targets machine learning engineering but emphasizes code execution success over the cognitive trajectory of hyperparameter tuning. Similarly, IOLBench (Zhang et al., 2024) focuses on linguistic reasoning. A critical gap remains for a unified benchmark that strictly evaluates **Intrinsic Self-Optimization**—the ability to leverage environmental feedback to climb performance landscapes—across both continuous (ML) and discrete (NP) domains. OPT-BENCH addresses this by shifting focus from "Can the agent run the code?" to "Can the agent evolve the solution?"

5 Conclusion

We introduced **OPT-BENCH** and **OPT-Agent** to evaluate the ability of LLMs to perform *iterative self-optimization* across contrasting search spaces. By juxtaposing 20 continuous (ML) and 10 discrete (NP) environments, our evaluation of 19 LLMs across 7 model families reveals a clear divergence: while models exhibit strong performance on **Continuous Parametric Optimization**—effectively leveraging numerical feedback for iterative refinement—they struggle with **Combinatorial Reasoning**, often failing to translate discrete error signals into structural repairs. We further find that reasoning-enhanced models consistently outperform general models, particularly in NP domains where surpassing the feasibility threshold requires structured reasoning. Such models are essential for handling combinatorial constraints. However, the persistent gap between even the strongest LLMs and **Human Expert Baselines** reveals a fundamental limitation: achieving true autonomous optimization demands more than local pattern matching—it requires global planning and high-level reasoning.

Limitations

Due to resource constraints, we were unable to include additional state-of-the-art models such as the Gemini 3 series, OpenAI GPT-5.2, and Claude 4.5 in our experiments. In addition, our current benchmark contains only 30 environments. In future work, we plan to scale the benchmark to a larger and more diverse set of environments in order to enable more comprehensive and robust evaluation.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and 1 others. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, and 1 others. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yao Fan, Xinyu Hua, and 1 others. 2023. Nphardeval: Dynamic benchmark on reasoning ability of large language models via complexity classes. *arXiv preprint arXiv:2312.14890*.

Google. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

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. *Preprint*, arXiv:2502.13138.

Xiaozhe Li, Xinyu Fang, Shengyuan Ding, Linyang Li, Haodong Duan, Qingwen Liu, and Kai Chen. 2025. Np-engine: Empowering optimization reasoning in large language models with verifiable synthetic np problems. *Preprint*, arXiv:2510.16476.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. Criticbench: Benchmarking llms for critique-correct reasoning. *Preprint*, arXiv:2402.14809.

Xiao Liu and 1 others. 2023. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*.

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.

OpenAI. 2024. Mle-bench: Evaluating machine learning agents on real-world machine learning engineering tasks. *arXiv preprint arXiv:2410.07095*.

OpenAI. 2025. Openai o1 system card. Technical report, OpenAI.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Timo Schick, Jane Dwivedi-Yu, and 1 others. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 14.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, and 1 others. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022a. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Shunyu Yao and 1 others. 2022b. Webshop: Towards scalable real-world web interaction with grounded language agents. *arXiv preprint arXiv:2207.01206*.

Yicheng Zhang and 1 others. 2024. Iolbench: Benchmarking llms on linguistic reasoning. *arXiv preprint arXiv:2501.04249*.

Shuyan Zhou and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Appendix

A.1 Use of Large Language Models

Large Language Models are used for grammar check and polishing in this paper.

A.2 Draft Setting

Unlike the refine setting, the draft setting requires models to generate solutions from scratch, providing a more direct test of their optimization capabilities. Results for three strong models are shown in Table 5. Notably, the open-source `Qwen2.5-72B-Instruct` consistently shows higher buggy rates than proprietary models, reflecting greater difficulty in producing valid solutions during draft optimization. However, except for `grok-3` at 5 steps, all models achieve higher improvement rates (*IR(d,r)*) than in the refine setting, indicating that draft optimization can outperform traditional refinement when valid solutions are found. The *Win Count* metric further shows that using historical information during draft optimization improves performance across all step counts. Increasing the number of steps yields significant gains in both improvement rates and win counts, highlighting the value of iterative refinement. These results emphasize the importance of managing the trade-off between exploration and solution validity in draft optimization and suggest that reducing buggy rates is key to advancing both proprietary and open-source LLMs in this setting.

