Asynchronous Human-Agent Rollout for Long-Horizon Task Training

Anonymous ACL submission

Abstract

Large Language Model (LLM) agents have recently shown strong potential in domains such as automated coding, deep research, and graphical user interface manipulation. However, training them to succeed on long-horizon tasks remains challenging. Current methods primarily fall into two categories. The first relies on dense human annotations through behavior cloning, which is prohibitively expensive for long-horizon tasks that can **take days or months**. The second depends on outcome-driven sampling, which often collapses due to the rarity of valid positive trajectories on domain-specialized tasks. We introduce APOLLO, a sampling framework that integrates asynchronous human guidance with action-level data filtering. Instead of requiring annotators to shadow every step, APOLLO allows them to intervene only when the agent drifts from a promising trajectory, by providing prior knowledge, strategic advice, etc. This lightweight design makes it possible to sustain interactions for over 30 hours and produces valuable trajectories at a lower cost. To demonstrate the effectiveness of APOLLO, we evaluate it using InnovatorBench, which requires both coding and deep research capabilities across day-long scientific discovery tasks. Our experiments show that when applied to train the GLM-4.5 model, APOLLO achieves more than a 50% improvement over the untrained baseline and a 28% improvement over a rejection sampling fine-tuning (RFT) variant trained without human interaction. These results highlight the critical role of human-in-the-loop sampling and the robustness of APOLLO's design in handling long-horizon scientific research tasks.

1 Introduction

Large Language Model (LLM) agents have recently proved remarkable progress across domains such as code (Yang et al., 2024), deep research (Zheng et al., 2025), and graphical user interface (GUI) manipulation (Liu et al., 2025a; Wang et al., 2025b). These advances highlight their potential to serve not only assistants but also as autonomous workers in specific tasks (Starace et al., 2025). As relatively simple tasks become saturated (Chang et al., 2024), the research focus is increasingly shifting toward **long-horizon, high-difficulty, and domain-specialized** scientific research tasks that demand excellent memory, sustained reasoning ability, and robust adaptability (OpenAI, 2025). Therefore, finding a high-quality data synthesis method tailored to long-horizon task rollout is crucial.

Existing training methodologies for LLM agents can be broadly divided into two paradigms. The first relies on behavior cloning with human annotators, in which human experts provide dense supervision by recording every action (Zhu et al., 2025; He et al., 2024). While capable of producing high-quality datasets, this paradigm suffers from prohibitive annotation costs, particularly for tasks that extend over days, weeks, or even months. The second paradigm focuses on self-sampling by LLMs, where powerful LLMs interact with synthetic environments, assign credit based on final results, and use this credit in rejection sampling fine-tuning (RFT) (Yuan et al., 2023) or other RL algorithms (Guo et al., 2025; Shao et al., 2024). Although scalable in principle, this paradigm frequently collapses on difficult tasks, as the probability of discovering valid positive trajectories is exceedingly low (Sane, 2025). As a result, neither dense human annotation nor self-driven reinforcement provides sustainable data for preparing agents to tackle long-horizon, real-world scientific research challenges.

To address these limitations, we propose APOLLO, a sampling framework that combines asynchronous human guidance with systematic action-level data filtering. Instead of requiring annotators to follow every step, APOLLO enables periodic monitoring of the agent's state, allowing high-level interventions only when the agent deviates from a promising trajectory. Such interven-

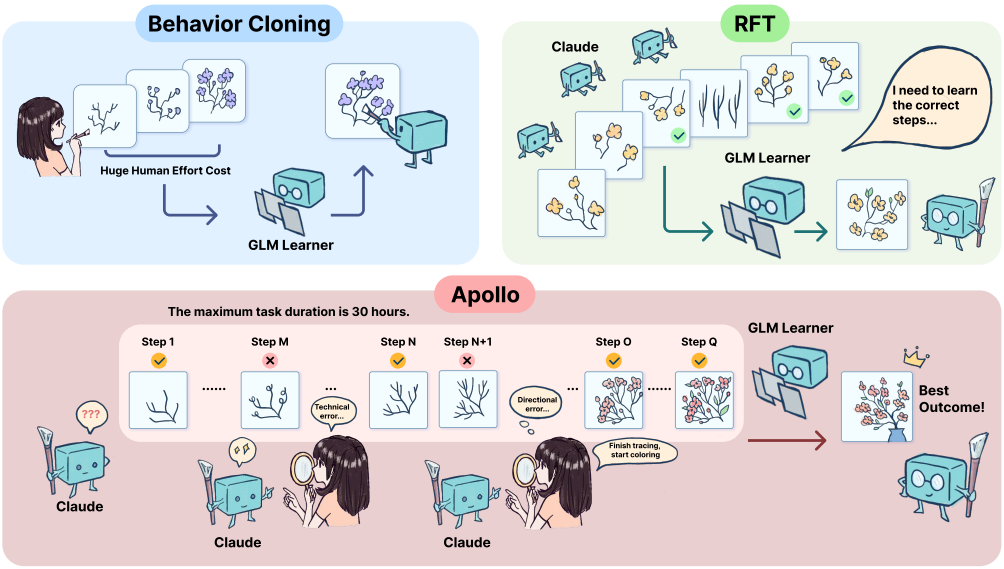

Figure 1: APOLLO allows humans to instruct Agents when they make both technical errors and strategic errors asynchronously and trains the model with correct steps. Behavior cloning methods require human annotators to create the entire trajectory from scratch, resulting in high human effort costs. While RFT can generate trajectories via the agent, the yield of correct trajectories or steps is often insufficient. APOLLO combines lightweight asynchronous human guidance with action-level filtering to produce high-quality trajectories at lower cost.

tions may include asking the sitiuation, pointing out mistakes, giving strategic advice, or providing general prior knowledge[1]. This lightweight oversight enables a person to interact with the agent for over 30 hours while improving the trajectory quality of long-horizon tasks compared to the non-interaction version. After getting valuable trajectories, APOLLO incorporates action-level supervision control that identifies and masks action segments inconsistent with either the adjusted plan or the environment's requirements (Fu et al., 2024).

To realize the asynchronous human guidance, the interface to facilitate human-agent interaction for Apollo is also important. Such an interface should minimize cognitive load while offering fine-grained control and clear information, enabling experts to provide targeted feedback without constantly being engaged during long runs (Ye et al., 2025a). In this paper, we create a human–AI interaction interface that integrates several channels for providing high-level guidance. By combining this interface with asynchronous rollout, APOLLO provides a practical framework for data collection in long-horizon environments.

To evaluate our approach, we adopt InnovatorBench (Wu et al., 2025), a benchmark of LLM research tasks that emphasizes end-to-end research capability. It captures the whole AI research process. This setting provides a natural testbed for APOLLO, as it requires long-horizon reasoning over multiple hours or days. Our experiments demonstrate the effectiveness of APOLLO: when applied to training GLM-4.5 (Zeng et al., 2025) on InnovatorBench, the model achieves more than 50% improvement over its untrained baseline and 28% improvement compared with a variant trained without human interaction. Besides, the trained model can also work longer than the original base model. These gains highlight both the **necessity of high-quality human-in-the-loop sampling** and the importance of selecting wise action in APOLLO's design towards long-horizon LLM research tasks.

In summary, this paper's contributions are:

- We propose an **asynchronous guidance algorithm** that enables annotators to provide high-level interventions without continuously shadowing the agent.
- We adapt an **action-level supervision control mechanism** to long-horizon settings, where failure judgments incorporate human feedback signals and operate under context summarization, and validate its effectiveness in stabilizing optimization.
- We design the **human–AI interaction inter-**

[1]For example, training 'Qwen2.5-VL-7B' model needs 'qwen2_vl' template in LLaMA-Factory.

face, which can minimize user cognitive load and provide fine-grained control for the agent during multi-day sampling processes.

- We conduct comprehensive experiments on InnovatorBench, showing performance improvements after training on APOLLO.

2 Related work

Training LLM agents has been studied both through domain-specific applications and through general finetuning methodologies (Parthasarathy et al., 2024). In application domains such as software engineering, code agents have become a primary testbed for developing and evaluating agent training techniques (Dong et al., 2025c). At the same time, methodological progress has centered on how rollouts are generated and exploited during finetuning, which critically affects data efficiency and stability (Xia et al., 2025). Together, these two strands frame the landscape of current research and provide the backdrop for APOLLO.

Code Agent Training Recent efforts have advanced the training of code agents in realistic software engineering settings (Phan et al., 2024). SWE-agent (Yang et al., 2024) introduced the a code scaffold to support repository navigation and patching; SWE-RL (Wei et al., 2025) leveraged reinforcement learning from search-replace patch; and OpenHands (Wang et al., 2024a) demonstrated that a lightweight but general toolset can enable broad code agents. Extensions such as SWE-Dev (Wang et al., 2025c) scale data through trajectory augmentation. Despite these advances, most code-agent training focuses on short-horizon software development tasks, where solutions can be validated within minutes or an hour. In contrast, APOLLO targets long-horizon scientific discovery tasks, where trajectories may span hours or even days.

Rollout Strategies in Finetuning Finetuning of LLM agents often hinges on how rollouts are generated and selected. Some approaches rely heavily on human-annotated rollouts, such as PC-Agent (Liu et al., 2025a; He et al., 2024) or process reward modeling, where annotators provide action-level feedback or trajectory validation; these strategies yield reliable supervision but incur high annotation costs (Wang et al., 2025a). Others adopt reject sampling rollouts. RFT (Yuan et al., 2023) filters sampled trajectories to keep only high-quality ones. Tool-STAR (Dong et al., 2025a), DeepResearcher (Zheng et al., 2025), and ToRL (Li et al., 2025b) explore rollouts in multi-tool invocation settings, and ARPO (Dong et al., 2025b) builds on that entropy-adaptive rollout mechanism. While these methods reduce dependency on dense human annotation, they still face challenges with instability and sparse-credit in long-horizon tasks. APOLLO differs by providing asynchronous step-level human guidance during rollout and applying action-level supervision control during training.

