Controllable Automation of ML Research Engineering via Structured Specifications

Anonymous ACL submission

Abstract

Large language model agents are increasingly being used to automate scientific workflows such as code development, experiment execution, and research reproduction. However, existing systems for automating machine learning (ML) research engineering are still limited by the lack of an explicit intermediate representation that connects research artifacts to executable implementations, leading to cascading failures and costly end-to-end repair. Therefore, we propose a Controllable Long-horizon Engineering system for Automating ML Research workflows (CLEAR). The key idea is to introduce a specification layer that converts heterogeneous research artifacts into structured, verifiable execution specifications prior to code generation. This design enables complex workflows to be decomposed into specification-conditioned local generation and repair, rather than relying on tightly coupled end-to-end regeneration. CLEAR further incorporates a large–small model routing mechanism for efficient task allocation, along with a lifecycle-oriented execution harness that provides runtime control, observability, and cost-aware intervention. Experimental results show that CLEAR achieves strong performance–efficiency trade-offs. The code is available at https://anonymous.4open.science/r/CLEAR-VSS.

1 Introduction

Recent advances in large language model (LLM) agents have enabled systems that automate complex scientific workflows, including idea generation(Lu et al., 2025), experiment execution(Huang et al., 2024), code development(Zhang et al., 2025), and scientific writing(Wang et al., 2024; Lu et al., 2026). These developments position autonomous scientific research as a realistic goal for general-purpose agent systems(Schmidgall et al., 2025; Xu et al., 2026; Tang et al., 2026).

Existing efforts broadly span three directions: (i) *scientific exploration*, which focuses on idea generation, hypothesis discovery, and refinement of existing research; (ii) *research engineering*, which aims to build executable ML systems and reproduce research artifacts; and (iii) *scientific writing*, which assists with paper drafting and presentation. Among them, automating ML research engineering is particularly challenging, as it requires transforming heterogeneous research artifacts into executable systems under strict cost and runtime constraints (Chan et al., 2025; Zhu et al., 2026; Chen et al., 2026). Unlike isolated tasks such as code generation or paper summarization, it is fundamentally a long-horizon execution problem that couples interpretation, implementation, and iterative debugging within a single workflow.

Recent systems such as *BasicAgent* (Starace et al., 2025), *Leeroo* (Mohammadshahi et al., 2024), *AIDE* (Jiang et al., 2025) and *AIBuildAI* (Zhang et al., 2026) demonstrate the feasibility of this paradigm, but they still suffer from a fundamental limitation: directly operating on heterogeneous research artifacts without an explicit intermediate representation causes cascading errors and expensive repair loops. We identify such failure mode arises from three interconnected factors: (i) *Ambiguous interpretation of heterogeneous inputs.* Research artifacts contain mixed modalities such as text, equations, tables, pseudo-code, and evaluation protocols, leading to underspecified implementation assumptions when directly processed by generation models. (ii) *Propagation of early-stage errors.* Because interpretation and execution are tightly coupled, errors in early planning or understanding propagate through subsequent coding and debugging stages, amplifying downstream failure. (iii) *Inefficient repair coupling.* Existing systems rely on repeated end-to-end regeneration in write-run-debug loops, which leads to redundant computation, high token cost, and unstable convergence.

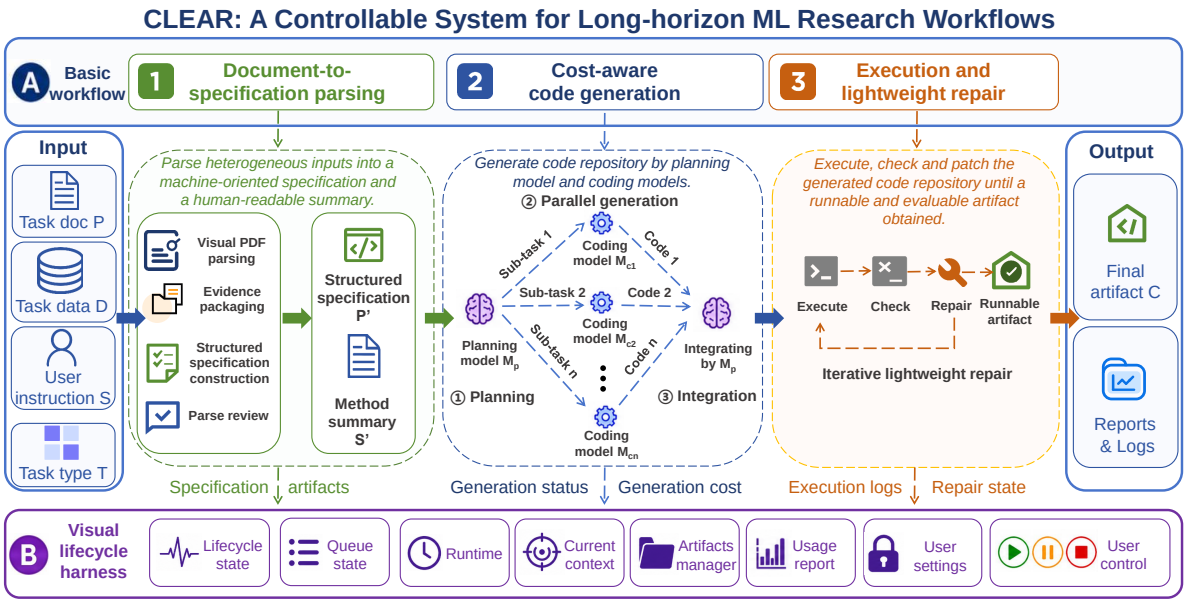

Figure 1: Architecture of CLEAR, a controllable system for long-horizon ML research workflows.

