

AUTOMIND: Adaptive Knowledgeable Agent for Automated Data Science

Yixin Ou^{♣♡*}, Yujie Luo^{♣♡*}, Jingsheng Zheng^{♣♡*}, Lanning Wei^{♣♡},
Zhuoyun Yu[♣], Shuofei Qiao^{♣♡}, Jintian Zhang^{♣♡}, Da Zheng^{♣♡†},
Yuren Mao[♣], Yunjun Gao[♣], Huajun Chen^{♣♡}, Ningyu Zhang^{♣♡†}

[♣]Zhejiang University [♣]Ant Group

[♡]Zhejiang University - Ant Group Joint Laboratory of Knowledge Graph
{ouyixin,zhangningyu}@zju.edu.cn zhengda.zheng@antgroup.com

 <https://github.com/innovatingAI/AutoMind>

Abstract

Large Language Model (LLM) agents have shown great potential in addressing real-world data science problems. LLM–driven data science agents promise to automate the entire machine learning pipeline, yet their real-world effectiveness remains limited. Existing frameworks depend on rigid, pre-defined workflows and inflexible coding strategies; consequently, they excel only on relatively simple, classical problems and fail to capture the empirical expertise that human practitioners bring to complex, innovative tasks. In this work, we introduce AUTOMIND, an adaptive, knowledgeable LLM-agent framework that overcomes these deficiencies through three key advances: (1) a curated expert knowledge base that grounds the agent in domain expert knowledge, (2) an agentic knowledgeable tree search algorithm that strategically explores possible solutions, and (3) a self-adaptive coding strategy that dynamically tailors code generation to task complexity. Evaluations on two automated data science benchmarks demonstrate that AUTOMIND delivers superior performance versus state-of-the-art baselines. Additional analyses confirm favorable effectiveness, efficiency, and qualitative solution quality, highlighting AUTOMIND as an efficient and robust step toward fully automated data science. Our code, data and logs for experiments are open-sourced.

1 Introduction

Data science agents aim to leverage LLM agents to automate data-centric machine learning tasks that begin with task comprehension, data exploration and analysis, advance through feature engineering, and culminate in model selection, training, and evaluation (Sun et al., 2024; Zheng et al., 2025a; Liu et al., 2025), serving as a critical component for future AI agents to achieve autonomous scientific

discovery. Many data science-related benchmarks (Huang et al., 2024a; Jing et al., 2024; Chan et al., 2025; Chen et al., 2024) have been introduced to provide structured tasks based on real-world challenges, enabling comprehensive evaluation of performance across the full problem-solving pipeline. Because of the great complexity of these tasks, most existing data science agent frameworks rely on pre-defined workflows and optimize on top of the specific workflow through search and refinement (Jiang et al., 2025; Hong et al., 2024a; Trirat et al., 2024) or extend to a multi-agent framework to better stimulate the performance of each workflow node (Li et al., 2024b).

However, current data science agents all overlook the fundamental limitations in model capabilities: despite being trained on a massive code-based corpus, the agents inherently lack the rich empirical expertise accumulated by human practitioners in data science tasks (Zheng et al., 2025a). Moreover, existing data science agents largely employ an inflexible coding strategy, and tend to implement code only for relatively simple and classic tasks in practice (Guo et al., 2024; Li et al., 2024b). Yet the diversity and complexity of real-world problems require a dynamic, context-aware coding strategy. Indeed, addressing truly complex or even cutting-edge tasks that require high levels of creativity and innovation, poses significant challenges for data science agents in generating high-quality code appropriately tailored to such complex tasks.

To tackle these issues, we propose **AUTOMIND**, an adaptive knowledgeable LLM agent framework designed for automated data science challenges. As illustrated in Figure 1, AUTOMIND introduces three major innovations:

- **Expert Knowledge Base.** An expert-curated repository of data-science knowledge that grounds the agent in empirical best practices, overcoming LLMs’ inherent lack of human

* Equal Contributions.

† Corresponding Author.

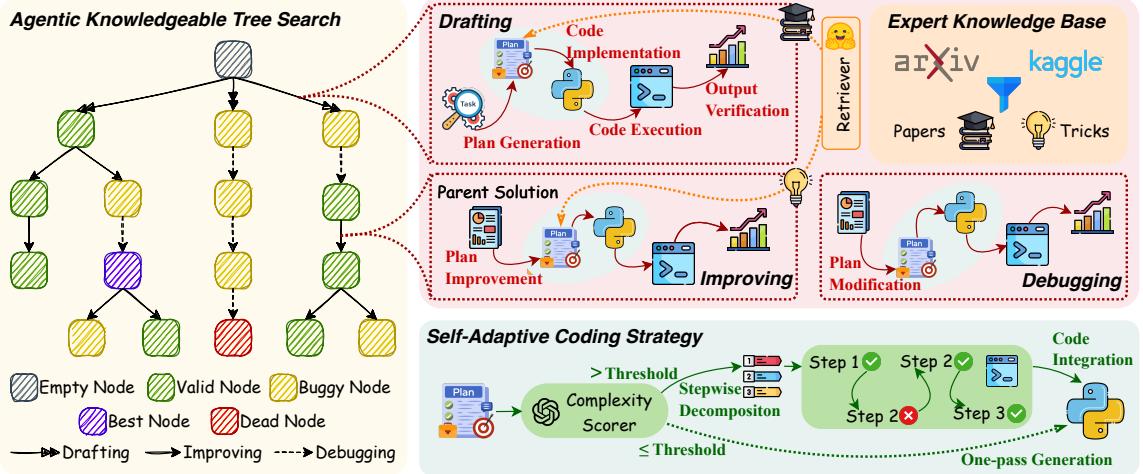


Figure 1: The framework of our AUTOMIND.

practitioner experience.

- **Agentic Knowledgeable Tree Search Algorithm.** A tree-search strategy that fully leverages the expert knowledge base, enabling the agent to dynamically explore multiple solution pathways and thereby enhance its performance on complex problem-solving tasks.

- **Self-Adaptive Coding Strategy.** A dynamic code-generation mechanism that scales with task complexity, replacing rigid workflows with context-aware implementations and thereby providing flexible, efficient solutions across varying levels of difficulty.

We evaluate AUTOMIND on two automated data science benchmarks with two different families of foundation models. Experimental results show that AUTOMIND achieves superior performance on both two of the benchmarks compared to baselines. Specifically, on the official MLE-Bench leaderboard, AUTOMIND surpasses **39.9%** of human participants on average, representing an improvement of **8.0%** over the prior state-of-the-art (SOTA). Moreover, we conduct an in-depth analysis to evaluate the effectiveness and efficiency of AUTOMIND, and find that AUTOMIND achieves higher efficiency and lower token costs compared to prior SOTA.

2 Preliminaries

Building on recent successes in integrating tree search strategies with workflows of LLM agents (Jiang et al., 2025; Chi et al., 2024; Yamada et al., 2025), we model LLM agent-driven automated data science as an optimization problem, and apply a

tree search algorithm to solve it.

Formally, a possible solution s for a data science task is defined as a tuple $s = (p, \sigma, \eta)$, where p denotes a textual plan of the proposed approach, σ is the Python code snippet, and η is the validation metric used to assess the execution results. Let \mathcal{S} be the space of all possible solutions, and the objective is to find the optimal solution:

$$s^* = \arg \max_{s \in \mathcal{S}} \eta, \quad (1)$$

where $\eta^* = \max\{\eta \mid (p, \sigma, \eta) \in \mathcal{S}\}$.

Unlike general agents (Yao et al., 2023; Hong et al., 2024b; Shinn et al., 2023; Chen et al., 2023), which conceptualize task solving as a long-horizon decision process aimed at maximizing cumulative reward through action choices based on prior observations, our modeling approach significantly simplifies the objective by directly evaluating and comparing possible solutions for data science tasks.

3 AUTOMIND

In this section, we introduce our AUTOMIND, an adaptive knowledgeable LLM agent framework designed for automated data science challenges. As illustrated in Figure 1, AUTOMIND introduces three major innovations: an expert knowledge base (§3.1), an agentic knowledge tree search module (§3.2), and a self-adaptive coding strategy (§3.3). First, the agent’s specialized knowledge retriever extracts multiple relevant papers and tricks from the expert knowledge base. Next, the agentic knowledgeable tree search module initiates an iterative loop in which it selects a parent node according to the search policy, executes an action that

synthesizes task information with retrieved knowledge to produce a new solution, and integrates the resulting node into the solution tree. Concurrently, the self-adaptive coding strategy is invoked during the code implementation stage of each action, reconciling solution complexity with the inherent coding capabilities of LLMs. Once the iteration limit is reached or the time budget is exhausted, the best node in the solution tree identified by Equation (1) is selected, submitted, and evaluated as the final solution. The following sections delve into key implementation details of each component.

3.1 Expert Knowledge for Data Science

The data science competitions are challenging due to the requirements of high-quality experience in designing effective solutions (Chan et al., 2025; Trirat et al., 2024). Using LLMs alone to solve these competitions is challenging due to their reliance on static, pre-trained knowledge, which may lack domain-specific or up-to-date insights. To address this challenge, we construct a knowledge base based on domain-specific resources, including papers from top-tier conferences and journals, as well as expert-curated insights from top-ranked competition solutions.

3.1.1 Knowledge Base Construction

In data science tasks, minor yet effective tricks can significantly enhance model performance. To incorporate such human insights in our framework, we identify all Kaggle competitions with publicly shared solutions¹, and then archive both competition descriptions and the content of associated technical forum posts. After filtering out invalid competitions and posts, we retain 3,237 public forum posts that offer valid solutions for 455 Kaggle competitions.

Besides, the papers accepted after peer-review are high-quality prior knowledge in solving different data science tasks. To utilize such knowledge, we first collect papers published in top-tier conferences in the recent three years from arXiv, like KDD, ICLR, NeurIPS, ICML, EMNLP, and domain-specific journal like Bioinformatics. For each paper, the meta information (including title, author, abstract, and keywords) and main content are preserved, from which we obtain the prepared paper knowledge.

¹We collect all Kaggle competitions with solutions using the list from <https://github.com/faridrashidi/kaggle-solutions>.

3.1.2 Knowledge Retrieval

Directly retrieving relevant knowledge using only task descriptions is challenging due to the weak correlation between real-world task descriptions and the available technical approaches. Consequently, traditional retrieval methods relying solely on task description embeddings prove ineffective in our context. To address this limitation, we propose a hierarchical labeling system to facilitate knowledge retrieval, filtering, and re-ranking.

