MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement

Jaehyun Nam[1 2 *], **Jinsung Yoon**[1], **Jiefeng Chen**[1]
Jinwoo Shin[2], **Sercan Ö. Arık**[1], **Tomas Pfister**[1]
[1]Google Cloud, [2]KAIST
jaehyun.nam@kaist.ac.kr, jinsungyoon@google.com

Abstract

Agents based on large language models (LLMs) for machine learning engineering (MLE) can automatically implement ML models via code generation. However, existing approaches to build such agents often rely heavily on inherent LLM knowledge and employ coarse exploration strategies that modify the entire code structure at once. This limits their ability to select effective task-specific models and perform deep exploration within specific components, such as experimenting extensively with feature engineering options. To overcome these, we propose *MLE-STAR*, a novel approach to build MLE agents. MLE-STAR first leverages external knowledge by using a search engine to retrieve effective models from the web, forming an initial solution, then iteratively refines it by exploring various strategies targeting specific ML components. This exploration is guided by ablation studies analyzing the impact of individual code blocks. Furthermore, we introduce a novel ensembling method using an effective strategy suggested by MLE-STAR. Our experimental results show that MLE-STAR achieves medals in 64% of the Kaggle competitions on the MLE-bench, significantly outperforming the best alternative.[1]

1 Introduction

The proliferation of machine learning (ML) has driven high-performance applications across diverse real-world scenarios, from fundamental tasks like tabular classification [1, 2, 3] to complex ones such as image denoising [4]. Despite these advances, developing such models remains a labor-intensive process for data scientists, involving extensive iterative experimentation and data engineering [5, 6]. To streamline such intensive workflows, recent research has focused on employing large language models (LLMs) [7, 8, 9] as *machine learning engineering (MLE) agents* [10, 11, 12]. By harnessing the coding and reasoning capabilities inherent in LLMs [13, 14], these agents conceptualize ML tasks as code optimization problems. They then navigate the potential code solutions ultimately producing executable code (*e.g.*, a Python script) based on a provided task description and dataset (see Figure 1).

Despite their promise as pioneering efforts, current MLE agents face several obstacles that limit their effectiveness. First, due to their strong reliance on inherent LLM knowledge, they are often biased toward familiar and frequently used methods (*e.g.*, the scikit-learn library [15] for tabular data), neglecting potentially promising task-specific methods. Additionally, these agents [10, 12] typically employ an exploration strategy that modifies the entire code structure at once in each iteration. This often results in agents pivoting prematurely to other steps (*e.g.*, model selection or hyperparameter tuning) because they lack the ability to perform deep, iterative exploration within specific pipeline components, such as experimenting different feature engineering options extensively.

[*]This work was done while Jaehyun was a student researcher at Google Cloud.
[1]We release open-source codebase of MLE-STAR at https://github.com/google/adk-samples.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

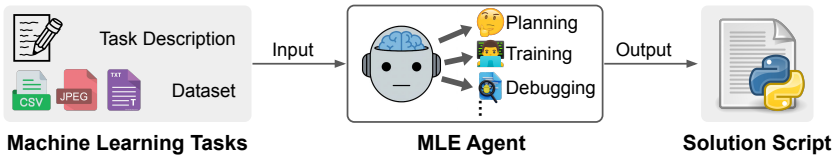

Figure 1: **Problem setup.** ML Engineering agents are designed to process a task description and datasets across various modalities (*e.g.*, tabular, text, image, audio, etc.) with the objective of determining the optimal solution for a given machine learning problem, such as classification, regression, sequence-to-sequence generation, image denoising, text normalization, etc.

Contributions. We propose **MLE-STAR**, a novel **ML** **E**ngineering agent that integrates web **S**earch and **TA**rgeted code block **R**efinement (see Figure 2 for an overview). Specifically, generating initial solution code, MLE-STAR utilizes Google Search to retrieve relevant and potentially state-of-the-art approaches that could be effective towards building a model. Moreover, to improve the solution, MLE-STAR extracts a specific code block that represents a distinct ML pipeline component, such as feature engineering or ensemble building, and then concentrates on exploring strategies that are targeted to that component, using previous attempts as feedback to reflect on. Here, to identify the code block that has the greatest impact on performance, MLE-STAR performs an ablation study that evaluates the contribution of each ML component. This refinement process is repeated, modifying various code blocks (*i.e.*, other ML components). In addition, we introduce a novel method to generate ensembles. MLE-STAR first proposes multiple candidate solutions. Then, instead of relying on a simple voting based on validation scores, MLE-STAR merges these candidates into a single improved solution using an ensemble strategy proposed by the agent itself. This ensemble strategy is iteratively refined based on the performance of the previous strategies.

To verify the effectiveness, we conduct comprehensive evaluations of MLE-STAR using the MLE-bench's Kaggle competitions [16]. The experimental results demonstrate that MLE-STAR, requiring only minimal human effort (*e.g.*, defining initial prompts that are generalizable to any tasks), significantly outperforms previous methods [12], including those requiring manual labor to collect strategies from Kaggle [10]. In particular, MLE-STAR achieves a substantial gain in medal achievement, improving it from 36.6% to 63.6% when compared to the top-performing baseline. Additionally, we show that our proposed ensemble technique provides a meaningful improvement to MLE-STAR.

2 Related work

LLM agents. Recent advances in LLMs have led to an active research in autonomous agents. General-purpose agents like ReAct [17] and HuggingGPT [18] typically use external tools to analyze various problems. Specialized agents, such as Voyager [19] for Minecraft or AlphaCode [20] for code generation, excel in specific domains, often using execution feedback to iteratively improve their approach. Extending these, we introduce MLE-STAR, an LLM agent that specialized in ML tasks.

Automated machine learning. Automated machine learning (AutoML) aims to reduce reliance on human experts by automating end-to-end ML pipelines [21, 22, 23]. Auto-WEKA [24], TPOT [25], and recent advances such as AutoGluon [26], have made progress through exploring within predefined model or hyperparameter spaces. AutoML research also specializes in areas such as neural network design [27, 28, 29, 30], and feature engineering [31, 32, 33, 34, 35]. However, these methods rely on predefined search spaces, which often require domain expertise to define. To address this, LLM-based MLE agents [10, 12], including MLE-STAR, are emerging, since they employ effective exploration strategies directly in the code space, without the need of manually-curated search spaces.