To address this, we propose CLEAR, a controllable system for long-horizon ML research workflow built on a simple principle: an explicit intermediate representation is necessary to separate research understanding from executable realization. CLEAR introduces a structured specification layer that transforms heterogeneous research artifacts into verifiable execution specifications, that serve as the foundation for downstream generation and repair rather than raw inputs.

As shown in Figure 1, CLEAR consists of three components: (i) *Document-to-specification parsing*, which transforms heterogeneous research artifacts into structured executable specifications with implementation details; (ii) *Large-small model collaboration*, where large models perform global planning and integration over specifications, while small models are assigned to localized code generation and iterative repair, enabling efficient parallelized execution. (iii) *Lifecycle-oriented orchestration*, implemented as an execution harness that manages the full parsing-execution-repair loop, enabling fine-grained control, runtime observability, cost tracking, and targeted intervention. Together, these components enable execution-time control over both model behavior and workflow states, supporting both controllability and cost efficiency in long-horizon ML research workflows.

Experiments on PaperBench and MLE-Bench Lite demonstrate that CLEAR achieves a strong balance between performance and computational efficiency. On PaperBench, CLEAR achieved a score of 29.87 with an average cost of \$7.91 per paper, a cost reduction of 35.16% compared to the latest work *AiScientist* (Chen et al., 2026), while performance only decreased by 3.86 points. On MLE-Bench Lite, CLEAR achieves 77.27% Any Medal, only 4.55 points below *AiScientist*, while CLEAR requires 27.08% fewer repair iterations than *AiScientist* to achieve the same performance on *Detecting Insults* task.

2 Task Formulation

We study ML research engineering as the problem of transforming heterogeneous research specifications into executable and evaluable computational artifacts. Formally, given a task type T, a task document P, optional task data D and user instruction S, an agent system $\mathcal{A}$ produces an artifact C:

$$C = \mathcal{A}(T, P, D, S). \tag{1}$$

The generated artifact C may include runnable code repositories, experiment pipelines, trained models, evaluation results, or valid competition submissions, depending on the type of task. A valid artifact must satisfy two requirements: (i) *executability*, meaning that the artifact can be successfully executed or submitted under the task environment; and (ii) *task performance*, measured by a task-specific evaluation function. We therefore formulate ML research engineering as the following constrained optimization problem:

$$\begin{aligned} C^\star &= \arg\max_{C \in \mathcal{C}_T} Q_T(C), \\ \text{s.t.} \quad & \mathrm{Exec}_T(C) = 1, \end{aligned} \tag{2}$$

where $\mathcal{C}_T$ denotes the artifact space for task type T, $Q_T(\cdot)$ is the task-specific evaluation metric, and $\mathrm{Exec}_T(C)$ indicates whether C can be executed or submitted successfully.

In this work, we consider two representative task types: (i) *Replicate AI Research (RAR)*, where the input is an AI paper P_r and the output is a runnable reproduction artifact $C_r = \mathcal{A}(\mathrm{RAR}, P_r, \varnothing, S_r)$; (ii) *Machine Learning Engineering (MLE)*, where the input consists of a task description P_m and task data D_m, and the output is a runnable ML pipeline or a valid competition submission $C_m = \mathcal{A}(\mathrm{MLE}, P_m, D_m, S_m)$.

Despite differences in inputs and evaluation protocols, both task types require translating heterogeneous specifications into long-horizon executable workflows. Consequently, both suffer from similar failure modes, including ambiguous specification grounding, cascading error propagation, and tightly coupled repair processes. These shared challenges motivate the structured and controllable design of our proposed CLEAR.

3 Our Method

CLEAR follows a specification-first and cost-aware design principle. Instead of directly asking LLM agents to generate and repair code from raw research artifacts, CLEAR first makes task understanding explicit through structured specifications. The subsequent generation and repair stages are then conditioned on these specifications, allowing the system to decompose long-horizon research engineering into more controllable local decisions. As shown in Figure 1, CLEAR follows a three-stage workflow: (i) *document-to-specification parsing*, where the system receives the task type T, task document P, optional task data D, and optional user instruction S, and parses these heterogeneous inputs into a machine-oriented structured specification P' and a human-readable method summary S'; (ii) *cost-aware code generation*, where a strong planning model decomposes P' and S' into localized subtasks and coordinates multiple lower-cost coding models to generate the initial repository C_0 in parallel; (iii) *execution and lightweight repair*, where the generated repository C_0 is executed, checked, and patched until a runnable and evaluable artifact C is obtained.