For the collection of tricks, we first construct a hierarchy label set based on all collected data science tasks from Kaggle with the assistance of LLMs, and it contains 11 top-level categories and corresponding subcategories (e.g., category Computer Vision and subcategory Image Classification). Then, to label each trick, AUTOMIND first selects the most relevant top-level categories and then identifies the most appropriate labels from the corresponding subcategories. Compared with tricks, papers are much more diverse in data and techniques, bringing difficulties in designing hierarchical labels. Then, we use LLMs to generate a brief summary for each paper from the perspective of data (including type, domain, and dataset name), data science tasks, the proposed techniques and key contributions. In this way, papers can be retrieved to solve the different competitions from different perspectives. In the retrieval stage, the input task is analyzed with the same labeling tricks. For each label, AUTOMIND performs a similarity search in the knowledge base to retrieve associated knowledge. Then, after filtering out solutions or tricks of the same target task to avoid plagiarism, the retrieved results are re-ranked based on the aforementioned label priority order, on which we obtain the final retrieved knowledge.

3.2 Agentic Knowledgeable Tree Search

To facilitate the exploration of possible solutions, we model the search space as a solution tree $\mathcal{T} = (\mathcal{N}, \mathcal{E})$. Each node $N \in \mathcal{N} \subset \mathcal{T}$ corresponds to a unique solution $s = (p, \sigma, \eta)$ as formalized in §2, while each edge $E = (N_{\text{parent}}, N_{\text{child}}) \in \mathcal{E} \subset \mathcal{T}$ corresponds to the specific action applied to the parent node N_{parent} that produces the child node N_{child} . As illustrated in Figure 1, the search tree is initialized as a single empty node N_{empty} , after which AUTOMIND begins iteratively exploring the solution space. At the start of each iteration, the search policy π receives the current state of the

\mathcal{T} , selects one node as the N_{parent} according to the search policy, and specifies an action that generates a N_{child} and launches the next iteration of the exploration. Next, we examine the core components of the exploration framework in greater detail.

Solution Nodes (\mathcal{N}) Each solution node $N \in \mathcal{N}$ consists of following information:

- **Plan p :** an end-to-end textual solution plan typically comprises sequential stages including data pre-processing, feature engineering, model training, and prediction.
- **Code σ :** a Python implementation of the outlined solution plan p .
- **Output o :** the terminal output generated during the execution of code σ , which serves as a diagnostic feedback signal.
- **Metric η :** the task-specific validation score extracted from the terminal output o .

As shown in Figure 1, solution nodes are classified as either *valid* nodes $\mathcal{N}_{\text{valid}}$ or *buggy* nodes $\mathcal{N}_{\text{buggy}}$, based on whether their metrics can be correctly computed. Additionally, if a buggy node reaches the pre-defined max debug depth, it will be marked as a *dead* node, which will not be further selected by the search policy. The *best* node N_{best} is marked as the valid node with the optimal validation metric in the \mathcal{T} .

Action Edges (\mathcal{E}) The action space of AUTOMIND consists of three distinct operations: **Drafting** $\mathcal{A}_{\text{draft}}$, **Improving** $\mathcal{A}_{\text{improve}}$, and **Debugging** $\mathcal{A}_{\text{debug}}$. The action specified at each iteration is based on the type of the parent node N_{parent} selected by the search policy π , which can be $\mathcal{A}_{\text{draft}}$, $\mathcal{A}_{\text{improve}}$ or $\mathcal{A}_{\text{debug}}$ if the N_{parent} is empty, valid, or buggy respectively. As illustrated in Figure 1, each action goes through a similar pipeline processed through plan generation, code implementation, code execution, and output verification. However, different types of actions vary primarily in the specific inputs provided for the plan generation stage. In the $\mathcal{A}_{\text{draft}}$, AUTOMIND synthesizes the task description with relevant papers retrieved from the expert knowledge base to formulate an initial solution. In the $\mathcal{A}_{\text{improve}}$, AUTOMIND is provided with the valid N_{parent} —consisting of plan p , code σ and output o —as well as tricks retrieved from the expert knowledge base, and is instructed to improve the plan accordingly. In the $\mathcal{A}_{\text{debug}}$, AUTOMIND receives only the buggy N_{parent} and is instructed to modify the plan to resolve the bug. Procedures for

code implementation, execution, and output verification are uniformly applied across all action types. Once an action completes, the resulting solution s is encapsulated as a new node N_{child} within the \mathcal{T} as the child of the N_{parent} selected by the search policy π .

Search Policy (π) The search policy is driven by a stochastic heuristic tree search algorithm. Specifically, the policy first ensures the maximum number of draft nodes C_{init} is met, which splits the search tree into multiple branches, to lay the foundation for further exploration:

$$\pi_0(\mathcal{T}) = \begin{cases} (N_{\text{empty}}, \mathcal{A}_{\text{draft}}) & \text{if } |\mathcal{N}_{\text{draft}}| < C_{\text{init}}, \\ \pi_1(\mathcal{T}) & \text{otherwise} \end{cases},$$

where $\mathcal{N}_{\text{draft}}$ is the set of draft nodes in the \mathcal{T} . It then prioritizes debugging the buggy leaf nodes with a heuristic probability H_{debug} , thereby enabling the exploration of the potential within existing buggy solutions and facilitating more valid comparisons among all solutions in the \mathcal{T} :

$$\pi_1(\mathcal{T}) = \begin{cases} (N_{\text{buggy}}, \mathcal{A}_{\text{debug}}) & \text{if } p_1 < H_{\text{debug}} \text{ and } \mathcal{N}_{\text{buggy}} \neq \emptyset, \\ \pi_2(\mathcal{T}) & \text{otherwise} \end{cases}$$

where $p_1 \sim U(0, 1)$ and $N_{\text{buggy}} \sim U(\mathcal{N}_{\text{buggy}})$.

Subsequently, there is a heuristic probability of H_{greedy} that the current best node with the optimal metric in the \mathcal{T} , will be selected for improvement; and the remaining $1 - H_{\text{greedy}}$ are allocated to the selection of other valid nodes for improvement, helping to mitigate the risk of overlooking potentially better solutions:

$$\pi_2(\mathcal{T}) = \begin{cases} (N_{\text{best}}, \mathcal{A}_{\text{improve}}) & \text{if } p_2 < H_{\text{greedy}} \text{ and } \mathcal{N}_{\text{valid}} \neq \emptyset, \\ (N_{\text{valid}}, \mathcal{A}_{\text{improve}}) & \text{otherwise} \end{cases}$$

where $p_2 \sim U(0, 1)$ and $N_{\text{valid}} \sim U(\mathcal{N}_{\text{valid}})$.

If no further nodes remain to improve or debug, the policy will expand the search branches by creating a new draft node. We provide a more detailed illustration for the search policy π in Appendix A.

3.3 Self-Adaptive Coding Strategy

To cope with the spectrum of data science workloads from straightforward machine learning models to multi-stage, state-of-the-art architectures, we introduce within AUTOMIND a self-adaptive coding mechanism, reconciling solution complexity

Method	Backbone	MLE-Bench								Top AI Competitions		
		Easy		Medium		Hard		Overall		OAG	BELKA	Overall
		Best@3	Avg@3	Best@3	Avg@3	Best@3	Avg@3	Best@3	Avg@3	-	-	-
MLAB	GPT-4o [‡]	0.22	0.13	0.08	0.04	0.03	0.02	0.11	0.06	-	-	-
OpenHands	GPT-4o [‡]	0.48	0.28	0.08	0.07	0.17	0.08	0.24	0.15	-	-	-
AIDE	GPT-4o [‡]	0.71	0.53	0.26	0.15	0.13	0.07	0.37	0.25	-	-	-
	o3-mini	0.53	0.44	0.22	0.19	0.27	0.20	0.34	0.28	0.56	0.09	0.33
	DeepSeek-V3	0.79	0.58	0.36	0.31	0.28	0.21	0.48	0.36	0.52	0.33	0.43
AUTOMIND w/o Knowledge	DeepSeek-V3	0.85	0.54	0.31	0.21	0.40	0.24	0.52	0.33	0.50	0.19	0.35
AUTOMIND	o3-mini	0.81	0.60	0.50	0.33	0.32	0.23	0.54	0.39	0.55	0.44	0.50
	DeepSeek-V3	0.83	0.65	0.48	0.33	0.44	0.26	0.58	0.41	0.58	0.39	0.49

Table 1: **Main results on MLE-Bench and Top AI Competitions.** For MLE-Bench, we report both the best@3 and avg@3 win rates against human participants for all methods. For the OAG and BELKA competitions, we report the avg@3 official task metrics, which are AUC and AP, respectively. The **best** and **suboptimal** results for each task are highlighted. [‡] indicates that the results are borrowed from the grading reports of previous work.

with the coding capabilities of LLMs, as illustrated in Figure 1.

During the code implementation stage of an action, AUTOMIND follows a list of professional rubrics to score the overall complexity of the solution plan on a five-point scale. When this score falls below a preset threshold—indicating that the agent regards the plan as straightforward—the agent implements the entire code for the plan in one pass to maximize efficiency. However, if the score exceeds this threshold, the agent switches to a stepwise strategy by decomposing the plan into sequential substeps and incorporating execution feedback at each substep. Specifically, for each substep, the agent performs an Abstract Syntax Tree (AST) check and then executes the code in a terminal session. If the tests pass, the agent advances to the next substep; otherwise, the agent regenerates the substep’s implementation using the error messages as feedback. This loop repeats until either the tests succeed for all substeps, after which the agent integrates the substeps’ code into a complete implementation; or a predefined retry limit is reached for any substep, forcing the agent to abandon the current plan.

4 Experiments

4.1 Experimental Setup

Backbone Models In the main experiment, we evaluate the agents by systematically varying the backbone models. Specifically, we test three representative models: GPT-4o² and o3-mini³ from OpenAI, and DeepSeek-V3⁴ from DeepSeek.

²gpt-4o-2024-08-06

³o3-mini-2025-01-31

⁴DeepSeek-V3-0324

Baseline Agents Given the substantial computational resources necessary for baseline reproduction, the results from MLAB (Huang et al., 2024a), OpenHands (Wang et al., 2025), and AIDE (Jiang et al., 2025) on MLE-Bench utilizing GPT-4o as the backbone are adopted from prior work⁵ to serve as initial baselines. Subsequently, to facilitate a broader quantitative comparison, we re-execute AIDE using o3-mini and DeepSeek-V3 as alternative backbones within the scope of the main experiment. For each evaluation task, the agents are allocated a 24-hour time budget to produce their final submission. We repeat all experiments with 3 runs per task. Given the inherent high variance of agents when performing long-duration tasks, we report the **best@3** and **avg@3** performance metrics to provide a more robust and reliable assessment. We provide detailed runtime environment settings in Appendix C and necessary hyperparameters for reproduction in Appendix D.