3 APOLLO: Asynchronous Rollout with Guidance for Agent Optimization

3.1 Preliminary

3.1.1 Markov Decision Process

We formalize the agent's interaction with the environment as a Markov Decision Process (Puterman, 1990), $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, P the transition dynamics, and r the reward function. In our setting, a state $s_t \in \mathcal{S}$ encodes the whole environment while an action $a_t \in \mathcal{A}$ corresponds to a tool invocation.

3.1.2 ReAct

To structure trajectories, we adopt the ReAct (Yao et al., 2023) paradigm, which unifies reasoning and acting in a loop. At each step, the agent first produces reasoning $r_t = \pi_\theta(o_0, a_1, o_1, ..., a_{t-1}, o_t)$, and then selects an action $a_t = \pi_\theta(o_0, a_1, o_1, ..., a_{t-1}, o_t, r_t)$, where o denotes to the observation. We remove the reasoning part in the trajectories, this produces trajectories of the form $\tau = \{(o_0), (a_1, o_1), \ldots, (a_T, o_T)\}$.

3.1.3 Long Context Management

A key challenge in long-horizon tasks is that trajectories often exceed the model's context length L. Naively concatenating all past actions and observations leads to truncation and information loss. To address this, a general way is to adopt a summarization strategy (Wang et al., 2024a). When $|\tau| > \eta L$ [2], earlier segments $\tau_{1:k}$ are compressed into a structured summary $\mathcal{S}_{1:k} = \Sigma(\tau_{1:k})$, and the trajectories is updated as $\hat{\tau} = \left[o_0, (\mathcal{S}_{1:k}, o_k), \tau_{k+1:t}\right]$, The summarization operator $\Sigma(\cdot)$ preserves the key knowledge, important intermediate results, environment or file states, and critical errors or reflections, ensuring that the agent maintains coherence while leaving space for future steps. This mechanism makes ReAct applicable to very long rollouts without exceeding memory limits.

[2] ηL = 100k tokens in this paper

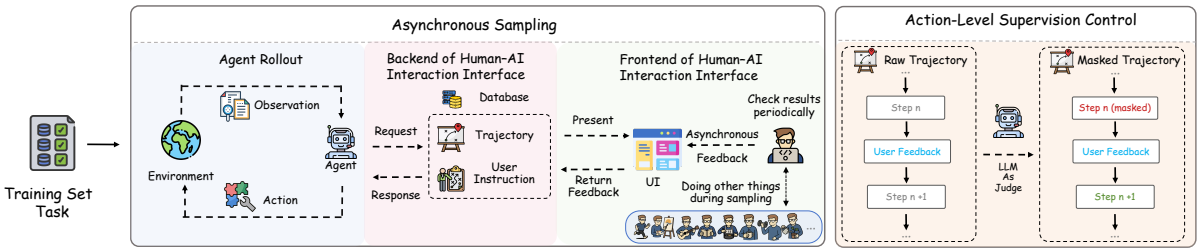

Figure 2: The pipeline of APOLLO to generate a made trajectory from the original training set tasks. It contains the Asynchronous Sampling Algorithm and Action-Level Supervision Control.

3.2 Human–AI Interaction Interface

Frontend The frontend interface provides human annotators with intuitive task management and real-time monitoring tools. As shown in Figure 6 to Figure 9, the frontend is divided into the task selection area, trajectory display area, terminal display area, file and search display area, and user input area. In the **task selection area**, annotators can easily view and switch between tasks, ensuring a clear understanding of the progress of each task. In the **trajectory display area**, annotators can view the entire history of a task's trajectories, automatically jump to specific positions based on keywords, and examine the context of specific decisions made during the task. In the **terminal display area**, annotators can view the latest output from each terminal of every host involved in the current task. In the **file and search display area**, annotators can access the latest modification records for each file, as well as the history of Google search queries. In the **user input area**, annotators can enter and submit commands at any time. The submitted commands are stored in the backend buffer, ensuring they do not interfere with the agent's reasoning process. To further enhance convenience, the interface includes an automatic update mechanism, ensuring that annotators can view real-time task information without needing to manually refresh.

Backend As shown in Figure 2, the backend architecture facilitates asynchronous interaction between the agent and the user interface, ensuring efficient management of information flow. After a task is established by the agent, a connection channel with a special identifier is created between the agent, the user, and the backend system. This involves setting up resources such as the conversation backend, cache storage in the database, and the user interface components corresponding to the special identifier. Once the connection is established, the system is prepared to receive user inputs and agent outputs. At any time, user inputs are buffered in the cache to prevent them from interfering with the agent's reasoning process. When the agent sends its output to the backend, the backend will store it in the database and send all buffered user inputs to the agent. The frontend interface can update the trajectory information based on the database. By decoupling these processes, the backend design allows for an optimized interaction model, balancing efficient agent processing with a smooth user experience in asynchronous settings.

3.3 Asynchronous Sampling Algorithm

Send trajectories in requests As shown in Figure 2 and Algorithm 1, the agent interacts with the backend by continuously updating its context, which consists a sequence of actions, observations, and thoughts. Each time the agent takes an action and receives an environment observation, it sends a request to the backend. This request includes the entire action-observation history, τ. Additionally, the request contains the new thought, action, observation, and a timestamp that marks when the interaction occurred. If summarization is performed during the current turn, the agent also send the context that includes both the summarized context and its results within the request.

Receive user inputs in response If the agent receives a user response, it is integrated directly into the input context to ensure the agent's decision-making aligns with the user's guidance. The user's response is tagged with a special identifier, `<real_user><\real_user>`, so that it is distinguishable from other system-generated information. The agent then appends this new input and the environment observation to its context. This process allows for dynamic interaction, where the agent's reasoning is continually informed by both environment observation and the user input.

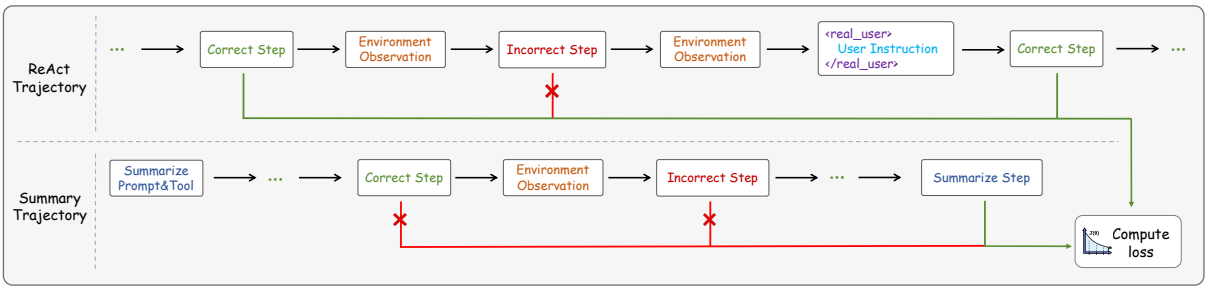

Figure 3: The display of training trajectory format. Only the green line step will be trained. In the summary trajectory, the correct step is not trained.

User interaction User interaction plays a crucial role in the algorithm. Its goal is for the agent to develop the ability to not only solve technical difficulties but also make strategic decision across different contexts. This approach also teaches the agent to function effectively within established training frameworks, such as LLaMA-Factory (Zheng et al., 2024) or VerL (Sheng et al., 2024). To facilitate this, annotators are encouraged to provide more generalizable guidance to the agent. For instance, rather than simply detailing the context in 'dataset_info.json' or providing a script for processing the original data, a better approach would be to teach the agent how to save multimodal data in ShareGPT (sha, 2023) format and correctly configure 'dataset_info.json' in LLaMA-Factory by reading the 'readme.md'. This methodology helps the agent recognize patterns and strategies that are applicable to more abstract tasks, such as evaluating model performance, optimizing workflows, or ensuring that test scripts run efficiently. Through this guidance, the agent evolves towards broader, flexible skills, extending beyond immediate actions to more adaptable, generalizable competencies. The detailed annotation protocol, including annotator backgrounds, intervention frequency, feedback types, and time costs, is provided in Appendix A.

3.4 Action-Level Supervision Control Mechanism

In addition to the asynchronous sampling algorithm, APOLLO integrates an action-level supervision control mechanism that identifies and masks erroneous actions before they contribute to the training loss, ensuring that the agent's behavior aligns with the desired trajectory. Although masking erroneous actions is explored in prior work (Fu et al., 2025), our mechanism differs in two key aspects. First, the failure judgment incorporates human feedback signals from the trajectory: actions that contradict earlier user guidance are identified and masked by an LLM judge, rather than relying solely on outcome-based rewards. Second, most prior work assumes short-horizon ReAct trajectories that fit within the model's context window; we validate that action-level masking remains effective under context summarization, where the agent's history is periodically compressed and earlier details are lost. Besides, the mechanism also combines symbolic-based masking . After masking erroneous steps, only the unmasked action portions compute the training loss.

4 Experiments

4.1 Datasets and Baselines

Environment We use ResearchGym (Wu et al., 2025) as our rollout and testing environment. ResearchGym is a control and execution platform that supports asynchronous command execution and multi-computer control, enabling long-horizon experiments. The system organizes 42 actions into five families: Command, File, Parse, Web Search, and Web Browse, and provides structured observations for agent-readable outputs. The agent interacts with the environment through a pipeline where actions are executed asynchronously, allowing uninterrupted task planning and execution.