The whole workflow is managed by a visual lifecycle harness. Instead of treating LLM agents as a single black-box generator, the harness controls

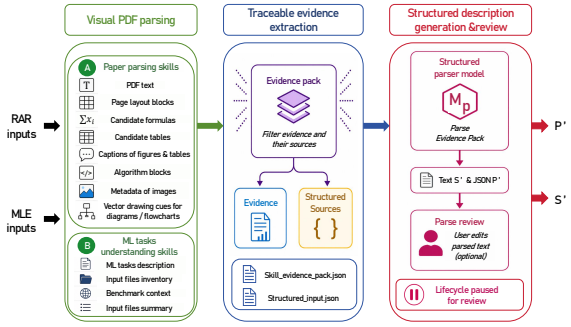

Figure 2: Skills design of CLEAR. Visual PDF parsing fully exploits various heterogeneous information in task input, traceable evidence extraction ensures interpretability of task execution, structured description generation and optional manual description review enhance the reliability of code generation requirements.

when agents parse, plan, generate, execute, and repair. It also exposes intermediate artifacts, supports user review and task control, and tracks running time, token usage, and estimated cost.

3.1 Document-to-Specification Parsing

The goal of this stage is to convert heterogeneous task inputs into explicit specification artifacts before code generation. Given task type T, task document P, optional task data D, and user instruction S, CLEAR parses the input into two complementary outputs: a machine-oriented structured specification P' and a human-readable method summary S'. The structured specification P' records information relevant to implementation, such as algorithms, inputs and outputs, dependencies, parameters, evaluation metrics, and submission requirements. The method summary S' exposes the parsed understanding to users for inspection and optional correction.

We formalize this stage as follows:

$$(S', P') = g(T, P, D, S), \tag{3}$$

where g denotes the document-to-specification parsing process. Here, P' is used by the downstream code generation models, while S' is shown to users for review. When user review is enabled, users may review S' before it is passed to the generation stage.

As shown in Figure 2, this stage consists of four parts: (i) *visual PDF parsing*, which extracts text, page layout, formulas, tables, figures, captions, and algorithm blocks from the input document; (ii) *evidence packaging*, which selects implementation-relevant elements and preserves their sources to form traceable evidence packs; (iii) *structured specification construction*, where M_p organizes the

evidence into the structured specification P' and the readable summary S'; (iv) *parse review*, where users can inspect and revise S' before expensive code generation begins. By turning raw research artifacts into explicit and traceable specifications, this stage reduces ambiguous implementation assumptions and prevents early parsing errors from silently propagating to code generation.

3.2 Cost-aware Code Generation

The cost-aware code generation stage receives the structured specification P' and the method summary S', and generates the initial repository C_0 through the collaboration of large-small models. The key intuition is that ML research engineering requires strong global reasoning for algorithm understanding, architecture planning, and project-level integration, while many localized implementation subtasks can be executed by lower-cost coding models once the specification is clear. Therefore, CLEAR routes global planning and integration to the stronger model M_p, and dispatches localized coding subtasks to multiple lower-cost workers M_c.

This stage consists of three steps: (i) *Planning.* The planning model M_p reads P' and S', analyzes the implementation requirements, and decomposes the project into a set of localized subtasks $I_{1:n}$, where n is the maximum number of parallel coding workers. (ii) *Parallel generation.* Each lower-cost coding worker M_c receives one subtask I_j and generates the corresponding code files under specified relative paths. (iii) *Integration.* The planning model M_p collects the generated files, checks dependencies and conflicts, and merges them into the initial code repository C_0.

We denote this stage as:

$$C_0 = G(P', S'; M_p, M_c, n), \tag{4}$$

where G denotes the collaborative generation process, in which M_p performs planning and integration and up to n lower-cost workers M_c perform localized code generation.

By separating global planning from localized implementation, CLEAR preserves the benefits of a strong model for complex task understanding and repository integration, while avoiding excessive large-model calls during code generation. This design improves generation efficiency and reduces model cost compared with using the same large model for all subtasks.

3.3 Execution and Lightweight Repair

In this stage, CLEAR verifies the generated repository and applies lightweight repair based on execution feedback: (i) It runs and tests the initial code repository C_0 in the internal runtime environment. When execution errors occur, the coding model M_c receives run reports and generates targeted patches, avoiding repeated full-repository regeneration. (ii) For MLE tasks, it further tests C_0 on the task data D and performs specialized Kaggle submission checks. (iii) It finally produces a submit-ready code repository C according to the task type T. We formalize this stage as follows:

$$C = e(T, D, C_0), \tag{5}$$

where e denotes the execution and lightweight repair process of CLEAR.

3.4 Lifecycle-Oriented Orchestration

For a long-running AI/ML research engineering system, runtime monitoring and user control are essential. CLEAR may pass through multiple stages such as document parsing, user review, code generation, execution, and repair, where failures can occur at different points. Therefore, CLEAR implements lifecycle management through a visual harness built around a persistent priority queue, enabling interruptible and recoverable control, human-in-the-loop checkpoints, and observable execution.

