OrchDAG: Complex Tool Orchestration in Multi-Turn Interactions with Plan DAGs

Yifu Lu*
Princeton University
yiful@princeton.edu

Shengjie Liu*
Amazon
zycjlsj@amazon.com

Li Dong
Amazon
ldonga@amazon.com

Abstract

Agentic tool use has gained traction with the rise of agentic tool calling, yet most existing work overlooks the complexity of multi-turn tool interactions. We introduce OrchDAG, a synthetic data generation pipeline that models tool execution as directed acyclic graphs (DAGs) with controllable complexity. Using this dataset, we benchmark model performance and propose a graph-based reward to enhance RLVR training. Experiments show that the dataset presents a challenging but solvable benchmark, and the proposed reward is effective when combined with GRPO-style algorithms, highlighting the importance of leveraging topological structure and data complexity in multi-turn tool use.

1 INTRODUCTION

Large Language Models (LLMs) ([3, 5, 6, 26, 36]) have been at the forefront of advancing artificial intelligence, marking significant breakthroughs in diverse fields. The planning capabilities of LLMs, particularly their ability to use tools ([33, 34]), enable them not only to execute commands and perform web searches but also to enhance their advanced mathematical reasoning abilities. LLM Compiler [12] and its subsequent work ([7, 8]) propose constructing tooling usage as a directed acyclic graph (DAG) to enable the parallel execution of independent tools, thereby improving tool-calling efficiency. CodeAct [28] and CodePlan [29] propose leveraging the generation of pseudo-Python code to outline high-level reasoning processes for complex multi-step reasoning tasks, where each tool usage is represented as a function call within the code. ReWOO [30] proposes a modular framework that decouples the reasoning process from the external observations of each tool usage, thereby reducing token consumption and improving efficiency. [15] clusters the provided tools into groups of toolkits, plans at the toolkit level, and replans by selecting tools within the same toolkit if error comes out. [17] proposes a method called Predictive-Decoding, which leverages Model Predictive Control from the optimal control field to mitigate early errors in planning and promote non-myopic planning, thereby enhancing overall accuracy. ReasonFlux [31] proposes a framework in which the LLM reasons over template fields, executes tools based on the templates, and employs reinforcement learning to improve planning accuracy using an action completion reward.

In the agentic setting, LLMs are evolving beyond purely textual reasoning toward dynamic agents capable of planning, tool use, and multi-step (also multi-turn) execution. The introduction of Group Relative Policy Optimization (GRPO) [22] further inspired the development of Reinforcement

*Work is done during internship in Amazon. Equal contribution.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Multi-Turn Interactions in Large Language Models.

Learning with Verifiable Reward (RLVR) for agentic tool use, driven by its efficiency. The xLAM [37] suite introduces purpose-built "large action models" optimized for function calling, offering strong baselines and open resources for multi-turn tool execution. Llama-Nemotron [2] extends this trajectory with efficient reasoning modes and scalable inference, enabling models to dynamically switch behaviors across long conversations. ToolRL [19] systematically studies reinforcement learning reward designs—covering granularity, temporal structure, and signal types—to improve generalization in multi-turn tool-integrated reasoning, while OTC [27] complements this by explicitly balancing accuracy and tool-call efficiency to maintain productivity over prolonged interactions. Kimi K2 [25] shows that stabilizing long-context training and using multi-stage RL leads to robust performance across multi-round software engineering, math, and agentic tasks. These works highlight that advancing agentic LLMs in multi-turn settings requires not only larger or more efficient models, but also principled reward design, cost-aware tool-use strategies, and scalable system pipelines.

Recent advances in evaluating agentic models have led to new benchmarks and environments for assessing performance in realistic, interactive scenarios. ACEBench [4] overcomes limits of prior evaluations by introducing a structured benchmark with Normal, Special, and Agent categories to test atomic-level tool use across simple, complex, multi-agent, and ambiguous instruction settings. Complementing this, BFCL (v3) [18] standardizes function-calling benchmarks across real-world contexts, supporting serial and parallel invocations in multiple languages through an AST-based evaluation. τ-Bench, τ^2-Bench, and UserBench [32, 1, 20] together extend evaluation from structured tool–user interactions to controlled bidirectional agent cooperation and finally to fully user-centric, dynamic environments, progressively enriching the realism and robustness of agent assessment.