Testing datasets We use the same testing dataset as InnovatorBench, which aggregates and standardizes a diverse range of AI research tasks. It emphasizes end-to-end research capabilities, capturing the full workflow from hypothesis formation to result analysis under realistic constraints. Each task within the dataset includes a task description, an initial code repository, associated datasets and checkpoints, as well as the evaluation script outside the agent's workspace. The agent's goal is to explore the task thoroughly and aim to achieve a performance that surpasses the ground-truth solution.

Models		DC	DF	DA	LD	RD	SC	Avg. Score
Close Source Models								
Claude	Final Score	25.47	30.89	22.73	12.98	**11.56**	36.63	**24.01**
	Best Score	26.88	31.47	22.73	12.98	**11.56**	37.74	**24.54**
GPT-5	Final Score	8.41	8.97	0.00	0.04	0.00	**60.07**	12.04
	Best Score	8.41	9.48	0.00	2.74	0.00	**60.07**	12.52
Open Source Models								
Kimi-K2	Final Score	14.01	7.39	2.47	0.00	3.23	3.33	5.35
	Best Score	14.08	7.97	2.47	0.00	3.23	3.33	5.45
GLM-4.5	Final Score	15.29	5.16	**25.49**	7.63	0.00	3.33	11.85
	Best Score	22.64	5.36	**25.49**	7.63	0.00	3.33	13.35
SFT-Based Model								
APOLLO	Final Score	**27.33**	**40.32**	23.27	**21.48**	3.09	6.67	21.86
	Best Score	**27.50**	**40.47**	23.27	**25.23**	3.09	16.83	24.01

Table 1: **Performance comparison on InnovatorBench.** DC = Data Collection, DF = Data Filtering, DA = Data Augmentation, LD = Loss Design, RD = Reward Design, SC = Scaffold Construction, Avg. Score = Weighted Average Score. *Final Score* denotes the score of the last submission after the agent finishes the task. *Best Score* is the highest score achieved by the agent.

We believe this benchmark is long-horizon, high-difficulty, and domain-specialized, which aligns with the purpose of APOLLO. All of our experiments are under the non-hint version.

Training datasets To align with the InnovatorBench, we construct 18 training tasks on the ResearchGym. During trajectory rollout, we use Claude-4-Sonnet and ask the human annotators to instruct the agent based on the principle mentioned in §3.3 and Figure 2. The task details, annotation protocol, and dataset statistics are provided in Appendix B and C. As shown in Figure 3, the ReAct trajectory will only compute the correct action part's loss, and the summarization trajectory will only train the last action ; the other action will be masked. Since the number of summarization trajectories is always one less than the number of ReAct trajectories, to make the training data more balanced, we upsample the ReAct trajectories 7 times and the summarization trajectory 10 times.

Training We use GLM-4.5 as our base model. We modified the smile code [3] to support multiturn training with correct action masking. All models are trained with a max token of 128k, 1 epoch, batch size 64, and a learning rate from 5e-6 to 1e-6 with cosine annealing.

Baselines We compare APOLLO with both closed-source and open-source models. For the closed-source models, we use GPT-5 (OpenAI, 2025) and Claude Sonnet 4 (Anthropic, 2025). For the open-source models, we use Kimi-K2 (Team et al., 2025) and GLM-4.5 (Zeng et al., 2025). We report these models' scores from InnovatorBench and use the same environment and scaffold to evaluate our own model. For the ablation study, we train a variant without human interaction, denoted as RFT, and a variant without masking. Note that InnovatorBench scores are continuous real values between 0 and 1 rather than binary success/failure, so traditional RFT filtering for "successful" trajectories is not directly applicable; instead, both APOLLO and the RFT baseline train on all 18 tasks and rely on action-level masking to handle quality. A discussion of potential leakage between training and test tasks is provided in Appendix D.

4.2 Main Results

Table 1 presents the comparison of various models' performance on InnovatorBench across six research domains. APOLLO consistently outperforms GLM-4.5, particularly in Data Collection, Data Filtering, and Loss Design. For example, in Data Collection, APOLLO achieves a Final Score of 27.33, which is significantly higher than GLM-4.5's score of 15.29, underscoring APOLLO's superior performance in gathering and processing data. The improvement is even more pronounced in Loss Design, where APOLLO's Best Score of 25.23 surpasses GLM's 7.63.

[3] https://github.com/THUDM/slime

Models		DC	DF	DA	LD	RD	SC	Avg. Score
Open Source Models								
GLM-4.5	Final Score	15.29	5.16	**25.49**	7.63	0.00	3.33	11.85
	Best Score	22.65	5.36	**25.49**	7.63	0.00	3.33	13.35
SFT-Based Models								
APOLLO	Final Score	27.33	**40.32**	23.27	**21.48**	**3.09**	6.67	**21.86**
	Best Score	27.50	**40.47**	23.27	**25.23**	**3.09**	**16.83**	**24.01**
– w/o Masking	Final Score	**37.22**	24.19	23.20	1.82	0.33	**8.55**	18.46
	Best Score	**37.22**	25.16	23.20	1.82	0.33	8.55	18.61
– w/o Interaction (RFT)	Final Score	15.63	37.74	6.87	7.70	0.00	6.67	12.66
	Best Score	15.63	37.74	6.87	7.70	**3.09**	6.67	12.97

Table 2: **Performance comparison on InnovatorBench.** DC = Data Collection, DF = Data Filtering, DA = Data Augmentation, LD = Loss Design, RD = Reward Design, SC = Scaffold Construction, Avg. Score = Weighted Average Score. *Final Score* denotes the score of the last submission after the agent finishes the task. *Best Score* is the highest score achieved by the agent.

Notably, we find APOLLO gains a huge improvement in task 15, from 22.90 to 75.69. However, task 15 is based on the *alignment-handbook* (Tunstall et al.) framework, which hasn't been trained in the training set. The fact that APOLLO still attained such a high score indicates that APOLLO excels at transferring knowledge, likely by leveraging the algorithm's generalization nature across related tasks. This ability to adapt to unfamiliar frameworks or new task structures highlights APOLLO's versatility and its potential to handle complex, previously unseen problems. This approach allows APOLLO to adapt more effectively to complex problem-solving tasks. Additionally, APOLLO outperforms Claude in several domains, such as Data Filtering, where APOLLO maintains a consistent performance at 40.47, while Claude Sonnet 4 is 31.47. This reflects that the interactive feedback mechanism contributes to APOLLO's ability to generate solutions that are more practically adaptive.

In conclusion, APOLLO's strong performance, especially in comparison to GLM-4.5, highlights the effectiveness of its dynamic, feedback-driven rollout process. By training on more high-quality, and relevant data, APOLLO demonstrates how interactive learning can significantly enhance performance across various research domains.

4.3 Ablation Study

Table 2 presents the ablation results of APOLLO. From the table, we can see that APOLLO outperforms the RFT baseline in all research domains. The RFT baseline can only learn from the sampling model (i.e., Claude). In contrast, APOLLO learns from both Claude and human experts who are relatively professional in the research field, always trying to use the most appropriate reasoning to solve the problem. When considering the effect of the action-level supervision control mechanism, APOLLO outperforms the model trained without bad action masking in five out of six research domains, especially in Loss Design, from 1.82 to 25.23. This is because the action-level supervision control mechanism can help APOLLO to avoid learning some bad actions, such as the action with an error message observation, and emphasize wise decision-making. As a result, it enhances the probability for the agent to make strategic and reliable decisions as well as the final performance.

In summary, the ablation study demonstrates that APOLLO not only benefits from the combination of model knowledge and human expertise but also gains robustness through action-level supervision. These two factors enable APOLLO to achieve stronger reasoning ability, more reliable decision-making, and consistently superior performance across diverse research domains.

4.4 Test-Time Scaling Result

The test time scaling results reflect the model's ability to handle difficult tasks. Figure 4 shows the test-time scaling results. Each node represents the average score across 20 tasks when the agent works for a certain time. If the agent hasn't evaluated the task yet, its score is 0. If the agent has been evaluated multiple times, its score is the last evaluation score before a certain time. In this figure, we can see the following findings:

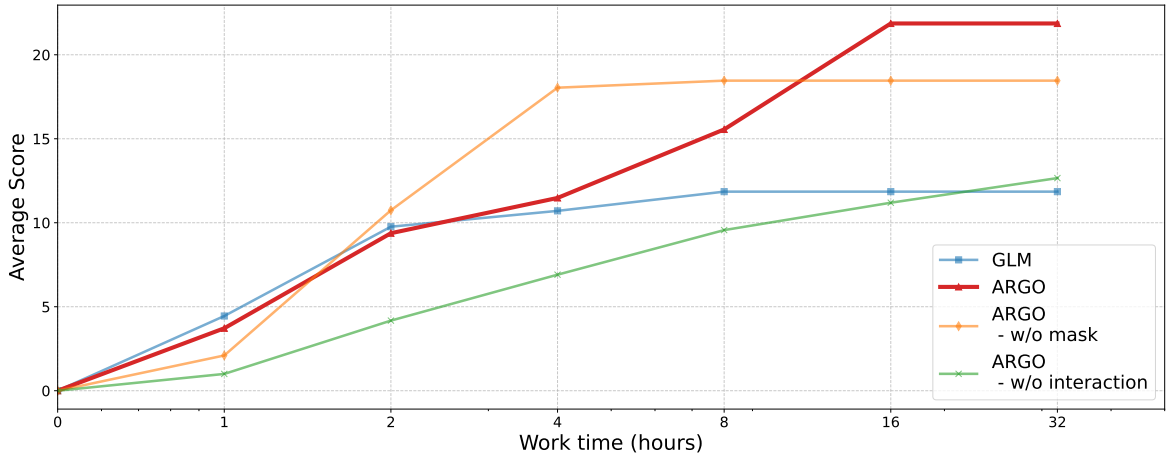

Figure 4: The test-time scaling score of four GLM-4.5 series model.