A.3 OPT-BENCH Dataset

The detailed information regarding the 20 Machine Learning (ML) tasks and 10 NP problems used in our OPT-BENCH is comprehensively summarized in Table 6 and Table 7, respectively. These tables provide concise descriptions of each task or problem, along with their corresponding evaluation metrics, which serve as the foundation for assessing the performance of OPT-Agent across diverse optimization scenarios.

B OPT-Agent Prompt

In this section, as illustrated in Figure 4 and Figure 8, we provide a comprehensive overview of the prompt templates used in both OPT-Agent-ML and OPT-Agent-NP. These prompts guide the model through three distinct types of actions—draft, improve, and debug—by delivering task-specific context and structured response formats. Specifically, the OPT-Agent-ML prompts focus on instructing

Model	5 steps				10 steps				20 steps			
	Buggy Rate		Win Count	IR(d,r)	Buggy Rate		Win Count	IR(d,r)	Buggy Rate		Win Count	IR(d,r)
	w	w.o			w	w.o			w	w.o		
gpt-4o-2024-08-06	0.20	0.20	8/6	1.24	**0.15**	**0.15**	**10/5**	1.38	**0.15**	**0.15**	**11/4**	1.41
grok-3	**0.15**	**0.15**	**10/5**	0.80	**0.15**	**0.15**	**10/5**	1.38	0.15	0.15	**11/4**	**1.42**
Qwen2.5-72B-Instruct	0.40	0.35	5/5	**1.54**	0.30	0.30	6/6	**1.94**	0.20	0.20	8/7	1.12

Table 5: **Evaluation Results of LLMs under Draft Settings.** Metrics include *Buggy Rate*, denoting the proportion of invalid solutions; *Win Count*, comparing OPT-Agent-draft optimization against the baseline without historical information; and *IR(d,r)*, the improvement rate comparing OPT-Agent-draft optimization to OPT-Agent-refine.

Kaggle Competition	Description	Metric
bike-sharing-demand	Forecast use of a city bikeshare system	Root Mean Squared Logarithmic Error (RMSLE) ↓
competitive-data-science-predict-future-sales	Predict total sales for every product and store	Root Mean Squared Error (RMSE) ↓
house-prices-advanced-regression-techniques	Predict house sales prices	Root-Mean-Squared-Error (RMSE) ↓
london-house-price-prediction-advanced-techniques	Predict London house prices	Mean Absolute Error (MAE) ↓
playground-series-s3e14	Predicting wild blueberry yields	Mean Absolute Error (MAE) ↓
playground-series-s3e16	Predict the age of crabs	Mean Absolute Error (MAE) ↓
playground-series-s3e19	Forecast Mini-Course Sales	Symmetric Mean Absolute Percentage Error (SMAPE) ↓
playground-series-s3e22	Predict Health Outcomes of Horses	micro-averaged F1-Score ↑
playground-series-s3e24	Binary Prediction of Smoker Status using Bio-Signals	area under the ROC curve ↑
playground-series-s3e25	Regression with a Mohs Hardness Dataset	Median Absolute Error (MedAE) ↓
playground-series-s3e3	Binary Classification with a Tabular Employee Attrition Dataset	area under the ROC curve ↑
playground-series-s3e5	Ordinal Regression with a Tabular Wine Quality Dataset	quadratic weighted kappa ↑
playground-series-s4e2	Multi-Class Prediction of Obesity Risk	Accuracy ↑
sentiment-analysis-on-movie-reviews	Classify the sentiment of sentences from the Rotten Tomatoes dataset	classification accuracy ↑
spaceship-titanic	Predict which passengers are transported to an alternate dimension	classification accuracy ↑
tabular-playground-series-aug-2022	Predict product failures	area under the ROC curve ↑
tabular-playground-series-feb-2021	Predict the amount of an insurance claim	Root-Mean-Squared-Error (RMSE) ↓
tabular-playground-series-jul-2021	Predict air pollution in a city	Root Mean Squared Logarithmic Error (RMSLE) ↓
tabular-playground-series-sep-2022	Predict book sales	Symmetric Mean Absolute Percentage Error (SMAPE) ↓
telstra-recruiting-network	Predict the severity of service disruptions on their network	multi-class logarithmic loss ↓