MLE agents. Leveraging coding and reasoning capabilities of LLMs [13, 14], research has been conducted on use of LLMs as MLE agents [11, 36, 37], which generate solution code, to automate ML workflows. While MLAB [38] and OpenHands [39] take general actions by calling tools to perform ML tasks, several studies specialize in ML automation. AIDE [12] generates candidate solutions in a tree structure to facilitate code space exploration. However, its heavy reliance on the LLM's internal knowledge can lead to outdated or overly simple model choices, and its refinement may prematurely shift focus between pipeline stages. DS-Agent [10] uses case-based reasoning [40, 41] to discover strategies for solution generation by utilizing manually curated cases (primarily from

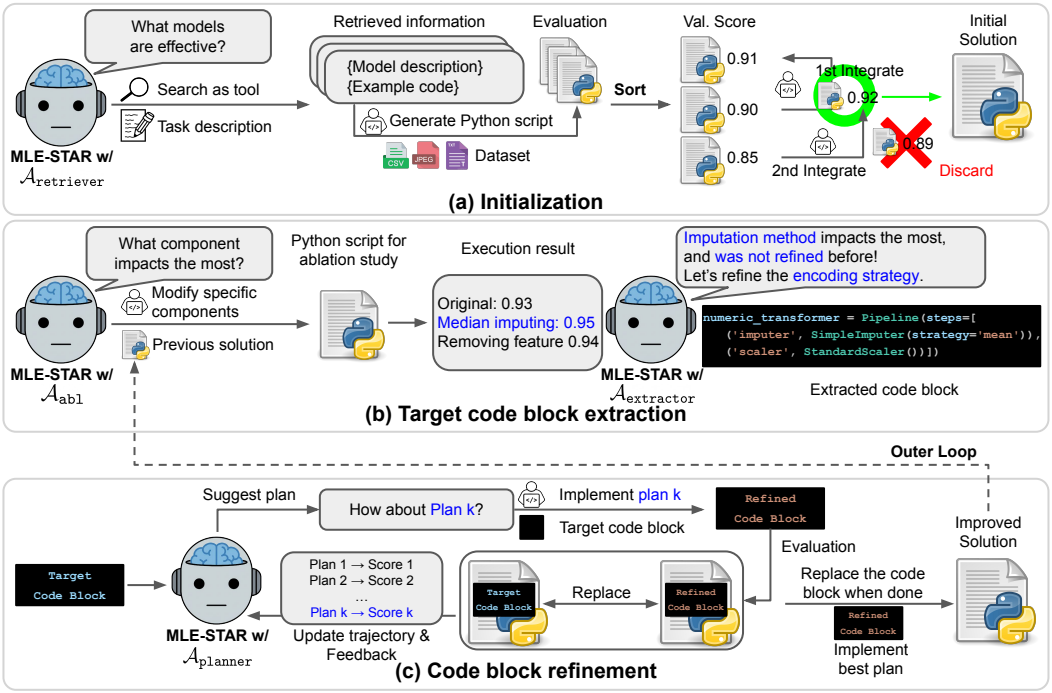

Figure 2: **Overview of MLE-STAR.** (a) Using search as a tool, MLE-STAR retrieves task-specific models and uses them to generate an initial solution. (b) In each refinement step, MLE-STAR performs an ablation study to extract the code block that have the greatest impact. Previously modified code blocks are also provided as feedback for diversity. (c) The extracted code block is iteratively refined based on plans suggested by the LLM, which explores various plans using previous experiments as feedback (*i.e.*, inner loop), and the target code block is also selected repeatedly (*i.e.*, outer loop, where the improved solution of (c) becomes the previous solution in (b)).

Kaggle). However, DS-Agent suffers from scalability issues due to its reliance on a manually built case bank, which requires significant human effort and can lead to solutions that are overfit to the source patterns. Also, it restricts applicability to novel task types (like complex multi-modal problems). Our method addresses these limitations. Instead of attempting to explore the broader code space or relying on a static case bank, MLE-STAR strategically explores implementation options for specific ML pipeline components. It also improves scalability by using LLMs with search as tool to retrieve effective models that fit the task beyond the constraints of a fixed case bank.

3 MLE-STAR

We introduce the proposed framework for MLE agents, MLE-STAR, that effectively leverages the coding and reasoning capabilities of LLMs to solve ML tasks. In a nutshell, our approach is based on first generating an initial solution by using web search as a tool (Section 3.1), and then refining solutions via nested loops. The outer loop targets one code block, which corresponds to the specific ML component extracted through an ablation study. The inner loop iteratively refines *only* this block until the outer loop moves to the next target (Section 3.2). We propose a novel ensemble method that improves the performance using the plan proposed by LLMs, which is iteratively refined (Section 3.3). To mitigate potential undesirable behaviors from LLMs, such as using test sample statistics for missing value imputation, we introduce specific modules (detailed in Section 3.4). The prompts and algorithms used in each step can be found in Appendix A and B, respectively.

Problem setup. Formally, our goal is to find an optimal solution $s^* = \arg\max_{s \in \mathcal{S}} h(s)$, where $\mathcal{S}$ is the space of possible solutions (*i.e.*, Python scripts) and $h : \mathcal{S} \to \mathbb{R}$ is a score function (*e.g.*, validation accuracy) [12]. To obtain s^*, we propose a multi-agent framework $\mathcal{A}$, which takes datasets $\mathcal{D}$ (that might contain multiple files) and a task description $\mathcal{T}_{\texttt{task}}$ (which includes task types, data

modalities, score functions, etc.) as input.[2] Here, $\mathcal{A}$ consists of n LLM agents $(\mathcal{A}_1, \cdots, \mathcal{A}_n)$. Each agent $\mathcal{A}_i$ possesses specific functionalities, which are elaborated upon in following sections.

3.1 Generating an initial solution using web search as a tool

Candidate model search. MLE-STAR starts by generating an initial solution. For high performance in ML tasks, selecting the appropriate model is paramount. However, relying solely on an LLM for model suggestions can lead to suboptimal choices. For instance, we observe that LLMs propose models like logistic regression [15] even for competitions like jigsaw-toxic-comment-classification, which is a text classification task, potentially because LLMs favor familiar patterns from their pre-training data over up-to-date information. To mitigate this, we propose using web search as a tool for MLE-STAR first to retrieve M effective, state-of-the-art models for the given task. This retrieved context is then used to guide the LLM in generating a more informed initial solution. Formally:

$$\{\mathcal{T}^i_{\texttt{model}}, \mathcal{T}^i_{\texttt{code}}\}_{i=1}^{M} = \mathcal{A}_{\texttt{retriever}}(\mathcal{T}_{\texttt{task}}), \tag{1}$$

where $\mathcal{T}_{\texttt{model}}$ represents the description of a retrieved model, while $\mathcal{T}_{\texttt{code}}$ provides corresponding example code. This example code is needed since the LLM can be unfamiliar with the model and cannot generate the executable code without proper guidance. Then, MLE-STAR involves evaluating of the performance of model i. To achieve this, candidate evaluation agent $\mathcal{A}_{\texttt{init}}$ first generates code, $s^i_{\texttt{init}}$, using the retrieved model to solve the given ML task. This process is formally defined as:

$$s^i_{\texttt{init}} = \mathcal{A}_{\texttt{init}}(\mathcal{T}_{\texttt{task}}, \mathcal{T}^i_{\texttt{model}}, \mathcal{T}^i_{\texttt{code}}). \tag{2}$$