4.2 Benchmarks

MLE-Bench We select MLE-Bench (Chan et al., 2025), which consists of 75 offline Kaggle competitions for evaluating LLM agents, as part of our test benchmarks. We further apply a rule-based filtering to the tasks in MLE-Bench as detailed in Appendix B. Consequently, we obtain a lite version of MLE-Bench consisting of 15 tasks, which are split into *Easy*, *Medium* and *Hard* tiers based on human experience and the results of previous works (Chan et al., 2025). We assess agent performance by comparing their submissions with official

⁵<https://github.com/openai/mle-bench/tree/main/runs>

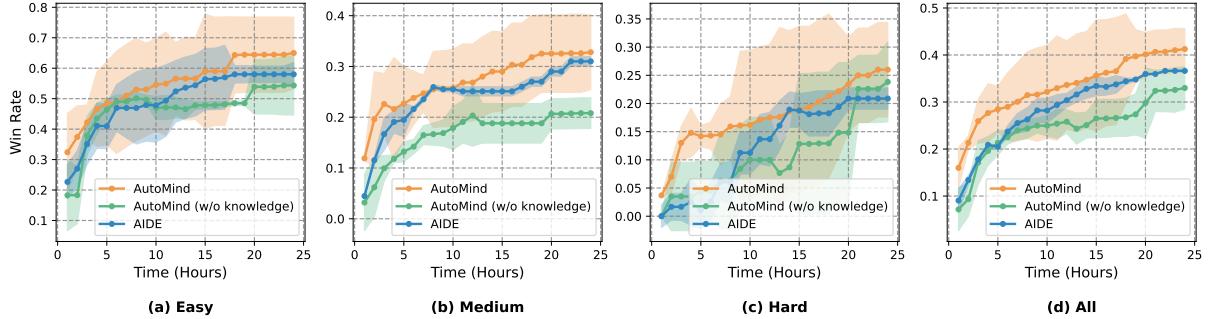


Figure 2: **Test time scaling results on MLE-Bench.** We record hourly snapshots of the percentage of human participants surpassed by the agent’s best solution over a 24-hour time budget in experiments with DeepSeek-V3.

competition leaderboards and report the win rate, which is defined as the proportion of human participants whose scores are surpassed by agents.

Top AI Competitions An inspection of the original MLE-Bench reveals that most tasks were curated before 2023, with several classical machine learning tasks dating to 2018 or earlier. Considering the fact that foundation models are likely seen corresponding tasks during pre-training, we supplement our evaluation with two tasks drawn from recent top AI competitions. Specifically, we include the WhoIsWho-IND track of the Open Academic Graph (OAG) Challenge at KDD Cup 2024 (Zhang et al., 2024), evaluated by the area under the ROC curve (AUC), and the BELKA Challenge at the NeurIPS 2024 Competition (Blevins et al., 2024), assessed by average precision (AP). We employ the official task metrics, evaluating the experimented agents directly by their raw scores. More details about the competitions are shown in Appendix B.

4.3 Main Results

As shown in Table 1, AUTOMIND consistently exceeds all baseline methods on the benchmarks under both the best@3 and avg@3 settings. We find that AUTOMIND (o3-mini) and AUTOMIND (DeepSeek-V3) outperform 38.7% and 41.2% of human participants on the official leaderboard of MLE-Bench respectively, representing performance gains of 11.0% and 5.2% over the prior SOTA AIDE under the avg@3 setting. Moreover, AUTOMIND exhibits remarkable superiority under the best@3 setting, achieving win rate improvements of 20.2% with o3-mini and 10.6% with DeepSeek-V3 over the prior SOTA. Across both the OAG and BELKA challenges in Top AI Competitions, AUTOMIND delivers performance that is at least on par with, and in most cases

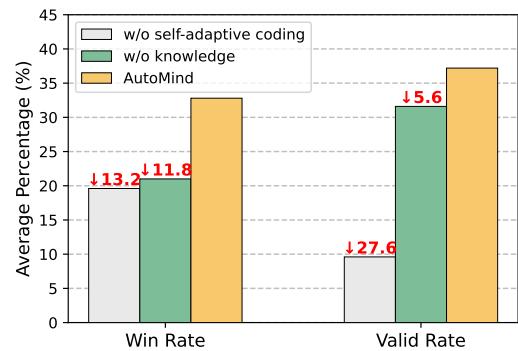


Figure 3: **Ablation studies** on DeepSeek-V3 for AUTOMIND on the *Medium* split of MLE-Bench. **Win Rate** represents the percentage of human participants surpassed by the agent on the official leaderboard. **Valid Rate** represents the percentage of valid submissions among all solutions the agent makes within a 24-hour time budget.

exceeds prior SOTA. Particularly, AUTOMIND (o3-mini) achieves an average precision of 0.44 on the BELKA challenge, representing a 0.35 absolute improvement over the prior SOTA.

5 Analysis

5.1 Ablation Study

To validate the effectiveness of our design, we conduct ablation experiments for AUTOMIND (DeepSeek-V3) on the *Medium* split of MLE-Bench, separately disabling two principal components in AUTOMIND: expert knowledge base and self-adaptive coding strategy.

Expert knowledge provides additional effective supervision for agentic tree search. When AUTOMIND is run without access to the expert knowledge base, the agent is forced to rely exclusively on its internal knowledge to draft and improve the solutions. The results in Figure 3 demonstrate that

ablation the expert knowledge base leads to respective declines of 11.8% and 5.6% in the win rate and valid rate metrics. Figure 2 presents hourly snapshots of the win rate metric over a 24-hour time budget, demonstrating that AUTOMIND equipped with the expert knowledge base consistently outperforms the variant without it. We attribute these performance gains to the integration of expert knowledge, which imposes additional constraints on the agent’s solution search space and acts like a “shortcut” to the collective craft of experienced Kaggle grand-masters and recent data science literature. By leveraging human-validated knowledge, AUTOMIND reduces its reliance on limited internal knowledge of backbone LLMs, avoids rediscovering effective ideas from scratch in the limited time budget, and focuses exploration within a more promising solution space.

Self-adaptive coding provides robust support for the implementation of more complex plans. We ablate the self-adaptive coding mechanism by completely replacing it with a one-pass coding strategy during the code implementation stage of AUTOMIND. The results in Figure 3 demonstrate that, replacing the self-adaptive coding mechanism with a one-pass strategy leads to respective declines of 13.2% and 27.6% in the win rate and valid rate metrics, highlighting its significant limitations in addressing complex tasks and plans. We attribute this decline to the limited coding capacity of the backbone LLMs, which proves insufficient to tackle complex tasks and plans in one-pass generation. By applying stepwise decomposition of complex plans and integrating AST check with execution feedback, error accumulation in the early segments of code generated by the one-pass strategy can be minimized, thereby preserving the efficient execution of subsequent code segments. As for simpler tasks and plans, the self-adaptive coding strategy inherently permits the utilization of one-pass generation, thereby striking a balance between efficiency and robustness in AUTOMIND.

5.2 Efficiency Analysis

Test-Time Scaling To assess the efficiency of different agent frameworks, we investigate test-time scaling by tracking the performance of both AUTOMIND and the prior SOTA AIDE over a 24-hour time budget in experiments with DeepSeek-V3. As shown in Figure 2, both agents are able to progressively improve their solutions as the available

test-time increases. Notably, on MLE-Bench, AUTOMIND achieves the prior SOTA’s 24-hour performance within an average of 15 hours, representing a 60% improvement in time efficiency.

Token Costs We quantify the cumulative token costs at the time by which each agent framework achieves the prior SOTA’s 24-hour performance. As shown in Table 2, owing to the efficiency improvements, AUTOMIND achieves a 9.6% reduction in token costs.

Agents	Input	Output	Total
AIDE (24h)	2.27 ± 0.28	0.22 ± 0.03	2.49 ± 0.31
AUTOMIND (15h)	2.15 ± 0.24	0.27 ± 0.04	2.25 ± 0.27

Table 2: **Token costs across all MLE-Bench tasks.** We present the input, output, and total token costs for experiments with DeepSeek-V3, each quantified in millions of tokens.

5.3 Case Study

As shown in Figure 4, we provide a case study on the BELKA task to verify the effectiveness of AUTOMIND. During execution, AUTOMIND first retrieves papers MolTrans (Huang et al., 2021) and DeepDTA (Öztürk et al., 2018) from the knowledge base, derives a frequent-subsequence mining strategy with dual-CNN blocks inspired by them, and then generates and runs code to implement the plan. On the contrary, AUTOMIND (w/o knowledge) focuses on extracting the statistical features of molecules and only adopts the simple MLPs to predict the binding probability. As for AIDE, the final solution employs a naive gradient boosting model, which is inadequate to tackle such a complex task. Compared with AIDE and AUTOMIND (w/o knowledge), AUTOMIND could retrieve the potential papers and design a more expressive model for complex tasks, the higher performance could demonstrate the effectiveness of the constructed knowledge base and retrieval strategy.