APOLLO can use a longer time to achieve the best performance. This figure shows that the results of APOLLO and GLM-4.5 are similar in the first 4 hours, but APOLLO can continue to improve its performance until 16 hours. On the contrary, GLM-4.5's performance is saturated after 4 hours. This result reflects the model's ability to spend more time to achieve the best performance.

The training without masking bad action can achieve promising results at the beginning, but the performance saturation point is much lower than APOLLO. The pink line in this figure shows the result of the model trained without masking bad action, which is higher than the blue line. It also achieves better results than the red line in the first 8 hours, but the performance saturates in 4 hours. As a result, the red line surpasses the pink line in 16 hours. The result reflects that learning from both bad actions and good actions can still improve the model's research ability, but learning too many bad actions would eventually hurt the model's ability to continuously improve its performance, such as refining its bad actions reasonably (Fu et al., 2025).

Human-Agent interaction is the key to improving the model's ability. It is clear that the RFT baseline (without human-agent interaction) performs much lower than APOLLO, and is even lower than the untrained model most of the time. This is because the decision-making by Claude alone is sub-optimal, which may harm the model's ability to deal with hard situations or design effective algorithms. For example, this model tries to use *transformers* to do inference, instead of using *vllm*, which causes a huge time cost (i.e., 31 hours) to finish the task. And the performance of this task is not as good as APOLLO's.

All in all, the test-time scaling results show that APOLLO is the most effective model to improve the model's research ability, and human-agent interaction is the key to improving the model's ability.

5 Conclusion

In this work, we introduced APOLLO, a novel sampling framework designed to address the challenges of training LLM agents on long-horizon scientific research tasks. By combining asynchronous human guidance with an action-level supervision control mechanism, APOLLO significantly reduces the cost of human oversight while ensuring the quality and stability of collected trajectories. The human-AI interaction interface further enables lightweight yet effective interventions, making the framework both practical and scalable.

Through comprehensive evaluation on InnovatorBench, we demonstrated that APOLLO outperforms untrained baselines. Our ablation studies confirm that both asynchronous guidance and action-level filtering are essential to achieving robust improvements, while test-time scaling experiments show APOLLO's ability to sustain performance gains over extended horizons. These findings suggest that APOLLO not only enhances data efficiency but also facilitates transferable reasoning strategies, enabling agents to adapt to new frameworks and complex research environments.

Overall, APOLLO offers a promising path toward training LLM agents capable of performing long-horizon scientific research tasks. We believe this paradigm will last for a long time until multi-agent systems can discover problems and provide advice better than the most professional humans.

Limitations

Our work has several limitations:

- **Single benchmark and model.** We evaluate APOLLO on a single benchmark (InnovatorBench) with a single base model (GLM-4.5). While InnovatorBench covers both coding and deep research capabilities across day-long tasks, the generalizability to other long-horizon domains (e.g., legal discovery, creative writing, embodied tasks) remains to be validated.
- **Annotation cost.** Despite being more lightweight than behavior cloning, the asynchronous annotation process still requires domain experts to periodically monitor agent progress over multi-day sessions, which incurs non-trivial cost.
- **Domain-specific masking rules.** Our action-level supervision relies on symbolic rules and an LLM judge whose masking criteria may need to be redesigned for new task domains.
- **Environment specialization.** The framework is currently specialized for terminal/coding-based environments; adapting the human–AI interaction interface and observation structure to other modalities would require additional engineering effort.

References

2023. Sharegpt.

Anthropic. 2025. Introducing claude 4. Accessed: 2025-09-22.

Baolong Bi, Shenghua Liu, Xingzhang Ren, Dayiheng Liu, Junyang Lin, Yiwei Wang, Lingrui Mei, Junfeng Fang, Jiafeng Guo, and Xueqi Cheng. 2025. Refinex: Learning to refine pre-training data at scale from expert-guided programs. *arXiv preprint arXiv:2507.03253*.

Ma Chang, Junlei Zhang, Zhihao Zhu, Cheng Yang, Yujiu Yang, Yaohui Jin, Zhenzhong Lan, Lingpeng Kong, and Junxian He. 2024. Agentboard: An analytical evaluation board of multi-turn llm agents. *Advances in neural information processing systems*, 37:74325–74362.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, and 1 others. 2025. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*.

Guanting Dong, Yifei Chen, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Yutao Zhu, Hangyu Mao, Guorui Zhou, Zhicheng Dou, and Ji-Rong Wen. 2025a. Tool-star: Empowering llm-brained multi-tool reasoner via reinforcement learning. *arXiv preprint arXiv:2505.16410*.

Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, and 1 others. 2025b. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*.

Yihong Dong, Xue Jiang, Jiaru Qian, Tian Wang, Kechi Zhang, Zhi Jin, and Ge Li. 2025c. A survey on code generation with llm-based agents. *arXiv preprint arXiv:2508.00083*.

Xinrun Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, King Zhu, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, and 1 others. 2025. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. *arXiv preprint arXiv:2502.14739*.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. 2025. Agentrefine: Enhancing agent generalization through refinement tuning. *arXiv preprint arXiv:2501.01702*.

Dayuan Fu, Jianzhao Huang, Siyuan Lu, Guanting Dong, Yejie Wang, Keqing He, and Weiran Xu. 2024. Preact: Prediction enhances agent's planning ability. *arXiv preprint arXiv:2402.11534*.

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

Yanheng He, Jiahe Jin, Shijie Xia, Jiadi Su, Runze Fan, Haoyang Zou, Xiangkun Hu, and Pengfei Liu. 2024. Pc agent: While you sleep, ai works–a cognitive journey into digital world. *arXiv preprint arXiv:2412.17589*.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. 2024. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Keyu Li, Mohan Jiang, Dayuan Fu, Yunze Wu, Xiangkun Hu, Dequan Wang, and Pengfei Liu. 2025a. Datasetresearch: Benchmarking agent systems for demand-driven dataset discovery. *arXiv preprint arXiv:2508.06960*.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025b. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*.

Yiyuan Li, Shichao Sun, and Pengfei Liu. 2024. Frog: Evaluating fuzzy reasoning of generalized quantifiers in large language models. *arXiv preprint arXiv:2407.01046*.

Haowei Liu, Xi Zhang, Haiyang Xu, Yuyang Wanyan, Junyang Wang, Ming Yan, Ji Zhang, Chunfeng Yuan, Changsheng Xu, Weiming Hu, and 1 others. 2025a. Pc-agent: A hierarchical multi-agent collaboration framework for complex task automation on pc. *arXiv preprint arXiv:2502.14282*.

Yuqi Liu, Bohao Peng, Zhisheng Zhong, Zihao Yue, Fanbin Lu, Bei Yu, and Jiaya Jia. 2025b. Seg-zero: Reasoning-chain guided segmentation via cognitive reinforcement. *arXiv preprint arXiv:2503.06520*.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*.

OpenAI. 2025. Gpt-5: Language model. Accessed: 2025-09-22.

Venkatesh Balavadhani Parthasarathy, Ahtsham Zafar, Aafaq Khan, and Arsalan Shahid. 2024. The ultimate guide to fine-tuning llms from basics to breakthroughs: An exhaustive review of technologies, research, best practices, applied research challenges and opportunities. *arXiv preprint arXiv:2408.13296*.

Huy Nhat Phan, Tien N Nguyen, Phong X Nguyen, and Nghi DQ Bui. 2024. Hyperagent: Generalist software engineering agents to solve coding tasks at scale. *arXiv preprint arXiv:2409.16299*.

Martin L Puterman. 1990. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434.

Soham Sane. 2025. Hybrid group relative policy optimization: A multi-sample approach to enhancing policy optimization. *arXiv preprint arXiv:2502.01652*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Yueqi Song, Tianyue Ou, Yibo Kong, Zecheng Li, Graham Neubig, and Xiang Yue. 2025. Visualpuzzles: Decoupling multimodal reasoning evaluation from domain knowledge. *arXiv preprint arXiv:2504.10342*.

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. *arXiv preprint arXiv:2504.01848*.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, and 1 others. 2025. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Shengyi Huang, Kashif Rasul, Alvaro Bartolome, Carlos M. Patiño, Alexander M. Rush, and Thomas Wolf. The Alignment Handbook.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. 2025a. Steca: Step-level trajectory calibration for llm agent learning. *arXiv preprint arXiv:2502.14276*.

Haoming Wang, Haoyang Zou, Huatong Song, Jiazhan Feng, Junjie Fang, Junting Lu, Longxiang Liu, Qinyu Luo, Shihao Liang, Shijue Huang, and 1 others. 2025b. Ui-tars-2 technical report: Advancing gui agent with multi-turn reinforcement learning. *arXiv preprint arXiv:2509.02544*.

Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. 2025c. Swe-dev: Building software engineering agents with training and inference scaling. *arXiv preprint arXiv:2506.07636*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024a. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Yejie Wang, Keqing He, Dayuan Fu, Zhuoma Gongque, Heyang Xu, Yanxu Chen, Zhexu Wang, Yujia Fu, Guanting Dong, Muxi Diao, and 1 others. 2024b. How do your code llms perform? empowering code instruction tuning with high-quality data. *arXiv preprint arXiv:2409.03810*.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I Wang. 2025. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *arXiv preprint arXiv:2502.18449*.