Prior work on tool use has mainly studied general real-world APIs [21], such as *send email* or *make calendar*, along with related functions in *web search* systems like Manus. Meanwhile, current multi-turn settings mainly focus on computer-use tasks [18], such as manipulating files in the operating system. In industrial settings, however, an agent may need to work with hundreds of domain tools, including APIs and pipeline endpoints, and may also interact with other domain agents to produce a complete answer to a user query. The complexity usually arises from three aspects: (1) the dependencies among tools can be intricate, (2) the output of a tool is often represented as a JSON file with many fields, and (3) a key output field from one tool may serve as an input to another, but with different field names. Moreover, in multi-turn settings, the environment may execute the required tools but encounter time-outs or runtime errors in their responses. Therefore, it is important to construct a dataset that not only evaluates current models but also pushes their capabilities in complex multi-turn tool interaction, which is essential for building robust and reliable agents.

In summary, this work makes several pivotal contributions:

- We design a synthetic multi-turn data generation pipeline OrchDAG for agentic tool use, where each round of tool execution for a user query is represented as a DAG. The complexity of the generated data is controlled by a pipeline hyperparameter.
- Using the constructed dataset, we first evaluate the current model's performance and then introduce a graph-based reward derived from the DAG for RLVR training.
- Extensive experiments show the effectiveness of our approach, emphasizing the value of incorporating the topological structure of tool execution graphs and the importance of controlling data complexity in multi-turn tool use.

2 PRELIMINARY

2.1 LLM Reasoning with Tools for Multi-turn Settings

For the first turn, given a user query x and a pretrained LLM $\rho_\theta(\cdot)$, the LLM generates an tool execution plan represented as a graph with $p = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \sim \rho_\theta(p \mid \mathcal{T}, \mathcal{D}, x)$, where p is the plan list after topological sorting, $\mathcal{T}$ is the set of available tools, and $\mathcal{D}$ is the collection of descriptions for all available tools. At each step t, the LLM generates an intermediate reasoning output $r_t \sim \rho_\theta(r_t \mid \mathcal{T}, \mathcal{D}, x, p, \mathcal{O}_1, \ldots, \mathcal{O}_{t-1})$ and executes the plan step $\mathcal{P}_t$ to obtain the observation $\mathcal{O}_t$. The final response is then generated as $\mathcal{R} \sim \rho_\theta(\mathcal{R} \mid \mathcal{T}, \mathcal{D}, x, p, \mathcal{O}_1, \ldots, \mathcal{O}_n)$.

In later turns, a user may issue an irrelevant query requiring a completely new tool execution graph, or a dependent query that builds on partial outputs from earlier tool executions or responses. Additionally,

some tools may return errors (e.g., timeouts), requiring unfinished execution paths from prior queries to be rescheduled.

2.2 Tool Execution as DAG

Given a plan p generated by the LLM, we represent it as a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = (v_1, \ldots, v_n)$ is the set of nodes and $\mathcal{E} = (e_1, \ldots, e_m)$ is the set of edges. The node v_1 corresponds to the user query, and v_n represents the final node that aggregates observations and returns the response. The intermediate nodes $v_2, \ldots, v_{n-1}$ correspond to tool calls, each associated with an attribute that stores its tool payload in JSON format. An edge $e_i \in \mathcal{E}$ denotes a dependency between two tools, where an output key from the source tool serves as an input key to the target tool.

We represent the tool graph as an ordered list of tasks in a JSON-like text style. Each task contains four fields : `task_id`, `toolname`, `payload`, and `dependencies`. A task can be associated with multiple dependencies. A task can be expressed as {`task_id`: `task_4`, `toolname`: `name`, `payload`: {`param1`: `val1`, `param2`: `$2.outputkey1`, `param3`: `$3.outputkey4`}, `dependencies`: [`task_2`, `task_3`]}.

3 METHODOLOGY

In real-world domains, API specifications and orchestrations are often considerably more complex, making it challenging for LLMs pretrained on general public data to generate accurate and reliable plans for diverse user queries. Drawing inspiration from LLMCompiler [12, 8], for queries involving complex tool interactions, it is advantageous to first construct a tool execution DAG. This DAG serves as a blueprint for executing tools sequentially or in parallel, with subsequent replanning guided by both the execution results of the current DAG and any new user queries in later turns.