We evaluate the performance of each s using a task-specific metric h on dataset $\mathcal{D}$. We denote the resulting score by $h(s)$, which encapsulates the entire process done in s: splitting $\mathcal{D}$ into training and validation sets, training the model specified in s using the training data, and calculating h on the validation data. The performance for $s^i_{\texttt{init}}$ is thus $h(s^i_{\texttt{init}})$. As a result, a set of code scripts $\mathcal{S}_{\texttt{init}} = \{s^1_{\texttt{init}}, \cdots, s^M_{\texttt{init}}\}$ and their performance scores $\{h(s^1_{\texttt{init}}), \cdots, h(s^M_{\texttt{init}})\}$ are obtained.

Merging candidate models for initial solution. After the evaluation of the M retrieved models, a consolidated initial solution s_0 is constructed through an iterative merging procedure. Specifically, we first define π be a permutation of the indices such that the scores are sorted in descending order: $h(s^{\pi(1)}_{\texttt{init}}) \geq h(s^{\pi(2)}_{\texttt{init}}) \geq \cdots \geq h(s^{\pi(M)}_{\texttt{init}})$. Then, we initialize the initial solution s_0 with the top-performing script, and record the current best score, *i.e.*, $s_0 \leftarrow s_{(1)}$, $h_{\texttt{best}} \leftarrow h(s_0)$, where $s_{(k)}$ denote the script $s^{\pi(k)}_{\texttt{init}}$ for simplicity. Finally, we sequentially attempt to incorporate the remaining scripts $s_{(k)}$ for $k = 2, \cdots, M$ into s_0. For each k, MLE-STAR creates a candidate merged script by leveraging an agent $\mathcal{A}_{\texttt{merger}}$ that attempts to integrate $s_{(k)}$ into the current s_0. Formally,

$$s_0 \leftarrow \mathcal{A}_{\texttt{merger}}(s_0, s_{(k)}),\ h_{\texttt{best}} \leftarrow h(s_0) \tag{3}$$

where, $\mathcal{A}_{\texttt{merger}}$ is guided to introduce a simple average ensemble to merge multiple models. Finally, we merge the models until the validation score $h_{\texttt{best}}$ no longer improves (see Appendix B).

3.2 Refining a code block for solution improvement

The iterative refinement phase begins with an initial solution s_0 and proceeds for a predetermined number of T outer loop steps, indexed by $t = 0, 1, \cdots, T-1$. At each step t, the goal is to improve the current solution s_t to obtain s_{t+1}, optimizing for a performance metric h. This process involves two main stages: targeted code block extraction and code block refinement.

Targeted code block extraction. To effectively explore specialized improvement strategies, MLE-STAR identifies and targets specific code blocks within the ML pipeline represented by s_t. This selection is guided by an ablation study performed by an agent $\mathcal{A}_{\texttt{abl}}$. Specifically, the agent $\mathcal{A}_{\texttt{abl}}$ generates a code a_t designed to perform an ablation study on s_t. This script creates variations of s_t by modifying or disabling specific components. To encourage exploration of different pipeline parts across iterations, $\mathcal{A}_{\texttt{abl}}$ receives the summaries of previous ablation studies $\{\mathcal{T}^i_{\texttt{abl}}\}_{i=0}^{t-1}$ as input:

$$a_t = \mathcal{A}_{\texttt{abl}}(s_t, \{\mathcal{T}^i_{\texttt{abl}}\}_{i=0}^{t-1}). \tag{4}$$

[2] MLE-STAR works across any data modalities (*e.g.*, tabular, image, text, audio) and task types (*e.g.*, classification, image-to-image, sequence-to-sequence) – it is not restricted to specific inputs or objectives.

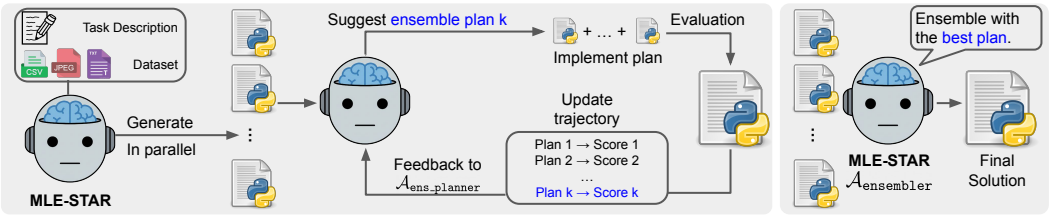

Figure 3: **Ensembling solutions.** MLE-STAR iteratively proposes effective ensemble strategies based on previous attempts, integrating multiple solutions generated in parallel into a single solution.

Then, a_t is executed, producing output results r_t (see examples in Appendix C). A summarization module $\mathcal{A}_{\texttt{summarize}}$ processes the script and its results to generate a concise ablation summary $\mathcal{T}_{\texttt{abl}}^t$:

$$\mathcal{T}_{\texttt{abl}}^t = \mathcal{A}_{\texttt{summarize}}(a_t, r_t). \tag{5}$$

Finally, MLE-STAR selects a critical block which has the most significant impact on performance. To achieve this, an extractor module $\mathcal{A}_{\texttt{extractor}}$ analyzes the $\mathcal{T}_{\texttt{abl}}^t$ to identify the code block c_t within s_t whose modification had the most significant impact on performance. To prioritize refinement of blocks not previously targeted, the set of already refined blocks $\{c_i\}_{i=0}^{t-1}$ is provided as context:

$$c_t, p_0 = \mathcal{A}_{\texttt{extractor}}(\mathcal{T}_{\texttt{abl}}^t, s_t, \{c_i\}_{i=0}^{t-1}). \tag{6}$$

Here, MLE-STAR also generates the initial plan p_0 for code block refinement at the same time, since $\mathcal{T}_{\texttt{abl}}$ can provide a good starting point by analyzing the modification of corresponding component.

Code block refinement. Once the targeted code block c_t is defined, MLE-STAR explores various refinement strategies to improve the metric h. This involves an inner loop exploring K potential refinement for c_t. An agent $\mathcal{A}_{\texttt{coder}}$ first implements p_0, transforming c_t into a refined block c_t^0, *i.e.*, $c_t^0 = \mathcal{A}_{\texttt{coder}}(c_t, p_0)$. A candidate solution s_t^0 is formed by substituting c_t^0 into s_t:

$$s_t^0 = s^t.\texttt{replace}(c_t, c_t^0), \tag{7}$$

where, `replace` denotes the code replacement operation. Finally, the performance $h(s_t^0)$ is evaluated.