Interruptible and recoverable control. CLEAR uses a persistent priority queue to schedule submitted tasks and manage concurrent subtasks. Since task states, input files, and intermediate artifacts are persistently stored, CLEAR can recover from local service interruptions and resume paused jobs after restart. Users can pause, stop, or resume a running task from the frontend, and the harness applies these actions at safe lifecycle boundaries. It also provides real-time feedback for failures, timeouts, suspensions, interruptions, and recovery states.

Human-in-the-loop checkpoints. After the parsing stage generates the method description S', CLEAR automatically pauses at the parsing result. This checkpoint allows users to inspect and revise the parsed task understanding before expensive code generation begins, correcting missing assumptions, parameters, or evaluation details before they propagate to downstream stages. After editing or confirmation, users can click the "Continue"

button to resume the workflow, and the updated S' will be passed to the downstream generation stage.

Observable execution. CLEAR divides each task into 15 operational stages and reports the current stage to the frontend in real time. During execution, the harness collects stage-level operational data, including intermediate artifacts, execution logs, and repair states. The "Queue" panel displays running, queued, and paused jobs, together with their priorities, queue order, current stages, and task IDs. After completion, the harness summarizes runtime statistics, including running time, token usage, and estimated model cost.

4 Experiment Settings

Datasets. We consider two ML research engineering tasks, RAR and MLE (Section 2), and evaluate CLEAR using one dataset per task.

- For RAR tasks, PaperBench (Starace et al., 2025) evaluates whether agents can understand 20 top-tier ML papers from scratch, implement their codebases, and successfully run experiments.
- For MLE tasks, MLE-Bench (Chan et al., 2025) evaluates agents' ability to iteratively improve performance across 75 ML competitions. We use its Lite subset (22 tasks) as a lower-cost alternative to full evaluation.

The above two complementary datasets effectively evaluate the efficiency and cost of agent execution and optimization in ML research engineering during long-duration complex tasks.

Evaluation metrics. For PaperBench, its task score is automatically evaluated based on a large model, which is the weighted mean of 8,316 leaf scoring nodes. For MLE-Bench Lite, we evaluated the dataset on 22 tasks, calculating the percentage of tasks that resulted in valid submissions out of the total number of tasks, the percentage of tasks that scored above the median, the percentage of tasks achieving bronze, silver and gold medals, and the percentage of tasks obtaining any medal.

Baselines. We compare CLEAR against the following baselines on each dataset:

- On PaperBench, we compared our method with three different agent methods, namely AiScientist (Chen et al., 2026), BasicAgent, and IterativeAgent (Starace et al., 2025), under the same evaluation protocol, despite their distinct processes and structures.
- On MLE-Bench Lite, we report comparative results against strong autonomous ML engineering systems, including methods such as AIDE (Jiang et al., 2025), ML Master 2.0 (Zhu et al., 2026), and LoongFlow (Wan et al., 2025).

Implementation details. Following the settings in AiScientist, we use Gemini-3-Flash and GLM-5 (Zeng et al., 2026) as the two backbone LLMs to instantiate the aforementioned baseline methods.

For our own method, CLEAR, to better meet the working needs of our designed large-small model collaboration, we have chosen Claude-Opus 4.6, which possesses strong thinking capabilities, as the planning model M_p, while Claude-Sonnet 4.6, which has similar habits but is more cost-effective, has been selected as the coding model M_c.

In PaperBench and MLE-Bench Lite, each run is allocated an Nvidia A800 80GB GPU and a 24-hour budget for each task. For the comprehensive evaluation of PaperBench, we adopted the official evaluation protocol (Starace et al., 2025) and used GPT-5.4 as judging model.

5 Experimental Results

We evaluate CLEAR by addressing three research questions: (i) **RQ1: Effectiveness.** Can CLEAR successfully perform ML research engineering tasks across both RAR and MLE settings? (ii) **RQ2: Efficiency.** Does CLEAR achieve a better performance–cost trade-off than existing long-horizon ML research agents? (iii) **RQ3: Mechanism.** How do its key design choices, particularly structured specifications and large–small model collaboration, contribute to performance, execution stability, and cost reduction? We next present **RQ1** and **RQ2** in Sections 5.1 and 5.2, and **RQ3** in Section 5.3.

5.1 Main Results on PaperBench

Table 1 reports the main results of PaperBench. Compared with the baseline method, CLEAR ranked second in 10 of 20 tasks and achieved an average score of 29.87 out of 20 tasks, with an average execution cost of only \$7.91 per task. The above results fully demonstrate the effectiveness of CLEAR method in executing ML research engineering tasks under RAR setting.