3.1 OrchDAG – Synthetic Data Generation Pipeline for Multi-Turn Tool Use

As discussed in Section 1, the design of the data generation pipeline follows several key principles to better reflect real-world tool-use scenarios: (1) the complexity of the tool execution DAG for each synthetic user query should be controllable through pipeline hyperparameters, (2) the system prompt provided to the LLM must include irrelevant tools (both schema and description) so that the model learns to identify and select only the relevant ones, (3) the output payload of each tool should contain multiple fields, typically four or five, and (4) at least one key output field from a tool should serve as an input to another tool, but with a different field name, to capture schema misalignment commonly observed in practice.

Moreover, in multi-turn settings, the data should capture scenarios where certain nodes in the tool execution DAG fail. When the user issues a follow-up query, the corresponding DAG in the dataset should exclude nodes that have already been executed in the previous turn, while reusing their available results whenever applicable. In light of these requirements, we develop a graph-based data generation pipeline implemented with LangGraph [2], accompanied by a set of validation functions to ensure the quality of the generated data points.

As shown in Figure 1, the data generation pipeline begins by collecting real high-quality tools from existing benchmarks, Specifically, we leverage APIGEN [16] and TOOLACE [14]. To ensure sufficient complexity, we retain only examples where the final answer involves more than two distinct functions represented in JSON format. Notably, we extract tools directly from the answers rather than from the system-prompt tool lists. This design choice eliminates the need for additional categorization or clustering steps, which could introduce unnecessary uncertainty, while naturally yielding a smaller and more coherent set of related tools. After filtering, APIGEN contributes 2,542 data points and TOOLACE contributes 1,005.

For each data point, suppose it contains four real tools; these are placed as the first layer of the tool execution DAG. Based on the hyperparameters of the DAG (height and width), we then randomly sample a topological order to obtain a DAG template (see Figure 1). According to this template, we synthesize the tools for each node layer by layer under the following conditions: (1) each input key must depend on an output key from one of its parent nodes in the DAG, and (2) the field names

[2] https://langchain-ai.github.io/langgraph/

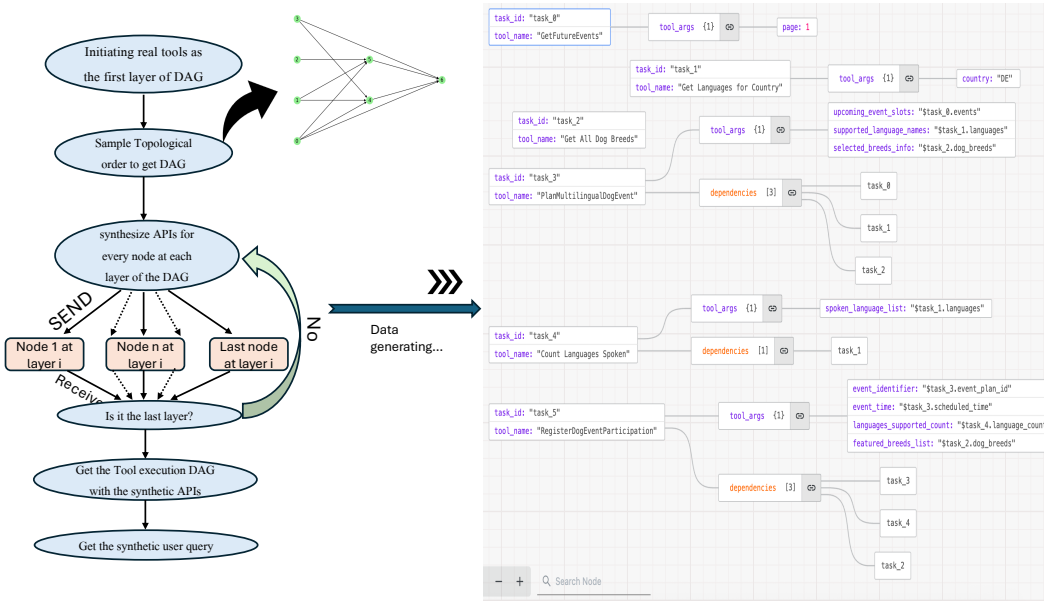

Figure 1: Data Generation Pipeline for a single turn

should not remain identical across instances but instead vary randomly. After populating the DAG template, we obtain the tool execution DAG, which is then used to prompt the LLM to generate the corresponding user query, conditioned on the DAG and a few-shot set of examples. Finally, we augment the system prompt with irrelevant tools to encourage the model to discriminate among available options.