6 Related Work

LLM Agents. LLMs, with excellent reasoning (Qiao et al., 2023; Sun et al., 2025; Chen et al., 2025) and planning (Huang et al., 2024b; Wei et al., 2025a) abilities, are becoming the central control components of AI agents (Wang et al., 2024; Xi et al., 2023; Durante et al., 2024) and have been increasingly applied in software engineering (Qian

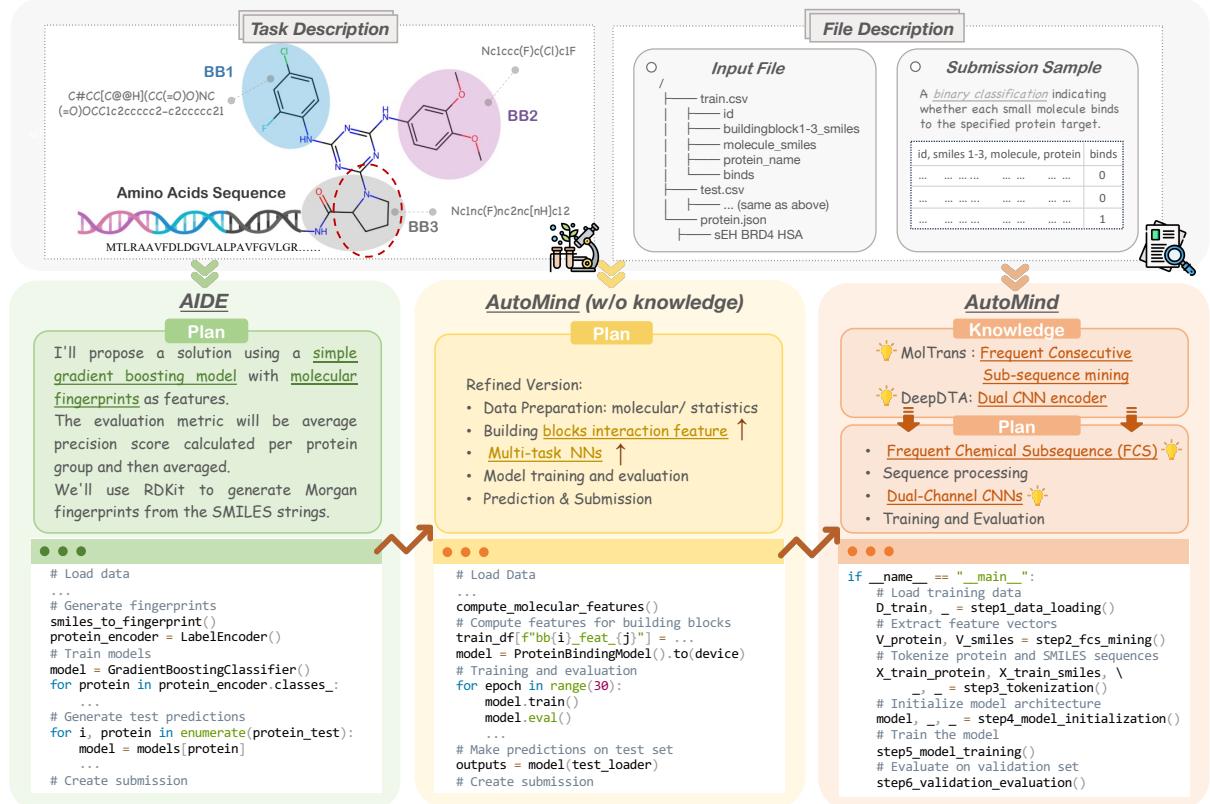


Figure 4: A running case on the BELKA challenge. We compare the proposed solution plans and corresponding code implementations generated by both AIDE and AUTOMIND.

et al., 2024; Hong et al., 2024b; Yang et al., 2024; Wei et al., 2025b), deep research (Li et al., 2025; Zheng et al., 2025b; Wu et al., 2025), GUI manipulation (Wu et al., 2024; Lai et al., 2024; Gou et al., 2024; Hu et al., 2024), scientific discovery (Chen et al., 2024; Hong et al., 2024a; Trirat et al., 2024), embodied intelligence (Ahn et al., 2022; Singh et al., 2023; Song et al., 2023), etc. Most current LLM agent frameworks rely on two paradigms. One is the training-free general architecture that depends on the strong capabilities of foundation models and carefully customized workflows (Hong et al., 2024b; Qian et al., 2024; Trirat et al., 2024; Li et al., 2024b). The other involves fine-tuning models in specific fields. Previous studies mainly focused on imitation learning based on a large amount of trajectory data (Chen et al., 2023; Zeng et al., 2023; Wu et al., 2024; Qiao et al., 2024). However, with the emergence of GRPO-like algorithms (Shao et al., 2024; Yu et al., 2025; Yue et al., 2025), models can now learn to complete target tasks through self-exploration with rule-based rewards (Song et al., 2025; Jin et al., 2025; Wei et al., 2025b; Lu et al., 2025; Feng et al., 2025).

LLM Agents for Data Science. Data science agents aim to leverage LLMs to automate data-centric machine learning tasks, including data analysis, data modeling, and data visualization, serving as a critical component for future AI agents to achieve autonomous scientific discovery. Most existing approaches decompose data science tasks into distinct subtasks with clear boundaries based on human expertise, executing them as workflows within single or multiple agents (Zhang et al., 2023; Li et al., 2024a; Guo et al., 2024; Li et al., 2024b). Furthermore, Hong et al. (2024a); Jiang et al. (2025); Trirat et al. (2024); Chi et al. (2024) employ reflection and search-based optimization. However, these methods overlook fundamental limitations in model capabilities: despite being trained on massive code datasets, the models inherently lack the rich empirical expertise accumulated by human practitioners in data science. To integrate human expertise, DS-Agent (Guo et al., 2024) adopts a knowledge-based approach by collecting expert Kaggle solutions and applying case-based reasoning to adapt these legacy solutions to new tasks. Additionally, the inherent complexity of data science task types necessitates diverse problem-solving

strategies, whereas current solutions predominantly apply uniform approaches across all tasks. To address these gaps, this paper proposes to enhance agent capabilities by incorporating human expertise (from research papers, Kaggle competitions, etc.) as an expert knowledge base, while implementing dynamic coding strategy selection mechanisms to adapt to different task requirements.

7 Conclusion and Future Work

We introduce AUTOMIND, an adaptive, knowledge-driven LLM-agent framework tailored for automated data science. By integrating three key innovations, including an expert knowledge base curated for data science, an agentic knowledgeable tree-search algorithm, and a self-adaptive coding strategy, AUTOMIND delivers superior performance versus state-of-the-art baselines on two automated data science benchmarks. Our experiments validate the effectiveness of AUTOMIND’s design and quantify its efficiency improvements, highlighting AUTOMIND as an efficient and robust step toward fully automated data science.

In the future, we envision extending this framework to create a fully autonomous, continuously evolving knowledge ecosystem, in which LLM agents not only read and synthesize papers and code but also generate novel insights, driving AI toward unprecedented levels of creativity, scientific discovery, and transformative impact across data-driven disciplines.

Limitations

Benchmarks and Baselines Due to limited computational resources, rather than evaluating the full set of 75 MLE-Bench (Chan et al., 2025) tasks, we select a representative subset of 15 tasks, chosen to span the entire spectrum of difficulty levels and task categories for our experiments.

Coding Capability of Backbone Models The performance of AUTOMIND is tightly coupled with the code generation proficiency of the underlying backbone model. If the coding capability of backbone models is insufficient for implementing complex solutions with high potential, our approach may lag behind previous data-science agents, which often favor simpler, easier-to-implement solutions. As a result, proprietary backbone models such as o3-mini and DeepSeek-V3 adopted in our experiments can better reflect the

advantages of our method for their potential to implement more complex and effective solutions.

References

- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alexander Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, and 24 others. 2022. *Do as I can, not as I say: Grounding language in robotic affordances*. *CoRR*, abs/2204.01691.
- Andrew Blevins, Ian K Quigley, Brayden J Halverson, Nate Wilkinson, Rebecca S Levin, Agastya Pulapaka, Walter Reade, and Addison Howard. 2024. Neurips 2024 - predict new medicines with belka. <https://kaggle.com/competitions/leash-BELKA>. Kaggle.
- Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Aleksander Madry, and Lilian Weng. 2025. *Mle-bench: Evaluating machine learning agents on machine learning engineering*. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. 2023. *Fireact: Toward language agent fine-tuning*. *CoRR*, abs/2310.05915.
- Qiguang Chen, Libo Qin, Jinhao Liu, Dengyun Peng, Jiannan Guan, Peng Wang, Mengkang Hu, Yuhang Zhou, Te Gao, and Wanxiang Che. 2025. *Towards reasoning era: A survey of long chain-of-thought for reasoning large language models*. *CoRR*, abs/2503.09567.
- Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. 2024. *Scienceagentbench: Toward rigorous assessment of language agents for data-driven scientific discovery*. *CoRR*, abs/2410.05080.
- Yizhou Chi, Yizhang Lin, Sirui Hong, Duyi Pan, Yaying Fei, Guanghao Mei, Bangbang Liu, Tianqi Pang, Jacky Kwok, Ceyao Zhang, Bang Liu, and Chenglin Wu. 2024. *SELA: tree-search enhanced LLM agents for automated machine learning*. *CoRR*, abs/2410.17238.
- Zane Durante, Qiuyuan Huang, Naoki Wake, Ran Gong, Jae Sung Park, Bidipta Sarkar, Rohan Taori, Yusuke Noda, Demetri Terzopoulos, Yejin Choi, Katsushi Ikeuchi, Hoi Vo, Li Fei-Fei, and Jianfeng Gao. 2024. *Agent AI: surveying the horizons of multimodal interaction*. *CoRR*, abs/2401.03568.

- Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. *Retool: Reinforcement learning for strategic tool use in llms*. *Preprint*, arXiv:2504.11536.
- Boyu Gou, Ruohan Wang, Boyuan Zheng, Yanan Xie, Cheng Chang, Yiheng Shu, Huan Sun, and Yu Su. 2024. *Navigating the digital world as humans do: Universal visual grounding for GUI agents*. *CoRR*, abs/2410.05243.
- Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. 2024. *Ds-agent: Automated data science by empowering large language models with case-based reasoning*. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhuo Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, and 6 others. 2024a. *Data interpreter: An LLM agent for data science*. *CoRR*, abs/2402.18679.
- Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiaowu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024b. *Metagpt: Meta programming for A multi-agent collaborative framework*. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, Yuhuai Li, Shengze Xu, Shawn Wang, Xinchen Xu, Shuofei Qiao, Kun Kuang, Tieyong Zeng, Liang Wang, Jiwei Li, and 9 others. 2024. Os agents: A survey on mllm-based agents for general computing devices use. <https://github.com/OS-Agent-Survey/OS-Agent-Survey/>.
- Kexin Huang, Cao Xiao, Lucas M Glass, and Jimeng Sun. 2021. Moltrans: molecular interaction transformer for drug–target interaction prediction. *Bioinformatics*, 37(6):830–836.
- Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024a. *Mlagentbench: Evaluating language agents on machine learning experimentation*. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024b. *Understanding the planning of LLM agents: A survey*. *CoRR*, abs/2402.02716.
- 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*. *CoRR*, abs/2502.13138.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. *Search-r1: Training llms to reason and leverage search engines with reinforcement learning*. *Preprint*, arXiv:2503.09516.
- Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2024. *Dsbench: How far are data science agents to becoming data science experts?* *CoRR*, abs/2409.07703.
- Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. *Autowebglm: A large language model-based web navigating agent*. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2024, Barcelona, Spain, August 25-29, 2024*, pages 5295–5306. ACM.
- Ruochen Li, Teerth Patel, Qingyun Wang, and Xinya Du. 2024a. *Mlr-copilot: Autonomous machine learning research based on large language models agents*. *CoRR*, abs/2408.14033.
- Xiaoxi Li, Jiajie Jin, Guanting Dong, Hongjin Qian, Yutao Zhu, Yongkang Wu, Ji-Rong Wen, and Zhicheng Dou. 2025. *Webthinker: Empowering large reasoning models with deep research capability*. *Preprint*, arXiv:2504.21776.
- Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, Wanjun Zhong, Wangchunshu Zhou, Wenhao Huang, and Ge Zhang. 2024b. *Autokaggle: A multi-agent framework for autonomous data science competitions*. *CoRR*, abs/2410.20424.
- Zexi Liu, Jingyi Chai, Xinyu Zhu, Shuo Tang, Rui Ye, Bo Zhang, Lei Bai, and Siheng Chen. 2025. *Ml-agent: Reinforcing llm agents for autonomous machine learning engineering*. *arXiv preprint arXiv:2505.23723*.
- Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Han Xiao, Shuai Ren, Guanjing Xiong, and Hongsheng Li. 2025. *Ui-r1: Enhancing action prediction of gui agents by reinforcement learning*. *Preprint*, arXiv:2503.21620.
- Hakime Öztürk, Arzucan Özgür, and Elif Ozkirimli. 2018. Deepdta: deep drug–target binding affinity prediction. *Bioinformatics*, 34(17):i821–i829.
- Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. *Chatdev: Communicative agents for software development*. In *Proceedings of the 62nd Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024, pages 15174–15186. Association for Computational Linguistics.

Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2023. **Reasoning with language model prompting: A survey**. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 5368–5393. Association for Computational Linguistics.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Eleanor Jiang, Chengfei Lv, and Huajun Chen. 2024. **Autoact: Automatic agent learning from scratch for QA via self-planning**. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3003–3021. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. **Deepseekmath: Pushing the limits of mathematical reasoning in open language models**. *CoRR*, abs/2402.03300.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. **Reflection: language agents with verbal reinforcement learning**. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. **Prog-prompt: Generating situated robot task plans using large language models**. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 11523–11530. IEEE.

Chan Hee Song, Brian M. Sadler, Jiaman Wu, Wei-Lun Chao, Clayton Washington, and Yu Su. 2023. **Llm-planner: Few-shot grounded planning for embodied agents with large language models**. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 2986–2997. IEEE.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. **R1-searcher: Incentivizing the search capability in llms via reinforcement learning**. *Preprint*, arXiv:2503.05592.

Jiankai Sun, Chuanyang Zheng, Enze Xie, Zhengying Liu, Ruihang Chu, Jianing Qiu, Jiaqi Xu, Mingyu Ding, Hongyang Li, Mengzhe Geng, Yue Wu, Wenhai Wang, Junsong Chen, Zhangyue Yin, Xiaozhe

Ren, Jie Fu, Junxian He, Yuan Wu, Qi Liu, and 15 others. 2025. **A survey of reasoning with foundation models: Concepts, methodologies, and outlook**. *ACM Comput. Surv.*

Maojun Sun, Ruijian Han, Binyan Jiang, Houduo Qi, Defeng Sun, Yancheng Yuan, and Jian Huang. 2024. **A survey on large language model-based agents for statistics and data science**. *CoRR*, abs/2412.14222.

Patara Trirat, Wonyong Jeong, and Sung Ju Hwang. 2024. **Automl-agent: A multi-agent LLM framework for full-pipeline automl**. *CoRR*, abs/2410.02958.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. **A survey on large language model based autonomous agents**. *Frontiers Comput. Sci.*, 18(6):186345.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 2 others. 2025. **Openhands: An open platform for AI software developers as generalist agents**. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025a. **Plangenllms: A modern survey of LLM planning capabilities**. *CoRR*, abs/2502.11221.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025b. **SWE-RL: advancing LLM reasoning via reinforcement learning on open software evolution**. *CoRR*, abs/2502.18449.

Junde Wu, Jiayuan Zhu, and Yuyuan Liu. 2025. **Agentic reasoning: Reasoning llms with tools for the deep research**. *CoRR*, abs/2502.04644.

Zhiyong Wu, Zhenyu Wu, Fangzhi Xu, Yian Wang, Qiushi Sun, Chengyou Jia, Kanzhi Cheng, Zichen Ding, Liheng Chen, Paul Pu Liang, and Yu Qiao. 2024. **OS-ATLAS: A foundation action model for generalist GUI agents**. *CoRR*, abs/2410.23218.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, and 10 others. 2023. **The rise and potential of large language model based agents: A survey**. *CoRR*, abs/2309.07864.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. 2025. **The ai scientist-v2: Workshop-level**

automated scientific discovery via agentic tree search.
Preprint, arXiv:2504.08066.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. **Swe-agent: Agent-computer interfaces enable automated software engineering**. *CoRR*, abs/2405.15793.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. **React: Synergizing reasoning and acting in language models**. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Qiyi Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, and 16 others. 2025. **Dapo: An open-source llm reinforcement learning system at scale**. *Preprint*, arXiv:2503.14476.

Yu Yue, Yufeng Yuan, Qiyi Yu, Xiaochen Zuo, Ruofei Zhu, Wenyuan Xu, Jiaze Chen, Chengyi Wang, TianTian Fan, Zhengyin Du, Xiangpeng Wei, Xiangyu Yu, Gaohong Liu, Juncai Liu, Lingjun Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Chi Zhang, and 8 others. 2025. **Vapo: Efficient and reliable reinforcement learning for advanced reasoning tasks**. *Preprint*, arXiv:2504.05118.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. 2023. **Agenttuning: Enabling generalized agent abilities for llms**. *CoRR*, abs/2310.12823.

Fanjin Zhang, Shijie Shi, Yifan Zhu, Bo Chen, Yukuo Cen, Jifan Yu, Yelin Chen, Lulu Wang, Qingfei Zhao, Yuqing Cheng, Tianyi Han, Yuwei An, Dan Zhang, Weng Lam Tam, Kun Cao, Yunhe Pang, Xinyu Guan, Huihui Yuan, Jian Song, and 3 others. 2024. **Oag-bench: A human-curated benchmark for academic graph mining**. *arXiv preprint arXiv:2402.15810*.

Wenqi Zhang, Yongliang Shen, Weiming Lu, and Yueteng Zhuang. 2023. **Data-copilot: Bridging billions of data and humans with autonomous workflow**. *CoRR*, abs/2306.07209.

Da Zheng, Lun Du, Junwei Su, Yuchen Tian, Yuqi Zhu, Jintian Zhang, Lanning Wei, Ningyu Zhang, and Hua-jun Chen. 2025a. **Knowledge augmented complex problem solving with large language models: A survey**. *Preprint*, arXiv:2505.03418.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025b. **Deepresearcher: Scaling deep research via reinforcement learning in real-world environments**. *Preprint*, arXiv:2504.03160.

A Search Policy

We provide a detailed illustration for the search policy π of AUTOMIND in Algorithm 1.

B Benchmarks

B.1 MLE-Bench

The original MLE-Bench (Chan et al., 2025) consists of 75 offline Kaggle competitions for evaluating LLM agents. We apply a rule-based filtering to the tasks in MLE-Bench. Specifically, we first exclude the tasks for which no valid submission can be made from the prior SOTA, thereby eliminating tasks that might be excessively difficult or ill-posed for LLM agents. From the remaining tasks, we then sample a balanced subset, retaining at least one and no more than two tasks per task category (e.g., image classification, training LLMs). Consequently, we obtain a lite version of MLE-Bench consisting of 15 tasks, which are split into *Easy*, *Medium* and *Hard* tiers based on human experience and the performance of prior SOTA. We include a full list of tasks in MLE-Bench used for evaluation in Table 3.

B.2 Top AI Competitions

BELKA dataset collects from kaggle competition⁶, which predict the binding affinity of small molecules to specific protein targets. It provides 2.9B molecule-protein pairs of training data. In this paper, we sample 2.2M training rows and 590K test rows from full training data following the label distribution, and uses the AP score to evaluate the methods.

OAG dataset used in this paper is collected from KDD cup Who-is-Who incorrect assignment detection task⁷. Given the paper assignments of each author and the metadata of each paper, this task is to detect paper assignment errors for each author. In this task, each paper contains the title, abstract, author name and corresponding organization, keywords, venue and year. The training data contains several authors and corresponding paper assignments. For simplicity, we sample 10k papers from the given paper list, and then preserve the corresponding authors related to these papers. These authors are classified into training and test data following the label distribution used in training data.

⁶<https://www.kaggle.com/competitions/leash-BELKA>

⁷<https://www.biendata.xyz/kdd2024>

Algorithm 1 Search Policy π in AUTOMIND

Require: \mathcal{T} ▷ Current state of solution tree

Require: C_{init} ▷ Hyper-parameter of the counts of initial draft nodes

Require: H_{debug} ▷ Hyper-parameter of the probability of whether debug a node

Require: H_{greedy} ▷ Hyper-parameter of the probability of whether select the best node

Ensure: $(\mathcal{N}, \mathcal{A})$ ▷ Select a parent node and specify an action applied on it

- 1: $C_{\text{draft}} \leftarrow$ Get the counts of draft nodes in the \mathcal{T}
- 2: **if** $C_{\text{draft}} < C_{\text{init}}$ **then return** $(N_{\text{empty}}, \mathcal{A}_{\text{draft}})$ ▷ Draft a new solution
- 3: **end if**
- 4: $p_{\text{debug}} \leftarrow$ Get a random floating-point number from 0 to 1
- 5: $N_{\text{buggy}} \leftarrow$ Get a random buggy leaf node in the \mathcal{T}
- 6: **if** $p_{\text{debug}} < H_{\text{debug}}$ and N_{buggy} is not None **then return** $(N_{\text{buggy}}, \mathcal{A}_{\text{debug}})$ ▷ Debug a buggy solution
- 7: **end if**
- 8: $p_{\text{greedy}} \leftarrow$ Get a random floating-point number from 0 to 1
- 9: $N_{\text{best}} \leftarrow$ Get the best node in the solution tree
- 10: $N_{\text{valid}} \leftarrow$ Get a random valid node in the solution tree
- 11: **if** $p_{\text{greedy}} < H_{\text{greedy}}$ and N_{best} is not None **then**
- 12: **return** $(N_{\text{best}}, \mathcal{A}_{\text{improve}})$ ▷ Improve the current best valid solution
- 13: **else if** $p_{\text{greedy}} \geq H_{\text{greedy}}$ and N_{valid} is not None **then**
- 14: **return** $(N_{\text{valid}}, \mathcal{A}_{\text{improve}})$ ▷ Mitigate getting trapped in local optima
- 15: **end if**
- 16: **return** $(N_{\text{empty}}, \mathcal{A}_{\text{draft}})$ ▷ No solution to debug or improve, draft a new solution

Tasks	Task Type	Dataset Size (GB)	Metric Type	Original Complexity	Split
aptos2019-blindness-detection	Image Classification	10.22	Max	Low	Easy
random-acts-of-pizza	Text Classification	0.003	Max	Low	Easy
spooky-author-identification	Text Classification	0.0019	Min	Low	Easy
google-quest-challenge	Training LLMs	0.015	Max	Medium	Easy
stanford-covid-vaccine	Tabular	2.68	Min	High	Easy
predict-volcanic-eruptions-ingv-oe	Signal Processing	31.25	Min	High	Easy
lmsys-chatbot-arena	Text Classification	0.18	Min	Medium	Medium
us-patent-phrase-to-phrase-matching	Text Regression	0.00214	Max	Medium	Medium
mlsp-2013-birds	Audio Classification	0.5851	Max	Low	Medium
statoil-iceberg-classifier-challenge	Image Classification	0.3021	Min	Medium	Medium
tensorflow-speech-recognition-challenge	Audio Classification	3.76	Max	Medium	Medium
denoising-dirty-documents	Image to Image	0.06	Min	Low	Hard
new-york-city-taxi-fare-prediction	Tabular	5.7	Min	Low	Hard
tgs-salt-identification-challenge	Image Segmentation	0.5	Max	Medium	Hard
ventilator-pressure-prediction	Forecasting	0.7	Min	Medium	Hard

Table 3: Full list of tasks in MLE-Bench used for evaluation in our work.