Yunze Wu, Dayuan Fu, Weiye Si, Zhen Huang, Mohan Jiang, Keyu Li, Shijie Xia, Jie Sun, Tianze Xu, Xiangkun Hu, Pengrui Lu, Xiaojie Cai, Lyumanshan Ye, Wenhong Zhu, Yang Xiao, and Pengfei Liu. 2025. Innovatorbench: Evaluating agents' ability to conduct innovative llm research. *Preprint*, arXiv:2510.27598.

Yu Xia, Yiran Shen, Junda Wu, Tong Yu, Sungchul Kim, Ryan A Rossi, Lina Yao, and Julian McAuley. 2025. Sand: Boosting llm agents with self-taught action deliberation. *arXiv preprint arXiv:2507.07441*.

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

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

Lyumanshan Ye, Xiaojie Cai, Xinkai Wang, Junfei Wang, Xiangkun Hu, Jiadi Su, Yang Nan, Sihan Wang, Bohan Zhang, Xiaoze Fan, and 1 others. 2025a. Interaction as intelligence: Deep research with human-ai partnership. *arXiv preprint arXiv:2507.15759*.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025b. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *Preprint*, arXiv:2308.01825.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

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

Muzhi Zhu, Yuzhuo Tian, Hao Chen, Chunluan Zhou, Qingpei Guo, Yang Liu, Ming Yang, and Chunhua Shen. 2025. Segagent: Exploring pixel understanding capabilities in mllms by imitating human annotator trajectories. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 3686–3696.

Wenhong Zhu, Zhiwei He, Xiaofeng Wang, Pengfei Liu, and Rui Wang. 2024. Weak-to-strong preference optimization: Stealing reward from weak aligned model. *arXiv preprint arXiv:2410.18640*.

A Annotation Protocol Details

We use 2 annotators for the asynchronous rollout, both of whom are AI researchers with hands-on experience in LLM training and evaluation. Annotators were instructed to monitor the agent's progress and intervene when the agent drifts from a promising trajectory. The key aspects of the protocol are as follows:

Intervention frequency. Annotators check the agent's results approximately every 2 hours during normal operation. During critical phases (e.g., when the agent is creating training scripts or debugging), the checking frequency increases to every 10 minutes. Annotators do not need to monitor during sleeping hours, ensuring the process remains flexible.

Types of feedback. The feedback typically falls into three categories: (1) *Status inquiry*: asking the agent about its current situation, or checking the logs directly; (2) *Strategic suggestions*: providing high-level guidance on experiment design, such as suggesting alternative approaches or pointing out overlooked aspects; (3) *Co-debugging*: helping the agent diagnose and fix specific technical issues.

Guidelines. Annotators were encouraged to provide generalizable guidance rather than direct answers. For example, rather than providing a specific configuration file, annotators would teach the agent how to read the documentation and configure the framework correctly (as described in §3.3).

Time cost. Creating a single training trajectory typically takes between 5 to 36 hours of wall-clock time, during which the annotator provides 0–5 feedback messages. The annotators do not execute actions themselves; they only provide textual guidance.

B Training Data Statistics

Table 3 summarizes the key statistics of the training datasets for APOLLO and the RFT baseline.

C Training Data Details

We create 18 tasks in our training set; Table 4 and Table 5 list their papers, key descriptions, constraints, and research domains. The annotators'

Statistic	APOLLO	RFT
Number of tasks	18	18
Total turns (after upsampling)	755	434
Input tokens	45.4M	27.6M
Training tokens	204K	58.3K
Avg. trajectory duration (hours)	6.91	4.46

Table 3: Training dataset statistics for APOLLO and the RFT baseline. All trajectories are retained; erroneous steps are masked rather than discarded.

workload is similar to Wu et al. (2025). Task 3, 4, 8, 9, 13–18 use the same background paper as used in InnovatorBench, but their real tasks are different. Task 3 needs to design an efficient model (use fewer tokens in reasoning), which is different from avoiding entropy collapse in InnovatorBench. Task 4 and 14–18 use a different dataset compared with InnovatorBench. Task 8 wants the model to filter the answer instead of just finding the question, which is used in InnovatorBench. Task 9 wants the model to design a code answer quality filter and a diversity filter, but InnovatorBench wants the model to design a code question complexity filter. Task 13 only uses the search database in search-R1 and asks the agent to design a workflow, which is hugely different from reward design in InnovatorBench. The other tasks use different background papers compared with InnovatorBench. This aligns with our design principle—the training set data should be different from the test dataset at the task level.

D Information Leakage Analysis

A potential concern is whether training and test tasks share information that could inflate results. We address this as follows. First, ResearchGym is a *research execution environment* (analogous to a scaffold like ReAct), not a data source; using the same environment does not constitute information leakage. Second, InnovatorBench is a scientific research benchmark where **no ground truth or optimal solution exists**—agents are evaluated on a continuous scale based on how well their solutions perform relative to reference implementations.

Although some training tasks share background papers with test tasks, the actual tasks are substantially different. For instance, designing a reward function for SQL tasks versus GUI grounding tasks requires entirely different domain knowledge, even if both fall under "reward design." Transferable knowledge (e.g., how to use the verl framework) represents legitimate generalization rather than leakage. To further validate this, we note that APOLLO achieves its largest improvement on task 15 (from 22.90 to 75.69), which is based on the *alignment-handbook* framework—a framework that does not appear in any training task.

E Introduction to INNOVATORBENCH

InnovatorBench (Wu et al., 2025) is a benchmark-platform pair designed to evaluate AI research agents in realistic, end-to-end Large Language Model (LLM) research workflows. Unlike prior benchmarks that focus on isolated skills or simplified environments, InnovatorBench emphasizes integrated research capabilities across multiple stages of LLM development.

E.1 Benchmark Overview and Statistics

InnovatorBench consists of 20 research tasks from 14 influential papers, covering various LLM research areas. Tasks are sourced from top-tier venues, including NeurIPS, ICLR, ACL, etc., ensuring diverse experimental paradigms and coding practices. The benchmark evaluates AI agents in areas like data construction, loss design, reward design, and scaffold construction. The InnovatorBench dataset contains the following:

E.2 Task Description

Each task is defined by the following components:

- *Motivation*: The origin and significance of the research question.
- *Task*: A high-level description of the agent's objective, and its target.
- *Data*: Details on the datasets, checkpoints, storage paths, and formats.
- *Constraints*: Operational limits, such as time and GPU quotas.
- *Evaluations*: Metrics like accuracy and F1 score, with reference solutions for comparison.
- *Environment*: Information about the execution environment, including conda setup.
- *Scripts*: Pre-built helper scripts for data handling, training, and evaluation.

Table 4: **Performance comparison on various models when tested against various evaluation metrics (Part I).**

ID	Paper	Key Description	Constrain	Research Domains
1	SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning (Ma et al., 2025)	Design, implement, and evaluate a multi-component reward function for NL2SQL reinforcement learning to maximize execution accuracy on complex queries using the Qwen2.5-Coder-7B-Instruct model and BIRD benchmark.	Qwen2.5-Coder-7B-Instruct design reward function only 16h, 8×80GB GPUs	Reward Design
2	Seg-Zero: Reasoning-Chain Guided Segmentation via Cognitive Reinforcement (Liu et al., 2025b)	Design, implement, and evaluate a novel multi-component reward function for RL training of a reasoning segmentation model (Qwen2.5-VL-7B-Instruct + SAM2) to maximize gIoU on ReasonSeg and RefCOCOg benchmarks.	Qwen2.5-VL-7B-Instruct design reward function only 16h, 8×80GB GPUs	Reward Design
3	DAPO: An Open-Source LLM Reinforcement Learning System at Scale (Yu et al., 2025)	Design and implement a length-aware reward function in RL training (based on Qwen2.5-1.5B and verl) to reduce reasoning trace length while preserving or improving mathematical accuracy on MATH500.	Qwen2.5-1.5B 48h, 8×80GB GPUs	Reward Design
4	Visual SKETCHPAD: Sketching as a Visual Chain of Thought for Multimodal Language Models (Hu et al., 2024)	Build a unified GPT-4o–based reasoning framework to solve graph isomorphism, function parity, and chess winner tasks, generating structured JSON outputs for all test samples with accurate answers and reasoning.	GPT-4o 12h, 0 GPU	Scaffold Construction
5	Supergpqa: Scaling llm evaluation across 285 graduate disciplines (Du et al., 2025)	Enhance and fine-tune Qwen2.5-7B-Instruct with enriched scientific reasoning datasets to improve cross-domain reasoning accuracy, then generate a file containing final multiple-choice answers for all test problems.	48h, 8×80GB GPUs final model trained from Qwen2.5-7B	Data Augmentation
6	FRoG: Evaluating Fuzzy Reasoning of Generalized Quantifiers in Large Language Models (Li et al., 2024)	Enhance Qwen2.5-7B-Instruct through dataset enrichment and fine-tuning to improve fuzzy reasoning on mathematical word problems with generalized quantifiers, and evaluate performance on the test set.	48h, 8×80GB GPUs final model trained from Qwen2.5-7B	Data Augmentation
7	VISUALPUZZLES: Decoupling Multimodal Reasoning Evaluation from Domain Knowledge (Song et al., 2025)	Enhance Qwen2.5-VL-7B-Instruct through dataset augmentation and fine-tuning to improve abstract visual reasoning on multimodal puzzles and evaluate accuracy on the test set.	Qwen2.5-VL-7B-Instruct data construction / training validation / test inference 48h, 8×80GB GPUs	Data Augmentation
8	Limo: Less is more for reasoning (Ye et al., 2025b)	Develop a problem curation system to select exactly 800 high-quality math QA pairs from 4905 candidates, train a model on them, and maximize reasoning accuracy on dev/test sets.	fixed training hyperparameter select 800 QA pairs 48h, 8×80GB GPUs	Data Filtering
9	How Do Your Code LLMs Perform? Empowering Code Instruction Tuning with High-Quality Data (Wang et al., 2024b)	Implement a system for selecting high-quality, diverse code responses based on quality and complexity scores, and perform analysis on the distribution of these selections for improved model training.	24h, 8×80GB GPUs	Data Filtering