Table 6: **Kaggle Machine Learning Competitions with Description and Metric.**

the model for machine learning tasks, while the OPT-Agent-NP prompts are carefully designed to include structured task descriptions, input-output examples, and response formatting guidelines, enabling the model to systematically address and refine solutions to NP problems.

B.1 OPT-Agent Results Analysis

B.2 ML Task

As shown in Figure 6, we present the optimization trajectory of OPT-Agent tackling bike sharing demand ML task, illustrating progressive improvements in evaluation metrics through various strategies. Beginning with hyperparameter tuning and sampling enhancements, the model undergoes iterative refinements including the introduction of early stopping, regularization techniques, and learning rate adjustments. The diagram also highlights encountered issues, such as the early stopping rounds exception, along with the corresponding fixes, demonstrating a systematic approach to model optimization and performance enhancement.

B.3 NP Problem

As shown in Figure 7, we illustrate the solution refinement process of OPT-Agent applied to the Hamiltonian Cycle NP problem. The flowchart depicts iterative attempts to build a valid Hamiltonian circuit by resolving challenges such as disconnected nodes and repeated visits. Each step includes metric evaluations, detailed state information, and proposed paths, demonstrating how OPT-Agent systematically enhances the solution toward a valid and optimized Hamiltonian cycle.

NP Problem	Description	Metric
Graph Coloring Problem (GCP)	Use the minimum number of colors necessary to achieve a valid coloring	Color Number ↓
Hamiltonian Cycle	Find the largest possible valid Hamiltonian circuit	Path Length ↑
Knapsack	Choose a subset of items to pack into a limited-capacity bag	Total Item Weight ↑
Maximum Clique Problem	Find the largest clique (a subset of vertices all connected to each other) in a given graph	Clique Size ↑
Maximum Set	Find the largest subset of a set under constraints	Set Size ↑
Meeting Schedule	Schedule meetings for as many as participants under various constraints	Total Attendees ↑
Minimum Cut	Find the minimum cut in a graph	Cut Weight ↓
Set Cover	Select a minimal number of sets from a collection such that their union covers all elements of a universal set	Subset Number ↓
Subset Sum	Find a subset of a set of numbers that adds up to a specific target value	Indice Number ↑
Traveling Salesman Problem (TSP)	Find the shortest possible route that visits each city once and returns to the starting point	Route Length ↓

Table 7: **NP Problems with Description and Metric.**

OPT-Agent-ML

Introduction (draft): You are a Kaggle grandmaster attending a competition <task type>. In order to win this competition, you need to come up with an exceptional and creative plan. To address this problem, I will provide you with the specific task description, the evaluation metrics to be used, training set and submission format in sequence.
Introduction (improve): You are a Kaggle grandmaster attending a competition <task type>. You have been provided with previously developed solution, and your task is to improve it in order to increase the performance in test dataset. Review previous solution and improve based on it. You can only modify the model, optimizer, or hyperparameters, and adjust feature engineering for compatibility.
Introduction (draft): You are a Kaggle grandmaster attending a competition <task type>. The previous solution contains a bug. According to the buggy information, revise it to fix the issue.
Task description: <task description>
Evaluation metric: <metric>
Training set format: <dataset description>
Submission format: <submission format>
History Information: <history information>
Previous Solution: <previous solution>
Previous (buggy) Implementation: <previous (buggy) implementation >
Previous (buggy) Output: <previous (buggy) output >
Instructions: <Response Format>, <Implementation Guideline>, <Solution Draft Sketch Guideline>, <Solution Improvement Sketch Guideline>, <Solution Debug Sketch Guideline>

OPT-Agent-NP

Introduction (draft): You are a great expert solving <task description> question. You should propose a solution to this question.
Introduction (improve): You are a great expert solving <task description> question. You should optimize the solution based on the history information.
Introduction (debug): You are a great expert solving <task description> question. You should debug the solution based on the previous buggy information.
Task description: <task description>
Submission Format: <submission format>
Question: The <task type> question is: <question>
History Information: <history information>
Previous Buggy Information: <Previous buggy information>
Example Input and Output: <Example Input and Output>
Instructions: <Instructions>
Response Format: <Response Format>

Figure 4: **Prompt Template of OPT-Agent.** Orange denotes draft action. Green denotes improve action. Purple denotes debug action. Blue denotes shared prompts.