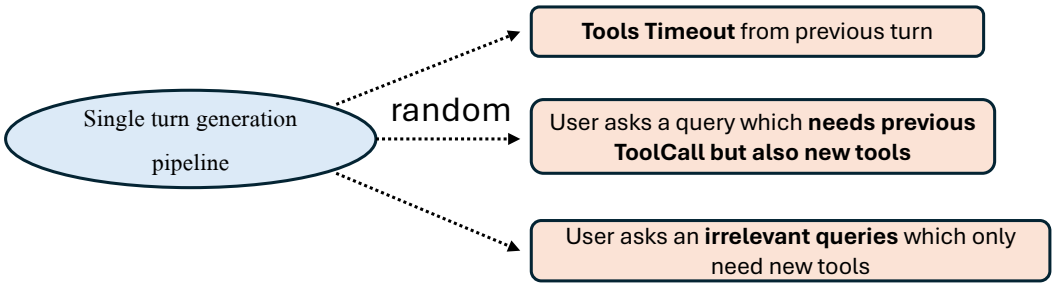

Figure 2: Extension of single-turn data generation pipeline to multi-turn settings

We further extend the generation pipeline to the multi-turn setting by attaching three additional nodes to the final node (see Figure 2), with only one being activated during generation. These nodes correspond to three possible multi-turn scenarios : (1) the user issues a completely irrelevant query, (2) the user poses a query that requires a new set of tools while also depending on the previous final response or intermediate tool outputs, and (3) a tool execution error occurs, such as a timeout. In case (1), the outcome is a completely new DAG; in case (2), the new DAG must explicitly encode cross-turn dependencies through the task identifiers from the previous DAG; and in case (3), the resulting DAG reduces to a partial subgraph of the original DAG. We adopt the final data generation format introduced in ToolRL [19]. An example of a generated data sample is shown in Figure 3.1.

The quality of the data, particularly the synthetic data, is critical. To ensure reliability, we incorporate a rule-based verification mechanism into the generation pipeline. All tools, plan DAGs, tool calls, and observations produced by the LLM are required to be in JSON format. The first verification layer checks JSON validity. For plan DAGs, we apply AST [18] matching at each node to guarantee that the LLM only references the provided tools with the correct argument names. We further validate

symbolic arguments by comparing each referenced key against the JSON schema of the predecessor's output. Every tool call is verified against the plan DAG and the preceding tool call observations, ensuring adherence to the plan and the correct use of return values as inputs for dependent calls. Likewise, each observation is matched to its corresponding tool call, and its return value is checked against the tool's JSON schema. If any verification step fails during generation, the LLM is required to restart the process.

Our pipeline enables the generation of diverse synthetic queries derived from high-quality real APIs, each requiring resolution through a plan DAG. The difficulty of the queries is controlled by the topological structure of randomly generated DAG templates. Since these templates vary in structure, a fixed workflow for synthetic data generation is impractical. Instead, the graph-based pipeline provides a flexible end-to-end framework for producing such data. Finally, the rule-based verification mechanism ensures reliability: it not only checks JSON validity to guarantee compatibility with downstream benchmarks, but also leverages AST matching to validate the correctness of DAG instantiations during data generation.

Synthetic Data Sample

System Prompt:
You are a dialogue assistant designed to leverage tool calls to solve user tasks and provide structured responses.

Available Tools
In your response, you can use the following tools: {{Tool List}}

Steps for each turn
1. Think: Retrieve the relevant context and evaluate the current tool.
2. DAG: Produce a task list defined here 2.2
3. Respond: If a response is needed, generate one while maintaining consistency across user queries.

Synethetic User Query: ...

</think> . . . </think> [The think block is absent in the synthetic data but included during the training stage.]

</DAG> real DAG generated from pipeline </DAG>

<tool_call> tool call 1st DAG layer </tool_call>
<obs> observation 1st DAG layer </obs>
...
<tool_call> tool call last DAG layer </tool_call>
<obs> observation last DAG layer </obs>
</response> ... </response>

New User Query: ...

</think> . . . </think>

</DAG> new DAG based on the three scenarios defined in the multi-turn settings 3.1.
</DAG>

...