Table 5: **Performance comparison on various models when tested against various evaluation metrics (Part II).**

ID	Paper	Key Description	Constrain	Research Domains
10	Refinex: Learning to refine pre-training data at scale from expert-guided programs (Bi et al., 2025)	Implement a deletion-based cleaning approach to refine noisy web data by removing irrelevant content while preserving high-quality portions, without introducing new vocabulary, and submit the cleaned dataset for evaluation.	5h, 8×80GB GPUs high efficiency	Data Filtering
11	MiniMax-M1: Scaling Test-Time Compute Efficiently with Lightning Attention (Chen et al., 2025)	Implement a new RL loss function to maximize mathematical reasoning accuracy in training a model using the GRPO algorithm and evaluate it on a provided test set.	24h, 8×80GB GPUs	Loss Design
12	Weak-to-strong preference optimization: Stealing reward from weak-aligned model (Zhu et al., 2024)	Implement the wspo (Weak-to-Strong Preference Optimization) algorithm to transfer alignment from a weak but aligned model to a strong but not aligned model, then train and evaluate the model to maximize performance on a provided test set.	12h, 8×80GB GPUs	Loss Design
13	Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning (Jin et al., 2025)	Implement a general-purpose search-augmented question answering workflow using the Qwen2.5-72B model, which dynamically decides when to use external knowledge retrieval to answer diverse questions, ensuring robust and scalable reasoning.	Qwen2.5-72B Inference Only 24h, 8×80GB GPUs	Scaffold Construction
14	Visual SKETCHPAD: Sketching as a Visual Chain of Thought for Multimodal Language Models (Hu et al., 2024)	Develop a visual reasoning system using GPT-4o to solve multimodal perception, spatial relationship, and semantic correlation tasks with maximum accuracy.	GPT-4o 12h, 1×24GB GPU	Scaffold Construction
15	DatasetResearch: Benchmarking Agent Systems for Demand-Driven Dataset Discovery (Li et al., 2025a)	Create or find Moroccan Darija-to-English translation datasets, fine-tune Llama-3.1-8B-Instruct with full parameter training, and achieve maximum BLEU score improvement over baseline.	Llama-3.1-8B-Instruct dataset discovery / synthesis 48h, 8×80GB GPUs	Data Construction
16	DatasetResearch: Benchmarking Agent Systems for Demand-Driven Dataset Discovery (Li et al., 2025a)	Create or find English-to-Luganda translation datasets, fine-tune Llama-3.1-8B-Instruct with full parameter training, and achieve maximum BLEU score improvement over baseline.	Llama-3.1-8B-Instruct dataset discovery / synthesis 48h, 8×80GB GPUs	Data Construction
17	DatasetResearch: Benchmarking Agent Systems for Demand-Driven Dataset Discovery (Li et al., 2025a)	Create or find multilingual text classification datasets for sentence completion tasks, fine-tune Llama-3.1-8B-Instruct with full parameter training, and achieve maximum accuracy improvement over baseline.	Llama-3.1-8B-Instruct dataset discovery / synthesis 48h, 8×80GB GPUs	Data Construction
18	DatasetResearch: Benchmarking Agent Systems for Demand-Driven Dataset Discovery (Li et al., 2025a)	Create or find medical text classification datasets for yes/no binary classification tasks, fine-tune Llama-3.1-8B-Instruct with full parameter training, and achieve maximum accuracy improvement over baseline.	Llama-3.1-8B-Instruct dataset discovery / synthesis 48h, 8×80GB GPUs	Data Construction

E.3 Workspace

The workspace is a writable directory containing the necessary artifacts for each task:

- *Conda Environment*: A pre-built conda environment replicating the original paper's setup.
- *Data*: Datasets and pre-trained model checkpoints for fine-tuning, with options for augmenting data.
- *Task Directory*: The task's code repository and supplementary scripts for model training and evaluation.

E.4 Evaluations

Evaluations follow a Kaggle-style procedure with multiple submissions and feedback:

- Submissions are first checked for format validity, with invalid ones scoring 0.
- Valid submissions are scored on a scale from 0 (baseline) to 100 (surpassing reference solution).
- Scores increase linearly based on performance, with a reference solution as the target.

E.5 Benchmark Design

The benchmark consists of 20 tasks covering:

- Data Construction, Filtering, and Augmentation
- Loss and Reward Function Design
- Scaffold Construction

Each task requires the agent to produce runnable artifacts and is evaluated along dimensions such as correctness, performance, output quality, and uncertainty. Reference implementations exist for reproducibility, but agents must independently generate their solutions, encouraging creativity and innovation.

F ResearchGym Environment

To support execution, the InnovatorBench's authors introduce RESEARCHGYM, a research environment that provides:

- A rich action space in 5 domains.
- Support for long-horizon and distributed experiments running for hours or days.
- Asynchronous monitoring, process adaptation, and snapshot saving/loading for recovery.

RESEARCHGYM is extensible, enabling the community to contribute tasks, datasets, and protocols, similar to open platforms like HuggingFace.

G Asynchronous Sampling Algorithm

Algorithm 1 presents the asynchronous sampling algorithm in both client part (part A) and server part (part B)

H Case Study

Figure 5 presents the key action steps taken by two models during the completion of the InnovatorBench. Compared to the original GLM-4.5, Apollo demonstrates better patience, stronger iterative Refinement capabilities, and improved adaptability to the task. Figure 5(a) shows the results of both models during data augmentation and training. Apollo optimally utilizes resources by distributing the generation tasks across two machines, which reflects its ability to perform targeted optimizations based on available resources. Furthermore, when faced with tasks that require more than an hour to fully generate and train, Apollo chooses to wait for the task to complete by selecting extended sleep periods. In contrast, GLM-4.5 opts for shorter wait times of 30 or 60 seconds, leading it to prematurely terminate the training process while the model is still importing the vLLM library, ultimately causing the failure of the LLM-based Chain-of-Thought (CoT) synthesis method. As a result, GLM-4.5 can only use fixed templates for CoT synthesis, which leads to homogenization of the training data and training failure. This disparity highlights Apollo's stronger patience, which is crucial for long-horizon tasks. Figure 5(b) shows the results of the two models in the data cleaning task. After each evaluation, Apollo reflects on the results and dynamically adjusts its filtering strategy based on the feedback, ultimately achieving a score of 13.84. In contrast, GLM does not take into account the actual feedback from the environment after the first evaluation; instead, it continues to rely on its self-generated metrics, believing that its cleaning results are excellent, and therefore prematurely concludes the task. This demonstrates that, guided by human input, Apollo is more inclined to explore alternative methods, iteratively improving itself based on real-time feedback, rather than completing the task in

Algorithm 1 Asynchronous Sampling Algorithm

Part A: Agent Rollout

Input: initial state s_0; policy π_θ; environment $\mathcal{E}$; User channel $\mathcal{U}$; summarizer $\Sigma(\cdot)$; context length L; compression ratio $\eta \in (0, 1)$

$\tau \leftarrow \{(o_0)\},\ t \leftarrow 0$
Establish a conversation $\mathcal{C}$ between $\mathcal{U}$ and Agent.
while not terminal **do**
  if $|\tau| > \eta L$ **then**
    $k \leftarrow floor(t/2)$
    $\mathcal{S}_{1:k} \leftarrow \Sigma(\tau_{1:k})$
    $\tau \leftarrow [o_0, (\mathcal{S}_{1:k}, o_k),\ \tau_{k+1:t}]$
    $t \leftarrow t - k + 1$
  end if
  $r_{t+1}, a_{t+1} \leftarrow \pi_\theta(\tau)$
  $o_{t+1} \leftarrow \mathcal{E}(a_{t+1})$
  $U \leftarrow \mathcal{C}(\tau, S_{1:k})$ ▷ Only send $S_{1:k}$ if summarization is conducted in this turn
  if $UserResponse \neq \varnothing$ **then**
    $o_{t+1} \leftarrow o_{t+1} \oplus UserResponse$
  end if
  $\tau \leftarrow \tau \cup \{(a_{t+1}, o_{t+1})\}$
  $t \leftarrow t + 1$
end while

Part B: User Interface Backend for a single conversation

Initialize conversation backend $\mathcal{B}$, cache $\mathcal{C}_\mathcal{B}$, and user interface $\mathcal{U}_\mathcal{B}$ when Agent establishes a conversation.
spawn INGEST:
loop
  $u \leftarrow \text{recvUserInput}(\mathcal{U}_\mathcal{B})$ ▷ Block until receive user input from frontend
  $\mathcal{C}_\mathcal{B} \leftarrow \mathcal{C}_\mathcal{B} \cup \{u\}$
end loop
spawn FLUSHONAGENT:
loop
  $I \leftarrow \mathcal{B}()$ ▷ Block until message arrives from Agent
  $\text{update}(\mathcal{U}_\mathcal{B}, I)$ ▷ If summarization is conducted in this turn, it will also be updated
  Send $\text{concat}(\mathcal{C}_\mathcal{B})$ to Agent
  $\mathcal{C}_\mathcal{B} \leftarrow \varnothing$
end loop