The evalution metric is $\sum_1^M w_i \times AUC_i$ where M is the test author number,

$$w_i = \frac{\#ErrorsOfThisAuthor}{\#TotalErrors}.$$

C Runtime Environment

In our experiments, LLM agents are loaded into an Ubuntu 20.04 Docker container containing the dataset prepared for each task, and an Anaconda environment pre-installed with standard Python packages for machine learning (e.g., PyTorch, scikit-learn), thereby providing all requisite dependencies for code implementation and execution. The container runs on a compute node with 48 vCPUs, 448GB RAM, 9.6TB SSD storage, and a single NVIDIA Tesla V100 32G GPU, all of which are fully accessible to the agents.

D Hyperparameters

We list the detailed hyperparameters for AUTO-MIND and AIDE in Table 4 and Table 5, respectively.

E Prompts

In this section, we showcase some of the prompts used in the full pipeline of AUTOMIND, which serve as a reference.

Hyperparameter	Value
agent.retriever.model	gpt-4.1-mini-2025-04-14
agent.analyzer.model	gpt-4.1-mini-2025-04-14
agent.planner.model	&TARGET_MODEL
agent.coder.model	&TARGET_MODEL
agent.improver.model	&TARGET_MODEL
agent.verifier.model	gpt-4.1-mini-2025-04-14
agent.steps	2000
agent.search.num_drafts	5
agent.search.max_debug_depth	5
agent.search.debug_prob	1
agent.search.trick_prob	0.8
agent.search.greedy_prob	0.8
agent.time_limit	86400
exec.timeout	32400

Table 4: Hyperparameters for AUTOMIND. &TARGET_MODEL is the foundation model being evaluated. agent.search.num_drafts is the number of initial draft nodes. agent.search.debug_prob is the probability of whether debug a node. agent.search.trick_prob is the probability of whether use tricks to improve a node. agent.search.greedy_prob is the probability of whether select the best node.

Hyperparameter	Value
agent.code.model	&TARGET_MODEL
agent.feedback.model	gpt-4.1-mini-2025-04-14
agent.steps	2000
agent.search.num_drafts	5
agent.search.max_debug_depth	20
agent.search.debug_prob	1
agent.time_limit	86400
exec.timeout	32400

Table 5: Hyperparameters for AIDE. &TARGET_MODEL is the foundation model being evaluated. agent.search.num_drafts is the number of initial draft nodes. agent.search.debug_prob is the probability of whether debug a node.

Prompt for plan generation of drafting

Introduction

You are an expert machine learning engineer attempting a task. In order to complete this task, you need to come up with an excellent and creative plan for a solution, which will be implemented by another engineer. We will now provide a description of the task.

Task description

{task_description}

Memory

Take the Memory section into consideration when proposing the solution plan, don't propose the similar solution but keep the evaluation metric exactly the same.

{memory}

Knowledge

Some of the tricks that have proved useful for the same type of task are provided as follows:

{tricks}

You should carefully consider these tricks when designing your solution.

Data Analysis

{data_analysis}

Instructions

Response format

Your response should be a detailed outline/sketch of your proposed solution in natural language. You do not need to implement the solution but you should provide enough detail for another engineer to implement it in Python code. There should be no additional headings or text in your response. Just natural language text followed by a newline.

Installed Packages

Your solution can use any relevant machine learning packages such as: 'torch-geometric==2.6.1', 'xgboost==2.1.3', 'torchvision==0.17.0', 'lightgbm==4.5.0', 'transformers==4.44.2', 'matplotlib==3.9.2', 'scipy==1.11.4', 'statsmodels==0.14.4', 'pandas==2.1.4', 'torch==2.2.0', 'optuna==4.0.0', 'timm==0.9.7', 'scikit-learn==1.2.2', 'humpy==1.26.2', 'bayesian-optimization==1.5.1', 'seaborn==0.13.2'. Feel free to use any other packages too (all packages are already installed!). For neural networks please use PyTorch because of the unavailability of TensorFlow in the environment.

Plan guideline

- DO NOT CHEAT ON EVALUATION. The solution should calculate the evaluation metric described in the task description on a hold-out validation set.
- If the evaluation metric is not provided, you should propose a reasonable evaluation metric for the task and calculate it.
- The solution should print the evaluation metric computed on the hold-out validation set at the last step of the solution.
- Try to come up with more modern and powerful methods to feature engineering and modelling and avoid using outdated methods. For example, if the task is a classification task, you should use modern transformer-based models instead of traditional models like CNN or LSTM.

- The solution should adopt appropriate methods to prevent model overfitting, such as data augmentation, early stopping, regularization, dropout, and others.
- Don't suggest to do model ensembling.
- Don't suggest to do Exploratory Data Analysis.
- Don't suggest to do hyperparameter tuning.
- The data is already prepared and available in the './input' directory. There is no need to unzip any files.
- The solution should use os.walk to get the paths of all available files in the './input' directory for data loading.
- If a 'sample_submission.csv' file exists, directly load it and use it as a template for the 'submission.csv' file. The solution should save predictions on the provided unlabeled test data in the 'submission.csv' file in the ./submission/ directory.
- Prefer and explicitly use GPU (CUDA) acceleration when available: move models/tensors to GPU and handle CPU fallback if CUDA is not present.

Prompt for plan generation of debugging

Introduction

You are an expert machine learning engineer attempting a task. You are provided with the plan, code and execution output of a previous solution below that had a bug and/or did not produce a submission.csv, and should improve it in order to fix the bug. For this you should first propose a reasonable improvement and accordingly outline a detailed improved plan in natural language, which will be implemented by another engineer. We will now provide a description of the task.

Task description

```
{task_description}
```

Previous Solution

Previous Plan

```
{prev_plan}
```

Previous Code

```
{prev_code}
```

Previous Execution Output

```
{prev_output}
```

Data Analysis

```
{data_analysis}
```

Instructions

Response format

First, provide a brief explanation of your reasoning for the proposed improvement to the previous plan (wrapped in <think></think>). Then, provide a detailed outline/sketch of your improved solution in natural language based on the previous plan and your proposed improvement (wrapped in <plan></plan>). You do not need to implement the solution but you should provide enough detail for another engineer to implement it in Python code.

Installed Packages

Your solution can use any relevant machine learning packages such as: 'torch-geometric==2.6.1', 'xgboost==2.1.3', 'torchvision==0.17.0', 'lightgbm==4.5.0', 'transformers==4.44.2', 'matplotlib==3.9.2', 'scipy==1.11.4', 'statsmodels==0.14.4', 'pandas==2.1.4', 'torch==2.2.0', 'optuna==4.0.0', 'timm==0.9.7', 'scikit-learn==1.2.2', 'numpy==1.26.2', 'bayesian-optimization==1.5.1', 'seaborn==0.13.2'. Feel free to use any other packages too (all packages are already installed!). For neural networks please use PyTorch because of the unavailability of TensorFlow in the environment.

Improve guideline

- You should pay attention to the execution output of the previous solution, and propose an improvement that will fix the bug.
- The improved plan should be derived by adapting the previous plan only based on the proposed improvement, while retaining other details of the previous plan. Don't suggest to do Exploratory Data Analysis.
- Don't suggest to do hyperparameter optimization, you should use the best hyperparameters from the previous solution.
- If a 'sample_submission.csv' file exists, directly load it and use it as a template for the 'submission.csv' file. The solution should save predictions on the provided unlabeled test data in the 'submission.csv' file in the ./submission/ directory.
- When describing your improved plan, do not use phrases like 'the same as before' or 'as in the previous plan'. Instead, fully restate all details from the previous plan that you want to retain, as subsequent implementation will not have access to the previous plan.

Prompt for plan generation of improving with tricks

Introduction

You are an expert machine learning engineer attempting a task. You are provided with the plan, code and execution output of a previous solution below and should improve it in order to further increase the test time performance. For this you should integrate several useful tricks provided and accordingly outline a detailed improved plan in natural language, which will be implemented by another engineer. We will now provide a description of the task.

Task description

{task_description}

Memory

Take the Memory section into consideration when proposing the solution plan, don't propose the similar solution but keep the evaluation metric exactly the same.

{memory}

Previous Solution

Previous Plan

{prev_plan}

Previous Code

{prev_code}

Previous Execution Output

```
{prev_output}

# Knowledge
Here are some tricks that have proved useful for the task:
{tricks}
You should carefully consider these tricks when designing your solution.
```

```
# Data Analysis
{data_analysis}
```

```
# Instructions
```

```
## Response format
```

First, provide a brief explanation of your reasoning for the proposed improvement to the previous plan (wrapped in <think></think>). Then, provide a detailed outline/sketch of your improved solution in natural language based on the previous plan and your proposed improvement (wrapped in <plan></plan>). You do not need to implement the solution but you should provide enough detail for another engineer to implement it in Python code.

```
## Installed Packages
```

Your solution can use any relevant machine learning packages such as: 'torch-geometric==2.6.1', 'xgboost==2.1.3', 'torchvision==0.17.0', 'lightgbm==4.5.0', 'transformers==4.44.2', 'matplotlib==3.9.2', 'scipy==1.11.4', 'statsmodels==0.14.4', 'pandas==2.1.4', 'torch==2.2.0', 'optuna==4.0.0', 'timm==0.9.7', 'scikit-learn==1.2.2', 'numpy==1.26.2', 'bayesian-optimization==1.5.1', 'seaborn==0.13.2'. Feel free to use any other packages too (all packages are already installed!). For neural networks please use PyTorch because of the unavailability of TensorFlow in the environment.

```
## Improve guideline
```

- You should focus ONLY on integrating the provided tricks in the knowledge section into the previous solution to fully leverage their potentials.
- Make sure to fully integrate these tricks into your plan while preserving as much details as possible.
- Ensure that your plan clearly demonstrates the functions and specifics of the tricks.
- Identify the key areas in the previous solution where the knowledge can be applied.
- Suggest specific changes or additions to the code or plan based on the knowledge provided, and avoid unnecessary modifications irrelevant to the tricks.
- If a 'sample_submission.csv' file exists, directly load it and use it as a template for the 'submission.csv' file. The solution should save predictions on the provided unlabeled test data in the 'submission.csv' file in the ./submission/ directory.
- When describing your improved plan, do not use phrases like 'the same as before' or 'as in the previous plan'. Instead, fully restate all details from the previous plan that you want to retain, as subsequent implementation will not have access to the previous plan.