In terms of performance–cost trade-off, we make the following comparisons: (i) Compared with the simpler BasicAgent baseline, CLEAR improves performance by 32.29%, at the cost of an additional \$3.01 per task on average. This shows that the structured workflow introduces useful extra computation

Task Name	Gemini			GLM			Opus & Sonnet
	BasicAgent	IterAgent	AiScientist	BasicAgent	IterAgent	AiScientist	CLEAR (Ours)
adaptive-pruning	24.53	3.05	27.25	30.82	11.93	**33.26**	21.17
all-in-one	20.86	45.13	46.29	33.78	44.43	**49.47**	48.20
bam	48.46	45.04	56.59	51.45	47.91	**61.11**	52.33
bbox	15.43	8.30	**33.79**	23.55	19.28	30.02	28.23
bridging-data-gaps	12.59	12.44	23.09	9.80	12.50	**26.46**	26.16
fre	21.67	23.89	**35.21**	21.60	16.67	28.98	28.71
ftrl	5.87	4.15	**10.11**	3.71	6.70	8.34	8.82
lbcs	17.75	15.26	27.90	20.68	22.74	**30.10**	26.24
lca-on-the-line	12.97	18.30	**30.23**	22.55	26.15	28.53	24.64
mechanistic-understanding	14.86	21.89	29.95	32.49	34.96	**40.55**	35.69
pinn	26.63	30.81	49.92	22.18	25.77	**58.76**	49.97
rice	10.43	8.88	**10.87**	6.56	0.27	10.18	10.58
robust-clip	15.45	10.43	18.28	22.43	27.56	**28.66**	19.82
sample-specific-masks	25.39	33.34	36.77	36.93	41.26	**44.13**	37.35
sapg	11.45	12.65	19.85	6.99	4.95	**31.69**	24.19
sequential-neural	53.51	60.24	**64.94**	27.2	35.53	49.32	49.85
stay-on-topic	8.37	13.69	**20.13**	3.69	8.81	14.81	16.26
stochastic-interpolants	17.04	17.37	18.81	32.18	28.06	**42.10**	39.77
test-time-model-adaptation	15.27	18.13	**32.45**	17.81	21.19	27.33	28.46
what-will-my-model-forget	6.61	8.99	17.87	25.14	10.75	**30.82**	20.98
Average Score	19.26	20.60	30.52	22.58	22.37	**33.73**	29.87
Average Cost	$6.25	$27.44	$15.67	**$4.90**	$54.90	$12.20	$7.91

Table 1: Main results on PaperBench full evaluation. Bold and underlined denote best and second-best results.

for the RAR task, leading to a clear effectiveness gain over a lightweight baseline. (ii) Compared with IterAgent, which has iterative interaction capability but cannot think systematically for a long time, CLEAR achieved higher scores with an average cost of only 14.4% using IterAgent equipped with GLM-5. This suggests that performance gains do not come from more iterations alone, but from whether the system can organize these iterations through globally aware planning, integration, and targeted repair. (iii) Compared with the AiScientist method that uses the same model to complete different levels of work, CLEAR achieved a cost advantage of 35.16% at a sacrifice of only 11.44% of performance, indicating that CLEAR's collaborative code generation process for large and small models can effectively save costs. In summary, these results show that CLEAR achieves the intended performance–cost balance: it maintains competitive performance while reducing generation cost through specification-guided generation and lightweight repair.

5.2 Main Results on MLE-Bench Lite

Table 2 reports the main results of MLE-Bench Lite. Compared with the baseline method, CLEAR achieved the optimal level in Valid Submission, ranking second in both Above Median and Any Medal. All 22 tasks formed valid submissions, with 86.36% of tasks achieving scores above the median and 77.27% of tasks winning medals. The

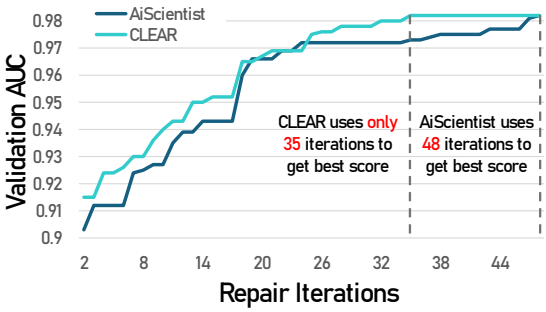

Figure 3: CLEAR autonomously improving performance on a competition-style ML task over iterations. On MLE-Bench Lite's Detecting Insults task, it completes 35 experiment cycles, reaching a validation AUC of 0.982 with 17 best-so-far updates. Compared to AiScientist, CLEAR reduces iterations by 27.08%.

above results fully demonstrate the effectiveness of CLEAR method in executing ML research engineering tasks under MLE setting.

In terms of performance–cost trade-off, Figure 3 compares the validation AUC trajectories of CLEAR and AiScientist on the Detecting Insults task from MLE-Bench Lite. Without human intervention, CLEAR reaches a validation AUC of 0.982 after 35 repair iterations, with 17 best-score updates. In comparison, AiScientist requires 48 repair iterations to reach the same level, meaning that CLEAR reduces the number of repair iterations by 27.08%. Specifically, the figure shows two main trends: (i) CLEAR reaches the same validation AUC earlier in most stages, suggest-

Method	Model	Valid	Above Median	Bronze	Silver	Gold	Any Medal
AIDE	Gemini	77.27	54.55	4.55	9.09	31.82	45.45
LoongFlow	Gemini	77.27	77.27	12.12	25.76	39.39	77.27
AiScientist	Gemini	100.00	86.36	**18.18**	31.82	31.82	**81.82**
AIDE	GLM	77.27	50.00	4.55	13.64	22.73	40.91
ML-Master 2.0	GLM	100.00	81.82	**18.18**	13.64	31.82	63.64
AiScientist	GLM	100.00	**90.91**	9.09	31.82	**40.91**	**81.82**
CLEAR (Ours)	Opus & Sonnet	100.00	86.36	13.64	**36.37**	27.28	77.27

Table 2: Main results (%) on MLE-Bench Lite. Bold and underlined denote best and second-best results.