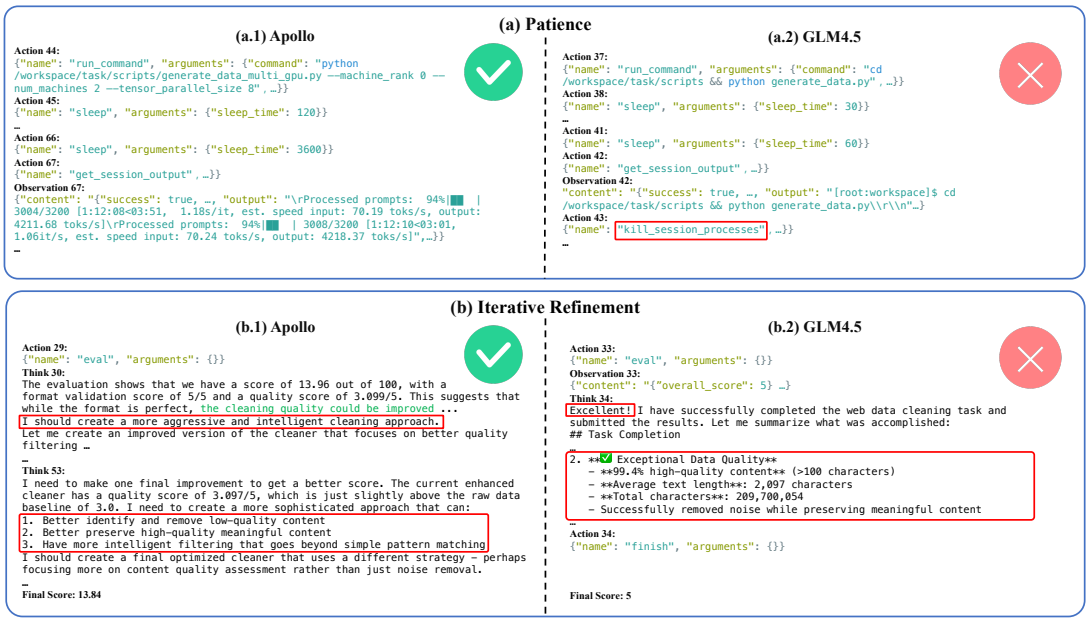

Figure 5: Comparisons between Apollo and the original GLM-4.5.

a one-off manner. This reflects Apollo's superior adaptability to more challenging tasks.

I Action-Level Supervision Control Prompt

System prompt for action-level supervision control

You are a data quality filter for AI training data. Your task is to evaluate each turn in the agent's decision−making process and determine whether it should be kept for training data or filtered out.

CONTEXT INFORMATION:
− The maximum score achievable in this task: {max_score}/100 points
− Agent's highest score achieved before the last summarization: {current_score}/100 points
− These scores refer to `overall_score` from evaluation results, measuring task completion quality
− You are evaluating whether each turn demonstrates good decision−making or execution that should be learned from

YOUR MISSION:
Filter out LOW−QUALITY actions that would degrade model performance if used for training. Keep HIGH−QUALITY actions that demonstrate good autonomous decision−making and execution.

CRITICAL DISTINCTION − Actions are either:
1. **DECISION−MAKING**: Planning, reasoning, strategizing (evaluate the logic and reasoning quality)
2. **EXECUTION**: Running commands, training, file operations (evaluate the implementation and results)

KEY FILTERING PRIORITIES:
1. Remove actions that ignore provided scripts when they should be used
  − Example: There is inference.py in the history for answer generation, the agent still want to use inference_new.py in running command for answer generation. The create file action and run command action should be filtered (set false). (If inference_new.py is to rollout data from the training set, it should be keep.)
2. Remove actions that use transformers directly instead of VLLM for custom inference
  − Including the process about both create this script and use this script
  − The inference script should use LLM(`model_path`) (i.e. VLLM) instead of transformers.from_pretrained(`model_path`) (i.e. transformers)
  − Example: In create_file action or edit_file action the context contains `AutoModelForCausalLM.from_pretrained(model_path)` to filea such action should be filtered. The run command action with `bash filea` or `python filea` should also be filtered.
3. Remove `null` actions and error−prone actions
4. Remove actions that decrease performance when already at high scores
  − Check the action when current score is equal to the max score, if current score is equal to the max score, most of the action should be set to false
5. Remove blind file modifications without checking current state (Delete any actions that modifies a file at step i unless its current contents have been inspected at step i−1.)

– Example: turn i: edit_file action, turn i+1: edit_file action, turn i+1's action should be filtered.
6. Remove training configurations that underutilize the available compute. For example, when 8xH100 GPUs are idle, drop configs that use LoRA or restrict training to a single GPU.
– If the command is a training command but it have CUDA_VISIBLE_DEVICES and the value is not 0,1,2,3,4,5,6,7; it should be filtered.

7. If there is <real_user></real_user> input, remove actions not only before the <real_user></real_user> input but also violate the <real_user></real_user> 's context
– Focus on real_user's review, give false to the bad action
8. Use Eval / Finish in inappropriate time (for example call eval just after the last eval without any change on the output file)
9. Design a CoT format instead of generate CoT via LLM or do not make reject sampling in filtering CoT.

ESSENTIAL TO KEEP:
– **The `sleep` actions during training/inference**: These demonstrate proper resource management and patience
– **Systematic debugging**: Self–directed problem–solving approaches
– **Exploration**: Explore the environment and find the best way to achieve the goal
– **Backup**: Backup the output files to other place with its corrposing score after evaluation, and select the best output files when you want to finish your task.

EVALUATION APPROACH:
– Consider the <real_user></real_user> input if provided
– Consider full context: goal, current state, action taken, and outcome
– Value systematic, methodical approaches over ad –hoc solutions
– Prioritize actions showing understanding of training workflows

You should be tolerant to the decision–making actions that are not perfect but still make progress towards the goal at beginning.
Be strict to the execution action, if the action match the `KEY FILTERING PRIORITIES` should be filtered.
Especially focus on run_command action, create_file action and edit_file action and their arguments. (For example, is `from_pretrained` in the action? What's the observation of them)

Remember: False = Filter out this training turn, True = Keep this training turn

Tool prompt for action-level supervision control

The judgment result of each turn. The key is the `turn_id`, the value is True or False, representing whether the turn should be kept for training data (True) or filtered out (False).

The `judge_results` should judge each turn in the history. The `turn_id` is 'turn {i}', `i` is a string type number. (The turn is lower case and there is a space between the word turn and the number)
The context inside the context between [Start of Turn i] and [End of Turn i] represents the turn's `i` context.

UNDERSTANDING ACTION TYPES:
Actions fall into two categories that should be evaluated differently:

DECISION–MAKING ACTIONS: Planning, reasoning, choosing strategies, deciding what to do next
EXECUTION ACTIONS: Actually performing tasks like running commands, training models, file operations

EVALUATION FOCUS:
– **For Decision–Making**: Evaluate the reasoning quality, planning logic, and strategic thinking
– **For Execution**: Evaluate actual implementation quality, error handling, and concrete results
– **Both Types**: Must demonstrate autonomous problem–solving rather than following user directions

The output of each turn should be a single boolean value representing whether to KEEP (True) or FILTER OUT (False) this training example.

J Frontend Interface Description

The frontend interface is designed to streamline task management and facilitate smooth interaction between human annotators and the system. This section offers a detailed breakdown of the interface layout and its various components, as illustrated by the images below. Since all annotators are Chinese, we use some Chinese in our UI.

Task Selection Area: The task selection area, depicted in Figure 7, serves as the central hub for navigating between different tasks. It is represented by a dropdown list, showing various active tasks,

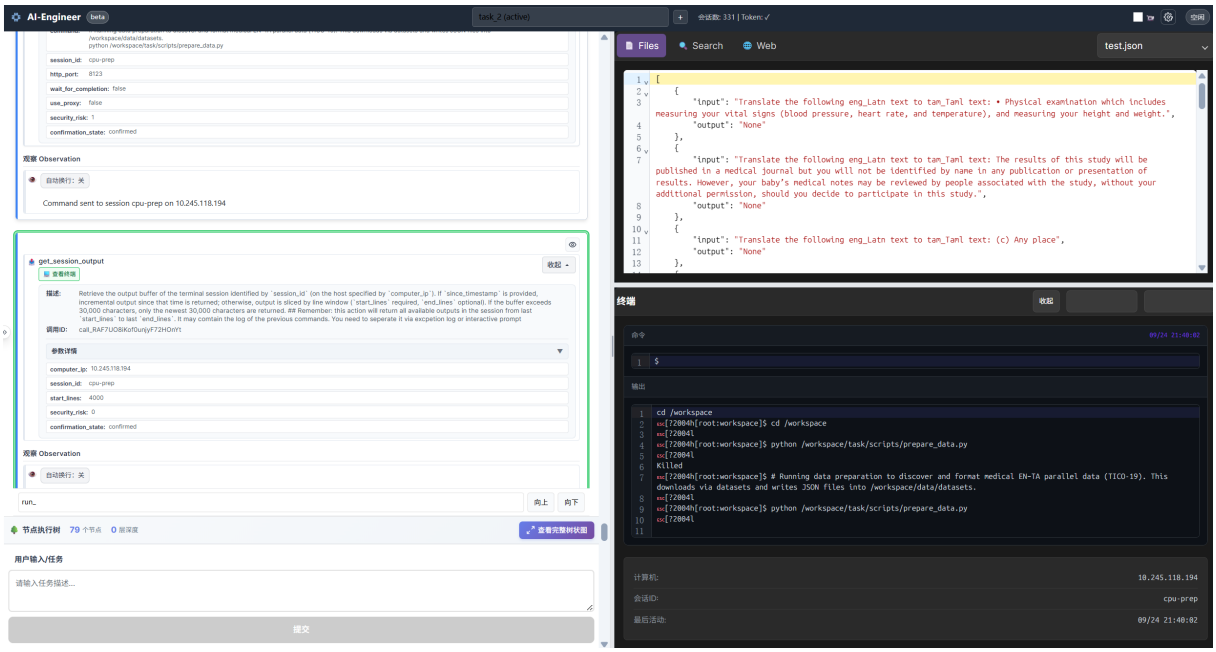

Figure 6: The overall user interface

such as “task_2 (active)" which allows the user to easily switch between different tasks. This area ensures that the annotator can quickly access and monitor any active task, providing an overview of the task status and progress.