To discover potentially more effective or novel refinement strategies, MLE-STAR iteratively generates and evaluates further plans. For $k = 1, \cdots, K-1$, a planning agent $\mathcal{A}_{\texttt{planner}}$ proposes the next plan p_k. This agent leverages the previous attempts within the current outer step t as feedback:

$$p_k = \mathcal{A}_{\texttt{planner}}(c_t, \{(p_j, h(s_t^j))\}_{j=0}^{k-1}). \tag{8}$$

For each plan p_k, the coding agent generates the corresponding refined block, *i.e.*, $c_t^k = \mathcal{A}_{\texttt{coder}}(c_t, p_k)$, creates the candidate solution $s_t^k = s_t.\texttt{replace}(c_t, c_t^k)$, and evaluates its performance $h(s_t^k)$. After exploring K refinement strategies (indexed $k = 0, \cdots, K-1$), the best-performing candidate solution is identified: $k^* = \arg\max_{k \in \{0, \cdots, K-1\}} h(s_t^k)$. The solution for the next outer step, s_{t+1}, is updated to $s_t^{k^*}$ only if an improvement over s_t is found. This iterative process continues until $t = T$.

3.3 Further improvement by exploring ensemble strategies

To further improve upon the best single solution generated, we introduce a novel ensembling procedure (Figure 3). Standard practice might involve generating multiple candidate solutions and selecting the one with the highest score [42] according to metric h. However, analogous to model ensembling, we posit that suboptimal solutions might contain complementary strengths, and combining multiple solutions could lead to superior performance compared to relying on any single one. Therefore, we employ the planning capabilities of MLE-STAR to automatically discover effective strategies for ensembling. Specifically, let $\{s_l\}_{l=1}^L$ be a set of L distinct solutions obtained (*e.g.*, from parallel runs of the process described earlier). Our goal is to find an effective ensemble plan e that merges these solutions, which mirrors the structure of the targeted code block refinement stage. We start with an initial ensemble plan e_0 (*e.g.*, a simple strategy like averaging the final predictions obtained from the models trained using each solution s_l), proposed by MLE-STAR itself. After the performance $h(s_{\texttt{ens}}^0)$ for the initial plan e_0 is calculated, for a fixed number of iterations, $r = 1, \cdots, R$, the planning agent $\mathcal{A}_{\texttt{ens\_planner}}$, specialized in suggesting ensemble plans, proposes subsequent ensemble plans e_r.

This agent uses the history of previously attempted ensemble plans and their resulting performance as feedback, *i.e.*, $e_r = \mathcal{A}_{\texttt{ens\_planner}}(\{s_l\}_{l=1}^{L}, \{(e_j, h(s_{\texttt{ens}}^j))\}_{j=0}^{r-1})$. Each e_r is implemented via $\mathcal{A}_{\texttt{ensembler}}$ to obtain $s_{\texttt{ens}}^r$:

$$s_{\texttt{ens}}^r = \mathcal{A}_{\texttt{ensembler}}(e_r, \{s_l\}_{l=1}^{L}). \tag{9}$$

Finally, after exploring R ensemble strategies, the ensemble result that achieves the highest performance is selected as the final output, yielding the final ensembled result $s_{\texttt{ens}}^* = s_{\texttt{ens}}^{r^*}$: $r^* = \arg\max_{r \in \{0,\dots,R\}} h(s_{\texttt{ens}}^r)$. This procedure allows MLE-STAR to autonomously explore and identify potentially novel and effective ways to combine multiple complex solutions.

3.4 Additional modules for robust MLE agents

Debugging agent. We detail the design of our debugging agent within MLE-STAR. If the execution of a Python script s triggers an error, resulting in a record $\mathcal{T}_{\texttt{bug}}$ (*e.g.*, a traceback), MLE-STAR employs a debugging module $\mathcal{A}_{\texttt{debugger}}$ to attempt correction. This process iteratively updates the script:

$$s \leftarrow \mathcal{A}_{\texttt{debugger}}(s, \mathcal{T}_{\texttt{bug}}). \tag{10}$$

The debugging step is repeated until either the script executes successfully, or a predefined maximum number of debugging rounds is reached. If the bug cannot be resolved, MLE-STAR proceeds to the next task using the latest version of the script that is known to be executable.

Data leakage checker. We observe that LLM-generated Python scripts might have the risk of introducing data leakage, for example, by improperly accessing information from a test dataset during training dataset preparation (see Figure 6). To address this, we introduce a checker agent, $\mathcal{A}_{\texttt{leakage}}$, which analyzes the solution script s prior to its execution. Recognizing that full-script analysis can be inefficient for lengthy code, we adopt a targeted approach. First, we extract the code block $c_{\texttt{data}}$ where data preprocessing is done. Second, $c_{\texttt{data}}$ is passed to the checker. If $\mathcal{A}_{\texttt{leakage}}$ detects potential data leakage, it generates a corrected version $c_{\texttt{data}}^*$: $c_{\texttt{data}}^* = \mathcal{A}_{\texttt{leakage}}(c_{\texttt{data}})$. Finally, the original script s is updated by replacing the identified segment with its corrected version: $s \leftarrow s.\texttt{replace}(c_{\texttt{data}}, c_{\texttt{data}}^*)$. If no leakage is detected in $c_{\texttt{data}}$ by $\mathcal{A}_{\texttt{leakage}}$, the script s remains unmodified. All generated solutions are passed through a data leakage checker, $\mathcal{A}_{\texttt{leakage}}$, prior to their execution for evaluation.

Data usage checker. We observe that LLM-generated scripts sometimes neglect using provided data sources, focusing solely on simple formats like CSVs (see Figure 7). To ensure the utilization of all relevant provided data, MLE-STAR introduces a data usage checker agent, $\mathcal{A}_{\texttt{data}}$. Specifically, before MLE-STAR starts refinement, $\mathcal{A}_{\texttt{data}}$ checks the initial solution s_0 along with the task description $\mathcal{T}_{\texttt{task}}$. If relevant provided data is not adequately used, $\mathcal{A}_{\texttt{data}}$ revises the initial script as:

$$s_0 \leftarrow \mathcal{A}_{\texttt{data}}(s_0, \mathcal{T}_{\texttt{task}}). \tag{11}$$

4 Experiments

In this section, we validate the effectiveness of MLE-STAR using 22 Kaggle competitions from MLE-bench Lite [16]. Our results demonstrate that MLE-STAR significantly outperforms baselines, including those employing various LLMs (Section 4.1). Furthermore, we show that using better models and leveraging our proposed ensemble strategy effectively improves performance (Section 4.2). We also provide the example solutions generated by MLE-STAR, in Appendix D.