Method	Model	Avg Score	Avg Cost
CLEAR	Opus & Sonnet	29.87	$7.91
CLEAR w/o specs	Opus & Sonnet	24.14	$10.08
CLEAR	GLM	26.41	$11.95

Table 3: Ablation results on PaperBench full evaluation. "CLEAR w/o specs" means CLEAR only uses M_p to parse input documents, without processing by our designed skills. Bold and underlined denote the best and second-best results on each task.

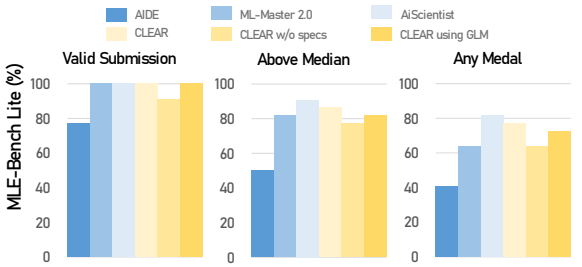

Figure 4: Ablation results on MLE-Bench Lite. All values are percentages (%). "CLEAR w/o specs" means CLEAR only uses M_p to parse input documents, without processing by our designed skills. "CLEAR using GLM" means CLEAR uses GLM-5 for M_p and M_c at the same time.

ing that the structured specification layer provides more effective guidance for downstream generation and repair; (ii) AiScientist tends to make larger but sparser improvements, while CLEAR makes smaller and more frequent updates. This suggests that CLEAR's lightweight repair process enables more fine-grained engineering optimization with fewer repair iterations.

Together with the PaperBench results, these findings answer **RQ1** and **RQ2**: CLEAR is effective across both RAR and MLE settings, and it achieves a favorable performance–cost trade-off by maintaining competitive task performance while reducing generation cost or repair iterations.

5.3 Mechanism Analysis

To answer **RQ3**, we study which components drive CLEAR's performance and efficiency. Table 3 reports ablation results on PaperBench, and Figure 4 shows corresponding results on MLE-Bench Lite. We focus on two key designs: *structured specifications* and *large–small model collaboration*.

Structured specifications. We first study the role of structured specifications using *CLEAR w/o specs*, which removes the designed document-to-specification parsing skills and relies only on the planning model M_p to parse input documents. As shown in Table 3 and Figure 4, removing structured specifications consistently hurts both performance and efficiency: (i) On PaperBench, compared with CLEAR, the average score of CLEAR w/o specs decreases by 19.18%, while the average cost increases by 27.43%. (ii) On MLE-Bench Lite, CLEAR w/o specs decreases by 9.09%, 10.53%, and 17.64% on Valid Submission, Above Median, and Any Medal, respectively. (iii) These results indicate that without structured specifications, the system not only loses accuracy in generation and verification, but also spends more model cost on document understanding and downstream repair.

These results support our central motivation: making task understanding explicit before code generation reduces ambiguous implementation assumptions and prevents early errors from propagating through later stages.

Large-small model collaboration. We then study the role of large-small model collaboration using *CLEAR using GLM*, where both the planning model M_p and the coding model M_c are replaced with GLM-5. Since this variant changes both the model assignment and the model backbone, we use it to analyze model-routing sensitivity rather than to isolate the effect of model collaboration alone. Table 3 and Figure 4 show that using a unified model for all stages weakens both performance and efficiency: (i) On PaperBench, compared with CLEAR, the average score of CLEAR using GLM decreases by 11.35%, while the average cost increases by 51.07%. (ii) On MLE-Bench Lite, CLEAR using GLM shows no change in Valid Submission, but decreases by 5.26% and 5.88% on Above Median and Any Medal, respectively. (iii) These results

suggest that a single model is not necessarily cost–effective for all stages. Stronger models are more useful for global planning and integration, while lower-cost coding models can handle localized implementation and repair more efficiently.

This validates CLEAR's design of separating global planning from localized implementation, using stronger models only where high-level reasoning is required while avoiding unnecessary large-model calls during routine coding and repair.

Summary. Overall, the ablation results in Table 3 and Figure 4 answer **RQ3**: structured specifications and large-small model collaboration both provide important support for CLEAR's effectiveness and performance–cost balance. Structured specifications improve the reliability of task understanding, while model collaboration helps reduce unnecessary generation cost without sacrificing competitive task performance.

6 Related Work

We review two lines of related work that underpin our study: LLM-based autonomous scientific research systems and multi-agent cooperation workflows for complex task execution.