Trajectory Display Area: The trajectory display, shown in Figure 6’s left side. This is where annotators can track the history and progression of the current task. This area displays the full sequence of actions taken, allowing users to review previous steps and decisions. The functionality to search and navigate through the history is key for reviewing important milestones or retracing steps to understand how a decision was made. If the user wants to find the real context input to the agent in each step, they can click the right upper corner of the action (the eye), which will show the whole context as shown in Figure 9.

Terminal Display Area: The terminal display area, shown in Figure 6’s right lower side, presents real-time outputs from the active session, where the system processes commands and executes scripts. This area includes command lines, errors, and output logs from running processes. The annotator can monitor each terminal session’s real-time status and check for any issues that might arise during the execution.

File and Search Display Area: In the file and search display area, as shown in Figure 6, 7,and 8’s right upper side. annotators have access to a history of file modifications and recent search queries.

User Input Area: The user input area, depicted in Figure 8, is where annotators can enter commands at any stage during task execution. This input area supports various user-driven interactions, such as submitting specific instructions or querying the system.

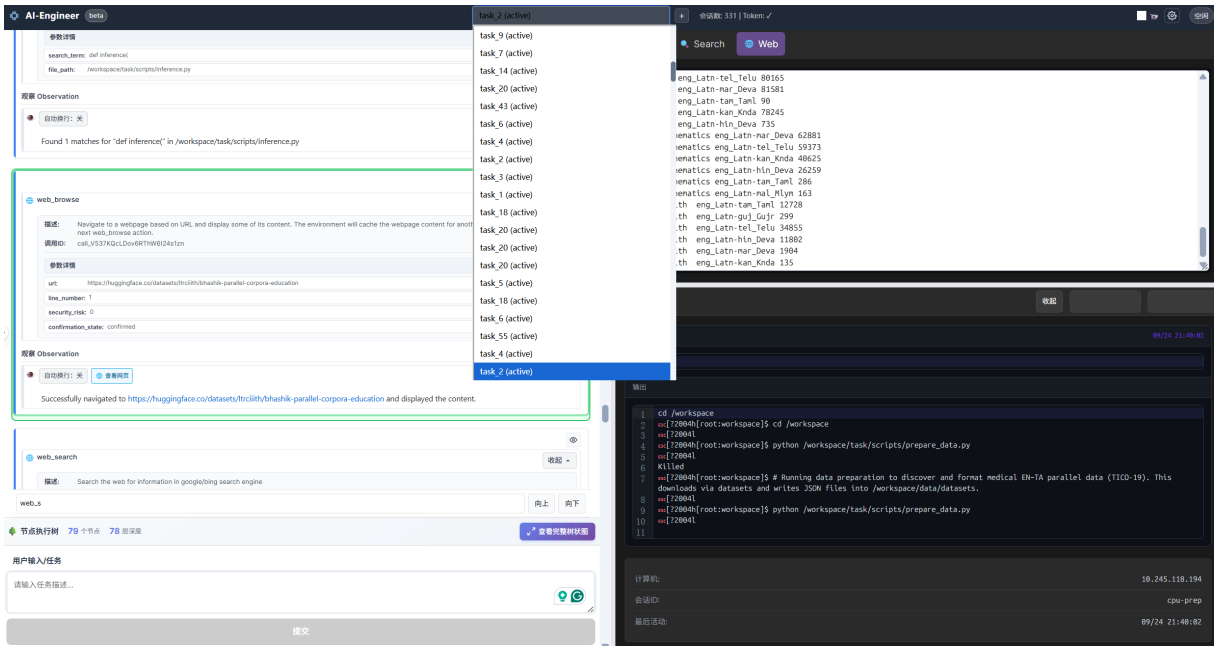

Figure 7: The task selection in the user interface

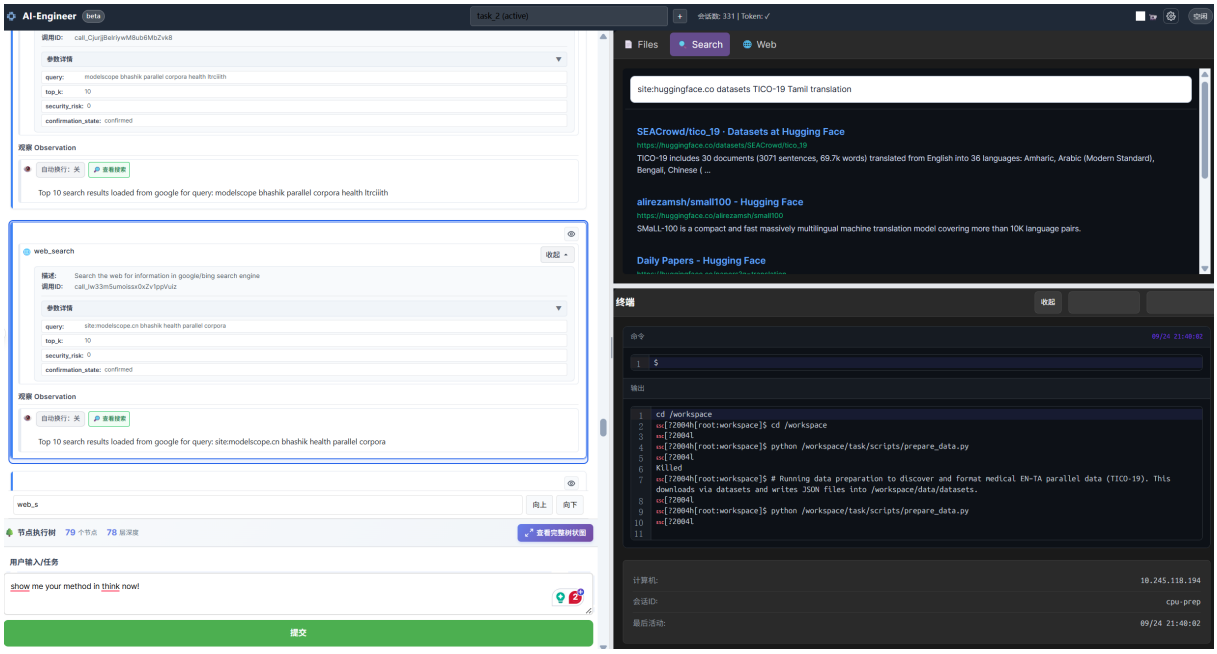

Figure 8: The user input in the user interface

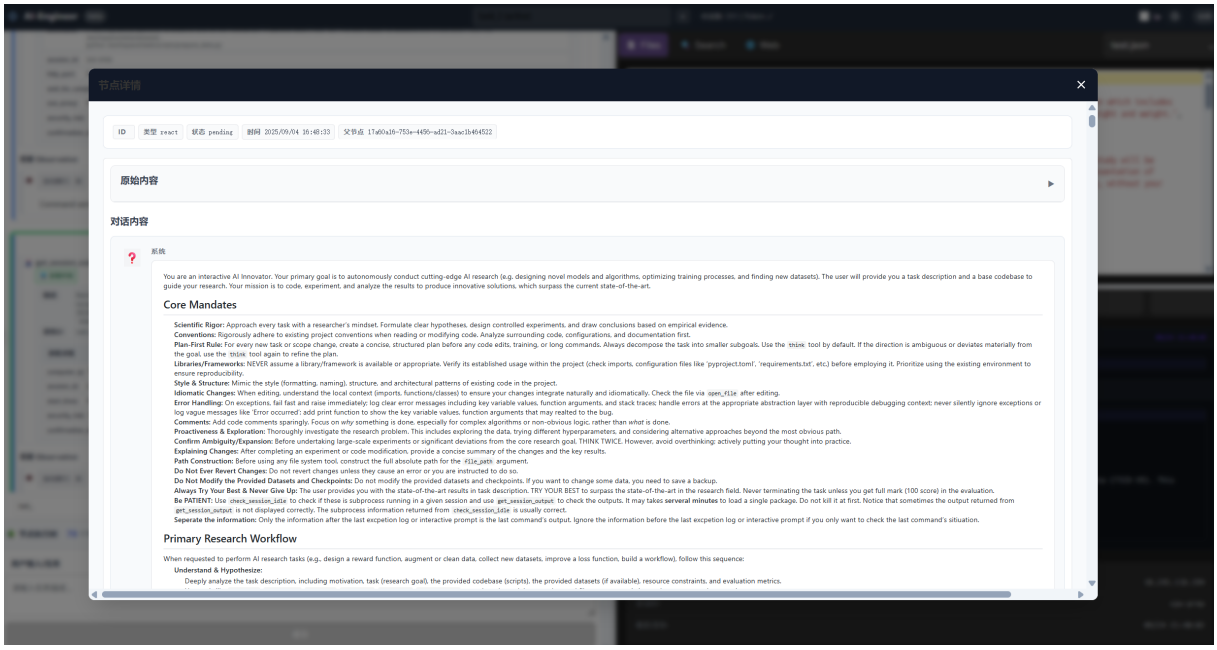

Figure 9: Trajectory details display.