Common setup. All experiments are conducted on 22 Kaggle competitions from MLE-bench Lite [16] using three random seeds, unless otherwise specified. Here, we use an agent $\mathcal{A}_{\texttt{test}}$, which takes the task description and the final solution as input, and outputs the code that incorporates loading test sample and creating a submission file (see Appendix E for details). MLE-STAR begins by retrieving four model candidates. MLE-STAR refines for four inner loops, while exploring four outer loops. For ensemble, MLE-STAR generates two solutions in parallel, and explore ensemble strategies for five rounds. Following the MLE-bench's setup, we set a maximum time limit of 24 hours for a fair comparison (see computation analysis in Appendix F). We primarily consider AIDE [12] as our main baseline, given its state-of-the-art performance on MLE-bench. It is important to note that other baselines often limit their generalizability across various task types (*e.g.*, audio classification, sequence-to-sequence), frequently showcasing results only on simpler modalities like tabular [11, 37]. For instance, DS-Agent [10] requires a manually constructed case bank, and their current GitHub repository lacks cases for audio classification, sequence-to-sequence, image classification, etc.

Table 1: **Main results from MLE-bench Lite.** Each experiment is repeated using three seeds, except for o1-preview (AIDE) and GPT-4o (AIDE), which use 16 and 36 seeds, respectively. All results are taken from the GitHub repository of MLE-bench paper [16], except for the model using Gemini-2.0-Flash and Gemini-2.5-Pro. Scores represent the mean and one standard error of the mean.

Model	Made Submission (%)	Valid Submission (%)	Above Median (%)	Bronze (%)	Silver (%)	Gold (%)	Any Medal (%)
MLE-STAR (Ours)							
gemini-2.5-pro	**100.0**$\pm$0.0	**100.0**$\pm$0.0	**83.3**$\pm$4.6	6.1$\pm$3.0	**21.2**$\pm$5.1	**36.4**$\pm$6.0	**63.6**$\pm$6.0
gemini-2.0-flash	95.5$\pm$2.6	95.5$\pm$2.6	63.6$\pm$6.0	**9.1**$\pm$3.6	4.5$\pm$2.6	30.3$\pm$5.7	43.9$\pm$6.2
AIDE [12]							
gemini-2.0-flash	87.9$\pm$4.0	78.8$\pm$5.0	39.4$\pm$6.0	4.5$\pm$2.6	9.1$\pm$3.5	12.1$\pm$4.0	25.8$\pm$5.4
o1-preview	99.7$\pm$0.3	90.3$\pm$1.6	58.2$\pm$2.6	4.8$\pm$1.1	11.1$\pm$1.7	20.7$\pm$2.2	36.6$\pm$2.6
gpt-4o	82.1$\pm$1.4	65.7$\pm$1.7	29.9$\pm$1.6	3.4$\pm$0.6	5.8$\pm$0.8	9.3$\pm$1.0	18.6$\pm$1.4
llama-3.1-405b-instruct	72.7$\pm$5.5	51.5$\pm$6.2	18.2$\pm$4.7	0.0$\pm$0.0	4.5$\pm$2.6	6.1$\pm$2.9	10.6$\pm$3.8
claude-3-5-sonnet	81.8$\pm$4.7	66.7$\pm$5.8	33.3$\pm$5.8	3.0$\pm$2.1	6.1$\pm$2.9	10.6$\pm$3.8	19.7$\pm$4.9
MLAB [38]							
gpt-4o	84.8$\pm$4.4	63.6$\pm$5.9	7.6$\pm$3.3	3.0$\pm$2.1	1.5$\pm$1.5	1.5$\pm$1.5	6.1$\pm$2.9
OpenHands [39]							
gpt-4o	81.8$\pm$4.7	71.2$\pm$5.6	16.7$\pm$4.6	3.0$\pm$2.1	3.0$\pm$2.1	6.1$\pm$2.9	12.1$\pm$4.0

Table 2: Comparison with DS-Agent.

Task	Metric	DS-Agent	**MLE-STAR**
WBY	MAE (↓)	213	**166**
MCC	RMLSE (↓)	0.2964	**0.2911**
ST	Accuracy (↑)	0.7982	**0.8091**
ES	AUROC (↑)	0.8727	**0.9101**

Table 3: Performance with Claude-Sonnet-4.

Task	Metric	2.0-Flash	**Sonnet-4**
DDD	RMSE (↓)	0.0681	**0.0155**
DBI	Log Loss (↓)	0.4535	**0.3114**
SAI	Log Loss (↓)	0.2797	**0.2610**
WCR	AUROC (↑)	**0.9903**	0.9888

4.1 Main results

Quantitative results. As demonstrated in Table 1, MLE-STAR significantly enhances the performance of various baseline models. For instance, when applied to Gemini-2.0-Flash, MLE-STAR improves AIDE's any medal achieving rates in Kaggle competitions from 25.8% to 43.9%, representing an improvement of over 18 percentage points, and rate of above median from 39.4% to 63.6%. Notably, MLE-STAR with Gemini-2.0-Flash also substantially outperforms AIDE using a powerful reasoning model (*i.e.*, o1-preview) in terms of achieving gold medals in 10% more tasks. Moreover, using Gemini-2.5-Pro, MLE-STAR shows a medal achievement of over 60%.

Comparison to DS-Agent. While DS-Agent [10] shows competitive results on ML tasks, it necessitates human effort to curate its case bank from Kaggle. Consequently, a direct comparison between DS-Agent and AIDE or our method is not feasible, as collecting tasks across diverse modalities, such as audio classification or image denoising, requires additional effort. Nevertheless, we utilize four tabular classification tasks, *i.e.*, wild-blueberry-yield (WBY), media-campaign-cost (MCC), spaceship-titanic (ST), and enzyme-substrate (ES), the same ones employed during DS-Agent's development stage [10], for a comparison. All experiments are done for 5 seeds following the original setup. As shown in Table 2, MLE-STAR significantly outperforms DS-Agent even without human efforts. See Appendix G for additional results, including comparison with AutoGluon [26].

4.2 Ablation studies

Performance with an advanced reasoning model. To assess if a more advanced reasoning model could enhance MLE-STAR's performance, we conduct an experiment with the recently released advanced reasoning models. First of all, as shown in Table 1, Gemini-2.5-Pro [43] yields better performance than using Gemini-2.0-Flash. For example, in denoising-dirty-documents competition, MLE-STAR with Gemini-2.0-Flash scored above the median across all three seeds, failing to achieve any medals. However, when using Gemini-2.5-Pro, MLE-STAR achieves two gold medals and one silver medal. These results demonstrate that MLE-STAR is designed to harness the advancements of rapidly improving reasoning-based LLMs.