Automated scientific research. Recent work has explored the use of LLM agents for automating scientific research, including idea generation(Lu et al., 2025), experiment execution(Huang et al., 2024), code development(Zhang et al., 2025; Li et al., 2025; Seo et al., 2026), and scientific writing(Wang et al., 2024; Lu et al., 2026). These developments position autonomous scientific research as a realistic goal for general-purpose agent systems(Schmidgall et al., 2025; Xu et al., 2026; Tang et al., 2026).

Among them, automating ML research engineering is particularly representative and challenging, as it requires transforming heterogeneous research artifacts into executable systems under limited cost and runtime constraints (Chan et al., 2025; Zhu et al., 2026; Chen et al., 2026). Recent systems such as BasicAgent (Starace et al., 2025), Leeroo (Mohammadshahi et al., 2024), AIDE (Jiang et al., 2025), AIBuildAI (Zhang et al., 2026), and AiScientist (Chen et al., 2026) demonstrate the feasibility of this paradigm. However, these systems often operate directly on heterogeneous research artifacts without an explicit intermediate representation, which can lead to cascading errors and expensive repair loops.

Multi-agent cooperation workflows. Multi-agent cooperation has become a common paradigm for improving LLM agents on complex tasks. Early frameworks explore role-playing agents, dynamic team formation, and programmable multi-agent conversations (Li et al., 2023; Chen et al., 2024b; Wu et al., 2023). These works show that task decomposition and agent specialization can improve problem solving, but coordination is often mediated through flexible natural-language interaction and heavily relies on human-designed prompts. In software engineering, multi-agent workflows have been adapted to code generation, repository editing, and issue resolution. MetaGPT and ChatDev organize agents through standardized software-development pipelines (Hong et al., 2024; Qian et al., 2024). SWE-agent emphasizes agent-computer interfaces for repository-level editing and testing (Yang et al., 2024), while AutoCodeRover and CodeR further incorporate code search, program analysis, and task graphs for issue resolution (Zhang et al., 2024; Chen et al., 2024a).

Despite recent advances, multi-agent systems still suffer from coordination failures, inter-agent misalignment, communication overhead, and weak verification (Cemri et al., 2025; Yan et al., 2025). These issues become more severe in long-horizon ML research engineering, where task understanding, execution feedback, and repair decisions must remain consistent across stages. This motivates introducing a more explicit intermediate representation between research understanding and executable implementation.

7 Conclusion

We propose CLEAR, a controllable framework for automating long-horizon ML research engineering. CLEAR introduces a structured specification layer between heterogeneous research artifacts and executable implementations, enabling downstream generation and repair to be guided by explicit and verifiable requirements. Combined with large–small model collaboration and lifecycle-oriented orchestration, CLEAR balances task performance, execution stability, and generation cost. Experiments on PaperBench and MLE-Bench Lite show that CLEAR achieves competitive performance with favorable performance–cost trade-offs across both RAR and MLE settings. Further analysis demonstrates that its key designs improve cost efficiency while maintaining strong task performance.

Limitations

While CLEAR shows promising results on both RAR and MLE tasks, there are still several limitations that require further investigation. (i) Due to the high cost of long-horizon ML research engineering experiments, our evaluation is mainly conducted on two representative benchmarks, PaperBench and MLE-Bench Lite. Although they cover both paper reproduction and ML competition settings, they may not fully reflect more fine-grained or more difficult research engineering scenarios. Future work will evaluate CLEAR on broader and more challenging benchmarks to better assess its scalability and generality. (ii) For RAR tasks, we summarize performance using averaged scores across subtasks, which may hide fine–grained differences among task types. For MLE tasks, metrics such as valid submission rate and medal rate capture overall task success, but may not fully reveal the quality of generated code, repair behavior, or engineering decisions. More detailed evaluation protocols are needed to better characterize long-horizon ML research engineering systems. (iii) CLEAR combines several mechanisms, including structured specification construction, large-small model collaboration, lifecycle-oriented orchestration, and lightweight repair. While our experiments show their usefulness, their stability under different model backbones, task difficulties, and execution environments has not been fully studied. A more systematic robustness analysis remains future work. (iv) CLEAR is motivated by the assumption that structured specifications can reduce input ambiguity and guide downstream generation. However, our current analysis does not fully explain when, why, and how such specifications help planning models. Future work will further analyze the content of generated specifications and study their impact on code generation, repair, and execution stability.

Ethical Considerations

This article is aimed at the problem that in automatic ML research engineering, directly operating on heterogeneous research artifacts without an explicit intermediate representation causes cascading errors and expensive repair loops. To mitigate this issue, we propose CLEAR, a controllable system for long-horizon ML research workflow in which a structured specification layer transforms heterogeneous research artifacts into verifiable execution specifications. With respect to data usage, all experiments are conducted exclusively on publicly available academic benchmarks, including PaperBench and MLE-Bench Lite. No private, proprietary, or sensitive data is used at any stage of data collection, training, or evaluation. Furthermore, the proposed method does not introduce or expose any new personal information. As a result, this study does not raise concerns related to privacy, data misuse, or unauthorized access, and complies with common ethical standards for academic research in information retrieval and natural language processing.