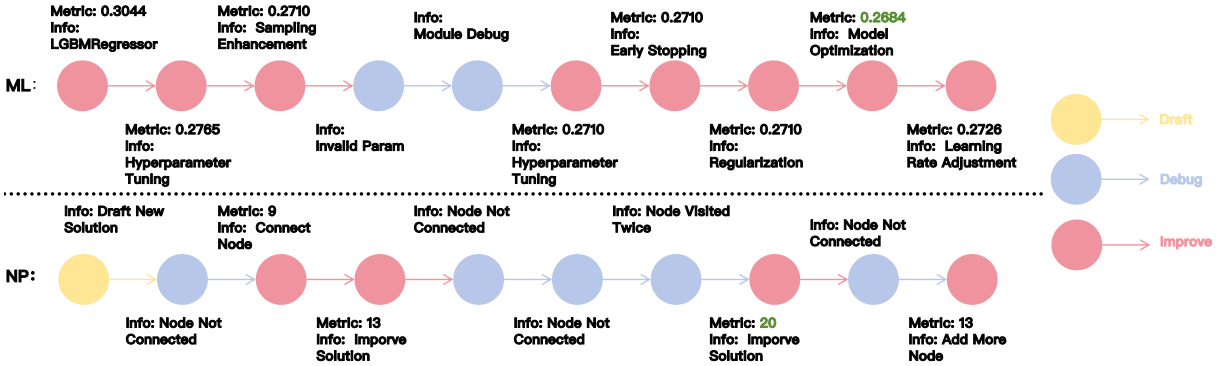

Figure 5: **Optimization Trajectories.** The figure contrasts the agent's path against the environment. In ML (Top), the agent uses error logs to monotonicially improve, demonstrating true *self-optimization*. In NP (Bottom), feedback often triggers erratic jumps, indicating a struggle to map discrete environmental signals to valid solution updates.

Figure 6: **Detailed OPT-Agent-ML Trace on the Bike Sharing Demand Task**, utilizing `gemini-2.0-flash` as LLM base model. The red, and blue nodes represent the improve, and debug action, respectively.