Table 4: **Ablation on ensemble strategy.** Experiment results on MLE-bench Lite, repeated three seeds using Gemini-2.0-Flash. Scores represent the mean and one standard error of the mean.

Ensemble strategy	Made Submission (%)	Valid Submission (%)	Above Median (%)	Bronze (%)	Silver (%)	Gold (%)	Any Medal (%)
AIDE [12]							
None	87.9±4.0	78.8±5.0	39.4±6.0	4.5±2.6	9.1±3.5	12.1±4.0	25.8±5.4
MLE-STAR (Ours)							
None	**95.5**±2.6	**95.5**±2.6	57.6±6.1	7.6±3.3	4.5±2.6	25.8±5.4	37.9±6.0
Best-of-N	**95.5**±2.6	**95.5**±2.6	62.1±6.0	6.1±3.0	7.6±3.3	28.8±5.6	42.4±6.1
Average ensemble	**95.5**±2.6	**95.5**±2.6	60.6±6.1	6.1±3.0	**12.1**±4.0	25.8±9.4	**43.9**±6.2
Ours	**95.5**±2.6	**95.5**±2.6	**63.6**±6.0	**9.1**±3.6	4.5±2.6	**30.3**±5.7	**43.9**±6.2

Table 5: **Sensitivity analysis on the number of ensemble rounds.** Experiment results on 4 tasks from MLE-bench Lite, repeated three seeds using Gemini-2.0-Flash. We report the mean score.

Ensemble Round	DDD (RMSE; ↓)	DBI (Log Loss; ↓)	SAI (Log Loss; ↓)	WCR (AUROC; ↑)
1	0.07147	**0.45351**	0.28164	0.98943
3	**0.06805**	**0.45351**	**0.27967**	0.98898
5	**0.06805**	**0.45351**	**0.27967**	**0.99028**

Table 6: **Ablation on proposed components.** Experiment results on 4 tasks from MLE-bench Lite, repeated three seeds using Gemini-2.0-Flash. We report the mean score and bold the best one.

Targeted Refinement	Search Tool	DDD (RMSE; ↓)	DBI (Log Loss; ↓)	SAI (Log Loss; ↓)	WCR (AUROC; ↑)
✗	✓	0.10818	0.45689	0.29141	0.98532
✓	✗	0.09303	0.65242	0.30529	0.96509
✓	✓	**0.06805**	**0.45351**	**0.27967**	**0.99028**

In addition, we conduct additional experiments using Claude-Sonnet-4. Here, we select four different type of competitions: image-to-image (denoising-dirty-documents; DDD), image classification (dog-breed-identification; DBI), text classification (spooky-author-identification, SAI), and audio classification (the-icml-2013-whale-challenge-right-whale-redux; WCR). We run each competition for three seeds. As shown in Table 3, Claude-Sonnet-4 also shows promising results, indicating that our framework is also compatible and generalizable in terms of LLM type.

Effectiveness of proposed ensemble method. As highlighted in Table 4, MLE-STAR demonstrates a significant performance improvement over the competing baseline, *i.e.*, AIDE, achieving over a 12% higher rate of obtaining any medal *even without* additional ensemble strategy. Notably, by ensembling multiple solution candidates, our approach yields even greater performance gains, *i.e.*, MLE-STAR consistently improves the success rate for achieving any medal (and specifically gold medals), also surpassing the median human expert's performance by a larger margin compared to scenarios where this ensembling method is not used. While simpler strategies, such as selecting the solution with the best validation score or averaging final submissions, also offer benefits, MLE-STAR shows stronger effectiveness, *e.g.*, leading to a higher number of gold medals.

Furthermore, we conduct a sensitivity analysis on the number of ensemble rounds. Here, we utilize four datasets as same as Table 3. Table 5 indicates that while we utilize five rounds for ensemble strategy exploration, comparable performance can be achieved with fewer rounds.

Effectiveness of proposed components. Here, we focus on two key components of the proposed approach: *targeted refinement strategy* and *integrating search as a tool*. As shown in Table 6, we verify the effectiveness of our targeted refinement and the benefits of the search tool integration. Specifically, our findings indicate that refining only the specific code block identified by the ablation study agent is more effective than refining the entire codebase. Moreover, our experiments consistently show that utilizing search as a tool significantly enhances the overall performance.

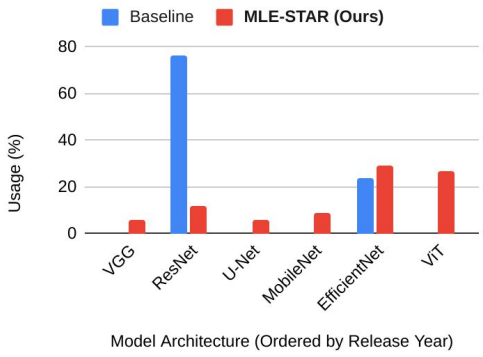

Figure 4: **Model usage** (%) on image classification competitions. Other models (11.7%), which are used by MLE-STAR, are omitted.

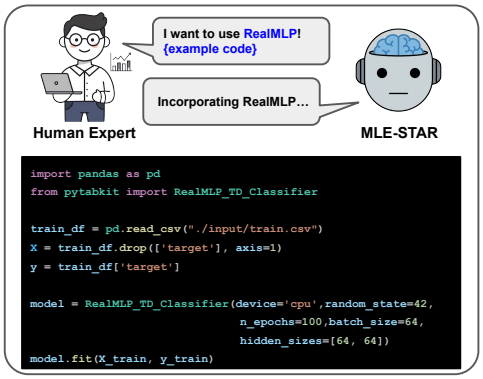

Figure 5: **Human intervention.** By manually adding a model description, MLE-STAR integrates its training into the framework.

Improperly imputed missing values

```
# Combined DataFrame for Consistent Preprocessing
all_df = pd.concat([train_df, test_df])
# Impute Numerical features with median
num_cols = ['Age', 'RoomService', 'FoodCourt']
for col in num_cols:
  if all_df[col].isnull().any():
      median_val = all_df[col].median()
      all_df[col] = all_df[col].fillna(median_val)
```

Python script after refined by data leakage checker

```
# Preprocess Training Set and get fit statistics
X_train_processed, stats =
      preprocess_data(X_train, fit_stats=None)
# Preprocess Test Set using statistics from Train Set
X_test_processed =
      preprocess_data(X_test, fit_stats=stats)
```

Figure 6: MLE-STAR's **data leakage checker** introduces appropriate preprocessing.