References

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent llm systems fail? *Preprint*, arXiv:2503.13657.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, and 1 others. 2025. Mle-bench: Evaluating machine learning agents on machine learning engineering. In *International Conference on Learning Representations*, volume 2025, pages 50466–50494.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. 2024a. Coder: Issue resolving with multi-agent and task graphs. *Preprint*, arXiv:2406.01304.

Guoxin Chen, Jie Chen, Lei Chen, Jiale Zhao, Fanzhe Meng, Wayne Xin Zhao, Ruihua Song, Cheng Chen, Ji-Rong Wen, and Kai Jia. 2026. Toward autonomous long-horizon engineering for ml research. *arXiv preprint arXiv:2604.13018*.

Weize Chen, Yusheng Su, Jingwei Zuo, Cheng Yang, Chenfei Yuan, Chi-Min Chan, Heyang Yu, Yaxi Lu, Yi-Hsin Hung, Chen Qian, Yujia Qin, Xin Cong, Ruobing Xie, Zhiyuan Liu, Maosong Sun, and Jie Zhou. 2024b. AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pages 20094–20136.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *International Conference on Learning Representations*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. Mlagentbench: Evaluating language agents on machine learning experimentation. In *International Conference on Machine Learning*, pages 20271–20309. PMLR.

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. *arXiv preprint arXiv:2502.13138*.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. CAMEL: Communicative agents for "Mind" exploration of large language model society. In *Advances in Neural Information Processing Systems*, volume 36, pages 51991–52008. Curran Associates, Inc.

Zongwei Li, Zhonghang Li, Zirui Guo, Xubin Ren, and Chao Huang. 2025. DeepCode: Open Agentic Coding. *Preprint*, arXiv:2512.07921.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. 2026. Towards end-to-end automation of ai research. *Nature*, 651(8107):914–919.

Cong Lu, Shengran Hu, and Jeff Clune. 2025. Automated capability discovery via model self-exploration. In *ICLR 2025 Workshop on Foundation Models in the Wild*.

Alireza Mohammadshahi, Ali Shaikh, and Majid Yazdani. 2024. Leeroo orchestrator: Elevating llms performance through model integration. *arXiv preprint arXiv:2401.13979*, 3.

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. *Preprint*, arXiv:2307.07924.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using llm agents as research assistants. *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5977–6043.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2026. Paper2code: Automating code generation from scientific papers in machine learning. *Preprint*, arXiv:2504.17192.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, and 1 others. 2025. Paperbench: Evaluating ai's ability to replicate ai research. In *International Conference on Machine Learning*, pages 56843–56873. PMLR.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. 2026. Ai-researcher: Autonomous scientific innovation. *Advances in Neural Information Processing Systems*, 38:9481–9520.

Chunhui Wan, Xunan Dai, Zhuo Wang, Minglei Li, Yanpeng Wang, Yinan Mao, Yu Lan, and Zhiwen Xiao. 2025. Loongflow: Directed evolutionary search via a cognitive plan-execute-summarize paradigm. *arXiv preprint arXiv:2512.24077*.

Yidong Wang, Qi Guo, Wenjin Yao, Hongbo Zhang, Xin Zhang, Zhen Wu, Meishan Zhang, Xinyu Dai, Min Zhang, Qingsong Wen, and 1 others. 2024. Autosurvey: Large language models can automatically write surveys. *Advances in neural information processing systems*, 37:115119–115145.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. Autogen: Enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Tengyue Xu, Zhuoyang Qian, Gaoge Liu, Li Ling, Zhentao Zhang, Biao Wu, Shuo Zhang, Ke Lu, Wei Shi, Ziqi Wang, and 1 others. 2026. Idea2story: An automated pipeline for transforming research concepts into complete scientific narratives. *arXiv preprint arXiv:2601.20833*.

Bingyu Yan, Zhibo Zhou, Litian Zhang, Lian Zhang, Ziyi Zhou, Dezhuang Miao, Zhoujun Li, Chaozhuo Li, and Xiaoming Zhang. 2025. Beyond self-talk: A communication-centric survey of llm-based multi-agent systems. *Preprint*, arXiv:2502.14321.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems*, volume 37, pages 50528–50652. Curran Associates, Inc.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, and 1 others. 2026. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2025. Darwin godel machine: Open-ended evolution of self-improving agents. *arXiv preprint arXiv:2505.22954*.

Ruiyi Zhang, Peijia Qin, Qi Cao, Li Zhang, and Pengtao Xie. 2026. Aibuildai: An ai agent for automatically building ai models. *arXiv preprint arXiv:2604.14455*.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. Autocoderover: Autonomous program improvement. In *Proceedings*

of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, page 1592–1604, New York, NY, USA. Association for Computing Machinery.

Xinyu Zhu, Yuzhu Cai, Zexi Liu, Bingyang Zheng, Cheng Wang, Rui Ye, Jiaao Chen, Hanrui Wang, Wei-Chen Wang, Yuzhi Zhang, and 1 others. 2026. Toward ultra-long-horizon agentic science: Cognitive accumulation for machine learning engineering. *arXiv preprint arXiv:2601.10402*.