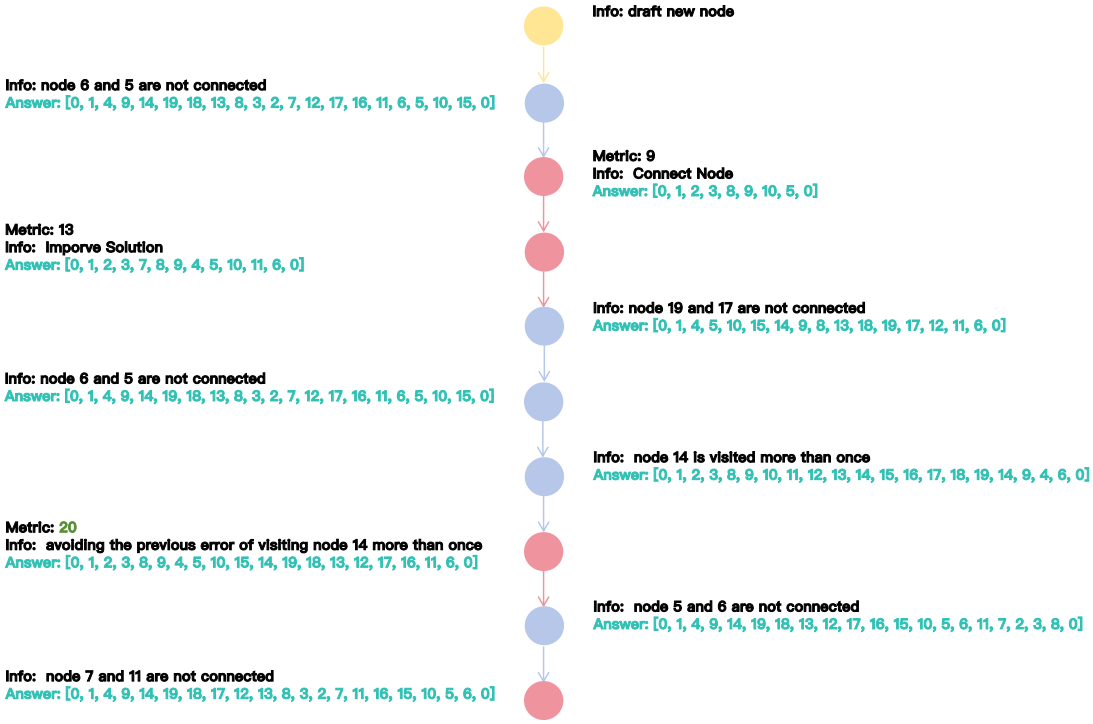

Figure 7: **Detailed OPT-Agent-NP Trace on the Hamiltonian Cycle Task**, utilizing `gemini-2.0-flash` as the LLM base model. The yellow, red, and blue nodes represent the draft, improve, and debug action, respectively.

OPT-Agent-ML

Response format: Your response should be a brief outline/sketch of your proposed solution for model, optimizer, and hyperparameters selection in natural language (3-5 sentences), followed by a single markdown code block (wrapped in "') 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 followed by a newline and then the markdown code block. Note that the code block should be a complete Python program.

Impl guideline: Be aware of the running time of the code, it should complete within `time`. All data is already available in the `./input` directory. You can also use the `./working` directory to store any temporary files that your code needs to create. The evaluation should be based on `k-fold-validation` but only if that's an appropriate evaluation for the task at hand.

Solution draft sketch guideline: The initial solution design should be simple, efficient, and avoid overfitting, with minimal iterations. Take the Memory section into consideration when proposing the design, and do not propose the same modeling solution while keeping the evaluation the same. The solution sketch should be 3-5 sentences and propose a reasonable evaluation metric for this task. Do not suggest performing Exploratory Data Analysis (EDA). The data is already prepared and available in the `./input directory`, so there is no need to unzip any files. Note that the training dataset should be shuffled before splitting into training and validation sets, and the random seed (state) should be fixed.

Solution improvement sketch guideline: The solution sketch should be a brief natural language description of how the previous solution can be improved. You should be very specific and propose only a single actionable improvement. Do not suggest performing Exploratory Data Analysis (EDA). Ensure that function parameters match the official documentation by checking for accuracy, compatibility, and any deprecated or renamed parameters, referring to the latest examples if needed. Note that only the model, optimizer, hyperparameters, and feature engineering should be modified. This improvement should be atomic so that its effect can be experimentally evaluated. Additionally, take the Memory section into consideration when proposing the improvement. The solution sketch should be 3-5 sentences.

Solution debug sketch guideline: You should write a brief natural language description (3-5 sentences) of how the issue in the previous implementation can be fixed. Do not suggest performing Exploratory Data Analysis (EDA). Ensure that function parameters match the official documentation by checking for accuracy, compatibility, and any deprecated or renamed parameters, referring to the latest examples if needed. If the previous buggy solution was due to time limitations, focus on reducing the code's time consumption rather than fixing the bug—for example, by simplifying the model's hyperparameters, reducing the number of iterations, or switching from K-Fold cross-validation to a single train-test split. Additionally, take the Memory section into consideration when proposing the improvement.

Figure 8: **Fixed prompts in OPT-Agent**. This encompasses the response format, implementation guidelines, solution draft sketch guidelines, solution improvement sketch guidelines, and solution debug sketch guidelines for ML tasks, as well as example inputs and outputs, instructions, and response format for NP problems.