Python script before refined by data usage checker

```
data = pd.read_csv('./input/train.csv')
```

Python script after refined by data leakage checker

```
def process_xyz_files(df): # Features using XYZ files
  def volume(filepath):
      lines = open(filepath, 'r').readlines()
      a_vec = list(map(float, lines[2].split()[1:]))
      b_vec = list(map(float, lines[3].split()[1:]))
      c_vec = list(map(float, lines[4].split()[1:]))
      volume = np.dot(a_vec, np.cross(b_vec, c_vec))
      return abs(volume)
  for index, row in df.iterrows():
      file_path = os.path.join(row['id'],'geometry.xyz')
      df.loc[index, 'atomic_volume'] = volume(file_path)
  return df
data = process_xyz_files(data) # Process the train data
```

Figure 7: MLE-STAR's **data usage checker** captures previously unused information.

5 Discussion

Qualitative observations on selected models. Figure 4 illustrates the model usage of two MLE agents: AIDE and MLE-STAR. AIDE primarily employs ResNet [44] for image classification. However, ResNet, released in 2015, is now considered outdated and can result in suboptimal performance. In contrast, our MLE-STAR primarily utilizes more recent and competitive models like EfficientNet [45] or ViT [46], leading to the performance gain, winning 37% of the medals, more than AIDE, which wins 26% of the image classification challenges.

Human intervention. MLE-STAR readily adopts even more recent models with minimal human intervention. While MLE-STAR automatically constructs a model description $\{\mathcal{T}_{\texttt{model}}, \mathcal{T}_{\texttt{code}}\}$ using search as tool, a natural extension involves leveraging human expertise for this construction. As shown in Figure 5, by manually adding a model description for RealMLP [47], MLE-STAR successfully integrates its training into the framework, a model not previously retrieved. In addition, users can also specify the target code blocks by replacing the ablation summary with manually written instructions.

Misbehavior of LLMs and corrections. We observe that while the code generated by the LLM executed correctly, their content is sometime unrealistic, exhibiting hallucination. For example, Figure 6 illustrates an impractical approach where test data is preprocessed using its own statistics. Since test data must remain unseen, correction in the code is necessitated, for which, MLE-STAR employs a data leakage checker $\mathcal{A}_{\texttt{leakage}}$ to identify such issues in the generated Python script. If a problem is detected, MLE-STAR refines the code. As shown in the Figure, MLE-STAR successfully identifies the issue and modifies the code by, first extracting statistics from the training data and then preprocessing the test data using these calculated statistics. In addition, the improvement process

Table 7: Improvement failure when not using data leakage checker $\mathcal{A}_{\texttt{leakage}}$ on spaceship-titanic competition.

Metric	Accuracy (↑)
Validation	0.8188 → **0.8677**
Test	**0.8033** → 0.7343

Table 8: Ablation study of data usage checker $\mathcal{A}_{\texttt{data}}$ on nomad2018-predicting competition.

Model	$\mathcal{A}_{\texttt{data}}$	RMSLE (↓)
MLE-STAR	✗	0.0591
MLE-STAR	✓	**0.0559**

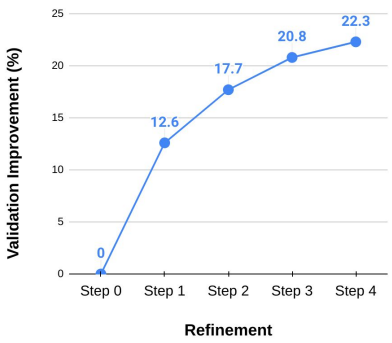

Figure 8: Solution refinement trajectory.

can fail to generalize when $\mathcal{A}_{\texttt{leakage}}$ is not employed, as exemplified in Table 7. In this example, the validation accuracy (*i.e.*, the target objective) improves, but the test accuracy drops significantly. This is attributed to the LLM performing feature engineering using the target variable `Transported`, which is not accessible in the test set, leading to data leakage and subsequently, poor test performance.

We also observe that LLMs often generate Python scripts that overlook some of the provided data sources. For example, in the nomad2018-predicting competition, Gemini-2.0-Flash solely loads train.csv, neglecting the use of geometry.xyz (see Figure 7). To address this, MLE-STAR employs $\mathcal{A}_{\texttt{data}}$, which reexamines the task description to ensure that all given data is utilized. As shown in Figure 7, this design enables MLE-STAR to incorporate previously neglected data. As a result, performance is significantly improved, as shown in Table 8.

Progressive improvement via MLE-STAR refinement. This section details the progressive improvement of solutions achieved by MLE-STAR, as measured by validation metrics. Given the task-specific nature of evaluation metrics, we report the average relative error reduction (%) across the all 22 challenges in MLE-bench Lite [16]. This metric measures the extent to which MLE-STAR reduces the error of an initial solution. Figure 8 demonstrates a consistent improvement as MLE-STAR proceeds through its refinement steps, which each step focusing on refining a single code block via an inner loop. Significantly, the magnitude of improvement is notable in the early refinement stages. We posit that this stems from MLE-STAR's ablation study module which helps to target the most influential code blocks for modification first.

Discussion on potential plagiarism. Following MLE-bench, we utilize Dolos [48], a source code plagiarism detection tool, to analyze generated solution code by MLE-STAR, against the top associated notebooks (*i.e.*, Jupyter notebook) from each Kaggle competition. Our analysis, summarized in Table 13 (see Appendix L), shows that no final solution code and notebook pair exceeded a 60% similarity score (*i.e.*, a criteria suggested from the MLE-bench paper), indicating no detected instances of plagiarism. This also shows that MLE-STAR's solution is sufficiently new compared to the existing solutions in Kaggle.

6 Conclusion

We propose MLE-STAR, a novel MLE agent designed for various ML tasks. Our key idea is to utilize a search engine to retrieve effective models and then explore various strategies targeting specific ML pipeline components to improve the solution. The effectiveness of MLE-STAR is validated by winning medals in 64% (where 36% are gold medals) of the MLE-bench Kaggle competitions.

Limitation. We acknowledge that MLE-STAR requires higher cost due to increased token usage. We include corresponding cost analysis in Appendix K. However, it is worth to note that still, with Gemini-2.0-Flash, the cost of MLE-STAR is only about $0.24 per each ML challenge. In addition, since Kaggle competitions are publicly accessible, there is a potential risk that LLMs might have been trained with the relevant discussions about the challenge. Nevertheless, we show that MLE-STAR's solution is sufficiently novel (using LLM as a judge) compared to the discussions on Kaggle (see Appendix H), and also show that its similarity compared to notebooks on Kaggle does not exceed 60%, alleviating such plagiarism issue (see Appendix L).