Prompt for plan generation of improving without tricks

```
# Introduction
```

You are an expert machine learning engineer attempting a task. You are provided with the plan, code and execution output of a previous solution below and should improve it in order to further increase the test time performance. For this you should first propose a reasonable improvement

and accordingly outline a detailed improved plan in natural language, which will be implemented by another engineer. We will now provide a description of the task.

Task description
{task_description}

Memory

Take the Memory section into consideration when proposing the solution plan, don't propose the similar solution but keep the evaluation metric exactly the same.

{memory}

Previous Solution

Previous Plan
{prev_plan}

Previous Code
{prev_code}

Previous Execution Output
{prev_output}

Data Analysis
{data_analysis}

Instructions

Response format

First, provide a brief explanation of your reasoning for the proposed improvement to the previous plan (wrapped in <think></think>). Then, provide a detailed outline/sketch of your improved solution in natural language based on the previous plan and your proposed improvement (wrapped in <plan></plan>). You do not need to implement the solution but you should provide enough detail for another engineer to implement it in Python code.

Installed Packages

Your solution can use any relevant machine learning packages such as: 'torch-geometric==2.6.1', 'xgboost==2.1.3', 'torchvision==0.17.0', 'lightgbm==4.5.0', 'transformers==4.44.2', 'matplotlib==3.9.2', 'scipy==1.11.4', 'statsmodels==0.14.4', 'pandas==2.1.4', 'torch==2.2.0', 'optuna==4.0.0', 'timm==0.9.7', 'scikit-learn==1.2.2', 'numpy==1.26.2', 'bayesian-optimization==1.5.1', 'seaborn==0.13.2'. Feel free to use any other packages too (all packages are already installed!). For neural networks please use PyTorch because of the unavailability of TensorFlow in the environment.

Improve guideline

- You should conduct only one expert-level actionable improvement to the previous solution.
- This improvement should be atomic so that the effect of the improved solution can be experimentally evaluated.
- The improved plan should be derived by adapting the previous plan only based on the proposed improvement, while retaining other details of the previous plan.
- Don't suggest to do Exploratory Data Analysis.

- Don't suggest to do hyperparameter optimization, you should use the best hyperparameters from the previous solution.
- If a 'sample_submission.csv' file exists, directly load it and use it as a template for the 'submission.csv' file. The solution should save predictions on the provided unlabeled test data in the 'submission.csv' file in the ./submission/ directory.
- When describing your improved plan, do not use phrases like 'the same as before' or 'as in the previous plan'. Instead, fully restate all details from the previous plan that you want to retain, as subsequent implementation will not have access to the previous plan.

Prompt for complexity scorer

Introduction

You are an expert machine learning engineer attempting a task. In order to complete this task, you are given a description of the task and a solution plan proposed by another engineer and need to assess the complexity of the task and the proposed solution. We will now provide a description of the task.

Task description
 {task_description}

Proposed Solution
 {proposed_solution}

Data Analysis
 {data_analysis}

Instructions

Response format

First, provide a brief explanation of your reasoning for the assessment of the complexity of the task and the proposed solution (wrapped in <think></think>). Then, provide ONLY ONE average complexity score as floating point number between 1 and 5, which can contain 0.5 points (wrapped in <score></score>).

Task complexity scoring criteria

- 5 = Extremely complex and cutting-edge task with high levels of innovation required. This involves solving a unique or highly specialized problem that may push the boundaries of existing knowledge or technology.
- 4 = Complex task that involves advanced techniques or methodologies, requiring considerable expertise in the domain, such as building novel algorithms, optimization methods, or handling advanced data.
- 3 = Moderately complex task that requires significant problem-solving, such as integrating different methods or creating custom algorithms for specific use cases.
- 2 = Simple task with some level of complexity, such as basic algorithms that need some degree of fine-tuning or adjustment to meet the specific requirements of the project.
- 1 = Very simple task that requires minimal effort, such as basic data manipulation or applying standard algorithms without any customization.

Proposed solution complexity scoring criteria

- 5 = A groundbreaking or transformative solution that pushes the envelope in the field. It introduces a novel approach that is scalable, efficient, and offers long-term value or sets a new standard.
- 4 = A highly original and effective solution that shows a deep understanding of the problem domain and offers a significant contribution to the field. The solution is well-optimized and efficient.
- 3 = An original and creative solution with a reasonable level of complexity. It involves designing and implementing custom solutions or combining existing methods in a new way.
- 2 = A somewhat original solution that involves adapting existing tools or methods with some customization to meet the needs of the project. There may be room for optimization or improvement.
- 1 = Very basic solution, perhaps using standard, off-the-shelf tools with minimal adaptation, lacking originality or novel contributions.

Complexity scoring guideline

- Evaluate the complexity of the task and the proposed solution, and assign a score between 1 and 5.
- Assign an average score between 1 and 5, consider factors such as the task's complexity, the proposed solution, the dataset size, and the time and hardware resources required for implementation and execution.

Prompt for code implementation through one-pass coding

Introduction

You are an expert machine learning engineer attempting a task. In order to complete this task, you are given a description of the task and a solution plan proposed by another engineer and need to assess the complexity of the task and the proposed solution. We will now provide a description of the task.

Task description

{task_description}

Proposed Solution

{proposed_solution}

Data Analysis

{data_analysis}

Instructions

Response format

Your response should be a single markdown code block (wrapped in "") which implements this solution plan and prints out and save the evaluation metric.

Installed Packages

Your solution can use any relevant machine learning packages such as: 'torch-geometric==2.6.1', 'xgboost==2.1.3', 'torchvision==0.17.0', 'lightgbm==4.5.0', 'transformers==4.44.2', 'matplotlib==3.9.2', 'scipy==1.11.4', 'statsmodels==0.14.4', 'pandas==2.1.4', 'torch==2.2.0', 'optuna==4.0.0', 'timm==0.9.7', 'scikit-learn==1.2.2', 'numpy==1.26.2', 'bayesian-optimization==1.5.1', 'seaborn==0.13.2'. Feel free to use any other packages too (all packages are already installed!). For neural networks please use PyTorch because of the unavailability of TensorFlow in the environment.

Code guideline

- The code should **implement the proposed solution** and **print the value of the evaluation metric computed on a hold-out validation set**,
- **AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A 'submission.csv' FILE IN THE ./submission/ DIRECTORY.**
- The code should save the evaluation metric computed on the hold-out validation set in a 'eval_metric.txt' file in the ./submission/ directory.
- The code should be a single-file python program that is self-contained and can be executed as-is.
- No parts of the code should be skipped, don't terminate the code before finishing the script.
- DO NOT WRAP THE CODE IN A MAIN FUNCTION, BUT WRAP ALL CODE in the '__main__' module, or it cannot be executed successfully.
- All class initializations and computational routines MUST BE WRAPPED in 'if __name__ == "__main__":'.
- DO NOT USE MULTIPROCESSING OR SET 'num_workers' IN DATA LOADER, as it may cause the container to crash.
- Your response should only contain a single code block.
- All input data is already prepared and available in the './input' directory. There is no need to unzip any files.
- DO NOT load data from "./data" directory, it is not available in the environment.
- Do not save any intermediate or temporary files through 'torch.save' or 'pickle.dump'.
- Try to accelerate the model training process if any GPU is available.
- DO NOT display progress bars. If you have to use function intergrated with progress bars, disable progress bars or using the appropriate parameter to silence them.
- Don't do Exploratory Data Analysis.
- Avoid printing detailed model architecture information unless debugging. When debugging model issues, use concise shape tracking during forward pass to quickly identify problematic layers without verbose model summaries.
- When debugging data-related errors, please refer to the data analysis section first for insights about data structure and format.
- **DO NOT HARDCODE OR FAKE THE EVALUATION METRIC VALUE. The metric must be computed from actual model performance on validation data.**

Prompt for stepwise decomposition

Introduction

You are an expert machine learning engineer attempting a task. In order to complete this task, you are given the proposed solution and supposed to decompose it into multiple steps. We will now provide a description of the task.

Task description
{task_description}

Proposed Solution
{proposed_solution}

Instructions

Response format

- Your response should be a single JSON code block (wrapped in "") which contains the decomposition steps of the proposed solution.

- The generated JSON should have the following format:

```
{
  "decomposed steps": [
    {
      "step": "Name of the step",
      "details": "Detailed description of the step",
    },
    ...
  ],
}
```

Solution decomposition guideline

- You should decompose the proposed solution into multiple steps, and provide detailed descriptions of each step.
- DO NOT MODIFY THE PROPOSED SOLUTION. In the description of each step, you should keep as many details of the proposed solution as possible, especially the exact hyperparameters and sample code.
- DO NOT CHEAT ON EVALUATION. The solution should calculate the evaluation metric described in the task description on a hold-out validation set.
- If the evaluation metric is not provided, you should propose a reasonable evaluation metric for the task and calculate it.
- The solution should save the evaluation metric computed on the hold-out validation set in a 'eval_metric.txt' file in the ./submission/ directory.
- The solution should use os.walk to get the paths of all available files in the '. /input' directory for data loading.
- If a sample_submission.csv file exists, directly load it and use it as a template for the 'submission.csv' file. The solution should save predictions on the provide unlabeled test data in the 'submission.csv' file in the ./submission/ directory.
- You should **print the value of the evaluation metric computed on a hold-out validation set** in the last step of the decomposed steps.
- Don't do Exploratory Data Analysis in the decomposition steps.
- If you find improvements suggestions in the plan, you should take them in serious consideration and include them in the decomposition steps.
- You do not need to implement the code in the decomposed steps.
- Note that the order of the decomposed steps determines the order in which the code is implemented and executed.