Acknowledgements and disclosure of funding

We would like to thank Raj Sinha, Subin Kim, Changyeon Kim, Dongjun Lee, Jihoon Tack, and anonymous reviewers for their helpful feedback and discussions. This work was partly supported by Institute for Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2019-II190075, Artificial Intelligence Graduate School Program (KAIST)) and ITRC (Information Technology Research Center) grant funded by the Korea government (MSIT) (IITP-2025-RS-2024-00436857, 50%).

References

[1] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[2] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 2018.

[3] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 2025.

[4] Linwei Fan, Fan Zhang, Hui Fan, and Caiming Zhang. Brief review of image denoising techniques. *Visual computing for industry, biomedicine, and art*, 2019.

[5] Noah Hollmann, Samuel Müller, and Frank Hutter. Large language models for automated data science: Introducing caafe for context-aware automated feature engineering. *Advances in Neural Information Processing Systems*, 2023.

[6] Jaehyun Nam, Kyuyoung Kim, Seunghyuk Oh, Jihoon Tack, Jaehyung Kim, and Jinwoo Shin. Optimized feature generation for tabular data via llms with decision tree reasoning. *Advances in Neural Information Processing Systems*, 2024.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 2020.

[8] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[9] Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

[10] Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. DS-agent: Automated data science by empowering large language models with case-based reasoning. *International Conference on Machine Learning*, 2024.

[11] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Ceyao Zhang, Chenxing Wei, Danyang Li, Jiaqi Chen, Jiayi Zhang, et al. Data interpreter: An llm agent for data science. *arXiv preprint arXiv:2402.18679*, 2024.

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

[13] Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? *International Conference on Learning Representations*, 2024.

[14] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. *International Conference on Learning Representations*, 2025.

[15] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 2011.

[16] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, et al. Mle-bench: Evaluating machine learning agents on machine learning engineering. *International Conference on Learning Representations*, 2025.

[17] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *International Conference on Learning Representations*, 2023.

[18] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 2023.

[19] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv: Arxiv-2305.16291*, 2023.

[20] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, 2022.

[21] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *Journal of Machine Learning Research*, 2022.

[22] Erin LeDell and Sebastien Poirier. H2O AutoML: Scalable automatic machine learning. *ICML Workshop on AutoML*, 2020.

[23] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[24] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 2017.

[25] Randal S Olson and Jason H Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. *ICML Workshop on AutoML*, 2016.

[26] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. *International Conference on Machine Learning*, 2018.

[28] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *International Conference on Learning Representations*, 2017.

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI Conference on Artificial Intelligence*, 2019.

[30] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 2019.

[31] Wei Fan, Erheng Zhong, Jing Peng, Olivier Verscheure, Kun Zhang, Jiangtao Ren, Rong Yan, and Qiang Yang. Generalized and heuristic-free feature construction for improved accuracy. *SIAM International Conference on Data Mining*, 2010.

[32] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. *IEEE International Conference on Data Science and Advanced Analytics*, 2015.

[33] Liyao Li, Haobo Wang, Liangyu Zha, Qingyi Huang, Sai Wu, Gang Chen, and Junbo Zhao. Learning a data-driven policy network for pre-training automated feature engineering. *International Conference on Learning Representations*, 2023.

[34] Franziska Horn, Robert Pack, and Michael Rieger. The autofeat python library for automated feature engineering and selection. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2019.

[35] Tianping Zhang, Zheyu Aqa Zhang, Zhiyuan Fan, Haoyan Luo, Fengyuan Liu, Qian Liu, Wei Cao, and Li Jian. Openfe: Automated feature generation with expert-level performance. *International Conference on Machine Learning*, 2023.

[36] Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using llm agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

[37] Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tuney Zheng, Minghao Liu, Xinyao Niu, Yue Wang, Jian Yang, Jiaheng Liu, et al. Autokaggle: A multi-agent framework for autonomous data science competitions. *arXiv preprint arXiv:2410.20424*, 2024.

[38] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. Mlagentbench: Evaluating language agents on machine learning experimentation. *International Conference on Machine Learning*, 2024.

[39] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. *International Conference on Learning Representations*, 2024.

[40] Janet L Kolodner. An introduction to case-based reasoning. *Artificial intelligence review*, 1992.

[41] Ian Watson and Farhi Marir. Case-based reasoning: A review. *The knowledge engineering review*, 1994.

[42] Yuki Ichihara, Yuu Jinnai, Tetsuro Morimura, Kenshi Abe, Kaito Ariu, Mitsuki Sakamoto, and Eiji Uchibe. Evaluation of best-of-n sampling strategies for language model alignment. *Transactions on Machine Learning Research*, 2025.

[43] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[44] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[45] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *International Conference on Machine Learning*, 2019.

[46] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *International Conference on Learning Representations*, 2021.

[47] David Holzmüller, Léo Grinsztajn, and Ingo Steinwart. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. *Advances in Neural Information Processing Systems*, 2024.

[48] Rien Maertens, Charlotte Van Petegem, Niko Strijbol, Toon Baeyens, Arne Carla Jacobs, Peter Dawyndt, and Bart Mesuere. Dolos: Language-agnostic plagiarism detection in source code. *Journal of Computer Assisted Learning*, 2022.

[49] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. Dsbench: How far are data science agents to becoming data science experts? *International Conference on Learning Representations*, 2025.

[50] Yiming Huang, Jianwen Luo, Yan Yu, Yitong Zhang, Fangyu Lei, Yifan Wei, Shizhu He, Lifu Huang, Xiao Liu, Jun Zhao, et al. Da-code: Agent data science code generation benchmark for large language models. *arXiv preprint arXiv:2410.07331*, 2024.

[51] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, et al. Infiagent-dabench: Evaluating agents on data analysis tasks. *arXiv preprint arXiv:2401.05507*, 2024.

[52] Ziming You, Yumiao Zhang, Dexuan Xu, Yiwei Lou, Yandong Yan, Wei Wang, Huaming Zhang, and Yu Huang. Datawiseagent: A notebook-centric llm agent framework for automated data science. *arXiv preprint arXiv:2503.07044*, 2025.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All claims in the introduction and abstract accurately reflect the contribution and scope.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss in Section 6.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: We do not have a theory in this paper.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We provide implementation details in Section 4. We also provide all prompts we use in Appendix A.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: The benchmark is already open-sourced, but we do not currently submit code when submitting.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the details in Section 4 and Appendix F.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: All experiments are conducted with multiple seeds, and we report one standard error of the mean.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We provide compute resources we used in Appendix F.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: We do not have any ethical concerns.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discuss the broader impacts in Appendix I.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our framework does not introduce risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have cited all papers and datasets in Reference.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We do not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: We do not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: We do not have human subject.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: The core method development does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.