Prompt for code implementation through stepwise coding

Introduction

You are an expert machine learning engineer attempting a task. In order to complete this task, you are given the code for previous steps and need to implement the current step of a natural language solution plan proposed by another engineer in Python code. We will now provide a description of the task.

```
# Task description
{task_description}
```

```
# Current Step
{current_step}
```

Previous Steps Code

You should continue the following code for previous steps to implement the current step of the solution plan, but do not repeat it:

```
{prev_steps}
```

Data Analysis

```
{data_analysis}
```

Instructions

Response format

First, provide suggestions for the current step based on the previous steps and the failed last try step if provided (wrapped in <think></think>). Then, provide a single markdown code block (wrapped in "") which implements the current step of a solution plan.

Installed Packages

Your solution can use any relevant machine learning packages such as: 'torch-geometric==2.6.1', 'xgboost==2.1.3', 'torchvision==0.17.0', 'lightgbm==4.5.0', 'transformers==4.44.2', 'matplotlib==3.9.2', 'scipy==1.11.4', 'statsmodels==0.14.4', 'pandas==2.1.4', 'torch==2.2.0', 'optuna==4.0.0', 'timm==0.9.7', 'scikit-learn==1.2.2', 'numpy==1.26.2', 'bayesian-optimization==1.5.1', 'seaborn==0.13.2'. Feel free to use any other packages too (all packages are already installed!). For neural networks please use PyTorch because of the unavailability of TensorFlow in the environment.

Code guideline

- You should implement the current step of the solution plan based on the previous steps and the failed last try step if provided.
- **You should ONLY implement the code for the current step of the solution plan, rather than the entire solution plan.**
- DO NOT MODIFY THE CURRENT STEP. You should implement the current step exactly as it is.
- You should **print the value of the evaluation metric computed on a hold-out validation set** if it is calculated in the current step.
- You should save the evaluation metric computed on the hold-out validation set in an 'eval_metric.txt' file in the './submission/' directory if it is calculated in the current step.
- DO NOT PRINT ANYTHING ELSE IN THE CODE, except for the evaluation metric and a concise completion message for the current step.
- **DO NOT REPEAT the code for previous steps; you should only import them from 'prev_steps.py'.**
- DO NOT REPETITIVELY IMPORT THE SAME MODULES ALREADY USED IN PREVIOUS STEPS, but you may import additional modules if needed.
- **AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A 'submission.csv' FILE IN THE './submission/' DIRECTORY** if predictions are involved in the current step.
- You can reference the based code to implement the current step, but do not completely repeat it.
- **DO NOT HARDCODE OR FAKE THE EVALUATION METRIC VALUE.** It must be computed from actual model performance on validation data.
- The code should be a single-file Python program that is self-contained and can be executed as-is.
- DO NOT wrap the code in a main function, BUT WRAP ALL CODE in the '__main__' module, or it cannot be executed successfully.

- All class initializations and computational routines MUST BE WRAPPED in 'if __name__ == "__main__":'.
- DO NOT USE MULTIPROCESSING OR SET 'num_workers' in any DataLoader.
- No parts of the code should be skipped; do not terminate early.
- All input data is already prepared and available in the './input' directory. There is no need to unzip any files.
- DO NOT load data from the './data' directory (not available).
- Do not save any intermediate or temporary files through 'torch.save' or 'pickle.dump'.
- Feel free to use GPU in any stage if it is available.
- DO NOT display progress bars; disable them or silence via parameters.
- Don't do Exploratory Data Analysis.
- Avoid printing detailed model architecture information unless debugging. For debugging, use concise tensor shape tracking.
- When debugging data-related errors, first refer to the data analysis section for structure/format insights.

Prompt for debugging through one-pass coding

Introduction

You are debugging a failed ML code step. Use precise SEARCH/REPLACE format to fix errors.

Task description

{task_description}

Improved Solution Plan

{improved_solution_plan}

Failed Code

{failed_code}

Instructions

- Diff Format

Use EXACT SEARCH/REPLACE format:

<<<<< SEARCH

exact code to replace (must match exactly)

=====

new code

>>>>> REPLACE

The SEARCH block must match the code exactly, including whitespace. Focus on targeted fixes, not full rewrites. You can make multiple changes with multiple diff blocks. Explain the reasoning for each change.

- The code should **implement the proposed solution** and **print the value of the evaluation metric computed on a hold-out validation set**,
- **AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A 'submission.csv' FILE IN THE ./submission/ DIRECTORY.**
- The code should save the evaluation metric computed on the hold-out validation set in a 'eval_metric.txt' file in the ./submission/ directory.
- DO NOT HARDCODE OR FAKE THE EVALUATION METRIC VALUE. The metric must be computed from actual model performance on validation data.
- The code should be a single-file python program that is self-contained and can be executed as-is.

- DO NOT WRAP THE CODE IN A MAIN FUNCTION, BUT WRAP ALL CODE in the '`__main__`' module, or it cannot be executed successfully.
- All class initializations and computational routines MUST BE WRAPPED in '`if __name__ == "__main__":`'.
- DO NOT USE MULTIPROCESSING OR SET '`num_workers`' IN DATA LOADER, as it may cause the container to crash.
- No parts of the code should be skipped, don't terminate the code before finishing the script.
- All input data is already prepared and available in the '`./input`' directory. There is no need to unzip any files.
- DO NOT load data from '`./data`' directory, it is not available in the environment.
- Do not save any intermediate or temporary files through '`torch.save`' or '`pickle.dump`'.
- Feel free to use GPU in any stage if it is available.
- DO NOT display progress bars. If you have to use function integrated with progress bars, disable progress bars or use the appropriate parameter to silence them.
- Don't do Exploratory Data Analysis.
- Avoid printing detailed model architecture information unless debugging. When debugging model issues, use concise shape tracking during forward pass to quickly identify problematic layers without verbose model summaries.
- When debugging data-related errors, please refer to the data analysis section first for insights about data structure and format.

Prompt for debugging in stepwise coding

Introduction

You are debugging a failed ML code step. Use precise SEARCH/REPLACE format to fix errors.

Current Step

{current_step}

Failed Code

{failed_code}

Error Output

{error_output}

Instructions

- IMPORTANT

- You can ONLY modify the Failed Code shown above

- Do NOT search for code from previous steps

- Your SEARCH blocks must match code in the Failed Code section exactly

- Focus only on fixing the current step's implementation

- Diff Format

Use EXACT SEARCH/REPLACE format:

<<<<< SEARCH

exact code to replace (must match exactly)

=====

new code

>>>>> REPLACE

The SEARCH block must match the code exactly, including whitespace. Focus on targeted fixes, not full rewrites. You can make multiple changes with multiple diff blocks. Explain the

reasoning for each change.

- The code should ****implement the proposed solution**** and ****print the value of the evaluation metric computed on a hold-out validation set****,
- ****AND MOST IMPORTANTLY SAVE PREDICTIONS ON THE PROVIDED UNLABELED TEST DATA IN A 'submission.csv' FILE IN THE ./submission/ DIRECTORY.****
- The code should save the evaluation metric computed on the hold-out validation set in a 'eval_metric.txt' file in the ./submission/ directory.
- **DO NOT HARDCODE OR FAKE THE EVALUATION METRIC VALUE.** The metric must be computed from actual model performance on validation data.
- The code should be a single-file python program that is self-contained and can be executed as-is.
- **DO NOT WRAP THE CODE IN A MAIN FUNCTION, BUT WRAP ALL CODE** in the '`__main__`' module, or it cannot be executed successfully.
- All class initializations and computational routines **MUST BE WRAPPED** in '`if __name__ == "__main__":`'.
- **DO NOT USE MULTIPROCESSING OR SET 'num_workers'** IN DATA LOADER, as it may cause the container to crash.
- No parts of the code should be skipped, don't terminate the code before finishing the script.
- All input data is already prepared and available in the './input' directory. There is no need to unzip any files.
- **DO NOT load data from './data'** directory, it is not available in the environment.
- Do not save any intermediate or temporary files through '`torch.save`' or '`pickle.dump`'.
- Feel free to use GPU in any stage if it is available.
- **DO NOT display progress bars.** If you have to use function integrated with progress bars, disable progress bars or use the appropriate parameter to silence them.
- Don't do Exploratory Data Analysis.
- Avoid printing detailed model architecture information unless debugging. When debugging model issues, use concise shape tracking during forward pass to quickly identify problematic layers without verbose model summaries.
- When debugging data-related errors, please refer to the data analysis section first for insights about data structure and format.

```
# Previous Steps Code  
Continue from (DO NOT MODIFY): {prev_code}
```

Prompt for output verification

Introduction

You are an expert machine learning engineer attempting a task. You have written code to solve this task and now need to evaluate the output of the code execution. You should determine if there were any bugs as well as report the empirical findings.

```
# Task description  
{task_description}
```

```
# Code  
{code}
```

```
# Execution Output  
{execution_output}
```

```

# Tool
{
  "type": "function",
  "function": {
    "name": "submission_verify",
    "description": "Verify the execution output of the written code.",
    "parameters": {
      "type": "object",
      "properties": {
        "is_bug": {
          "type": "boolean",
          "description": "true if the output log shows that the execution failed or has some bug, otherwise false.",
        },
        "is_overfitting": {
          "type": "boolean",
          "description": "true if the output log shows that the model is overfitting or validation metric is much worse than the training metric or validation loss is increasing, otherwise false. ",
        },
        "has_csv_submission": {
          "type": "boolean",
          "description": "true if the code saves the predictions on the test data in a 'submission.csv' file in the './submission/' directory, otherwise false. Note that the file MUST be saved in the ./submission/ directory for this to be evaluated as true, otherwise it should be evaluated as false. You can assume the ./submission/ directory exists and is writable.",
        },
        "summary": {
          "type": "string",
          "description": "write a short summary (2-3 sentences) describing the empirical findings. Alternatively mention if there is a bug or the submission.csv was not properly produced. You do not need to suggest fixes or improvements.",
        },
        "metric": {
          "type": "number",
          "description": "If the code ran successfully, report the value of the validation metric. Otherwise, leave it null.",
        },
        "lower_is_better": {
          "type": "boolean",
          "description": "true if the metric should be minimized (i.e. a lower metric value is better, such as with MSE), false if the metric should be maximized (i.e. a higher metric value is better, such as with accuracy).",
        },
      },
      "required": ["is_bug", "is_overfitting", "has_csv_submission", "summary", "metric", "lower_is_better"],
    },
  },
}

```