3.2 OrchDAG – Graph-based Reward Derived from the DAG for RLVR training

Due to the intricate tool interaction structure inherent in the synthetic data, the format reward, correctness reward, and parameter matching reward defined in [19] may remain sparse, even when initiating

a large number of rollouts. Moreover, this reward does not account for structural dependencies among tools; thus, no reward is given when the LLM correctly predicts partial dependencies for these tools.

To account for structural dependencies, and given that we have access to the ground-truth DAG during synthetic data generation, following [13], we use a weighted Graph Edit Distance (GED) as the reward signal at each turn. GED [10] measures the distance between two graphs by applying operations such as edge deletion, edge insertion, node insertion, or node relabeling to transform one graph into an isomorphic form of the other.

We define the reward for each turn as

$$\mathrm{R_{Total}} = \mathrm{R_{Format}} + \alpha \mathrm{R_{DAG}}, \quad \text{where} \quad \mathrm{R_{DAG}} = 1 - \frac{\mathrm{GED}(\mathrm{g}_1, \mathrm{g}_2)}{\mathrm{GED}(\mathrm{g}_1, \emptyset) + \mathrm{GED}(\mathrm{g}_2, \emptyset)}$$

Here g_1 is the predicted DAG, g_2 is the ground-truth DAG. We define the following node equivalence when calculating GED: the tool name, parameter names, and parameter values are treated as a single unit, and equivalence is evaluated at the level of the entire tool call. The reward $\mathrm{R_{Format}}$ is assigned a value of 1 if the output contains the special tokens in the correct order, and 0 otherwise. α is the hyperparameter used to balance the two types of rewards. In this reward design, we not only provide credit to LLMs for partially correcting the path during rollouts, but also make the rewards denser compared to the previous design. The multi-turn setting is naturally supported, as the ground-truth DAG is available at each turn.

4 EXPERIMENTS

In this section, we describe the generated dataset, focusing on the distribution of topological difficulty in the DAG templates and the proportion of single-turn versus multi-turn settings. For simplicity, we restrict evaluation to the two-turn setting, leaving extensions to longer horizons for future work. To ensure independence between training and test data, we construct them from two disjoint sets of data points from APIGEN [16] and TOOLACE [14]., as described in 3.1.

Table 1: Synthetic Data Distribution (Height and width are hyperparameters controlling DAG complexity, and success rate is the proportion of data that passes rule-based validation)

Type	data #	Multi-turn proportion	Average Height	Average Width	Sucessful Rate
Training	1800	30%	2.50 ± 0.12	3.4 ± 0.24	0.6
Test	250	25%	2.7 ± 0.08	3.1 ± 0.14	0.7

4.1 Task Difficulty

We investigate two central questions: (1) Given the designed difficulty of our synthetic data, is the dataset solvable in principle? A dataset that cannot be solved even by advanced closed-source models such as Claude 4 or GPT-4o would lack practical utility. (2) If it is solvable, does the dataset offer a sufficient level of challenge to meaningfully evaluate model performance? To begin with, we evaluate several closed-source and open-source models by providing them with the system prompt defined in Section 3.1, and measure whether they can correctly predict the DAG by analyzing tool dependencies. In the multi-turn setting, models must generate the DAG by considering both the available tools and the observations from previous turns. We use Accuracy (pass@1) to evaluate performance, as our focus here is on assessing the task difficulty introduced by the dataset.

For completeness, we also report Qwen3 pass@64 accuracy: 20.23% for Qwen3-4B and 26.55% for Qwen3-8B. From Table 2, we can see that GPT-4o maintains the highest accuracy for the zero-shot setting and the three-shots setting with nearly same performance with Claude 4 in the one-shot setting. The accuracy for GPT-4o and also for Claude 4 shows that our dataset is solvable however the perofmrance for Claude 3.5, Qwen2.5 3B with accuracy 0, and Qwen 2.5 7B demonstrates the challenge of our datasets to current LLMs. Comparison between the one-shot and three-shot results shows that providing additional examples does not necessarily improve LLM performance in DAG prediction.

Table 2: Pass@1 Accuracy for predicting the DAG for single/multi-turn settings (All experiments were run 10 times with the temperature of the base LLMs set to 0.1)

Models	Zero Shot	One shot	Three shots
GPT-4o	**(0.18 ± 0.03)**	(0.22 ± 0.02)	**(0.24 ± 0.04)**
Claude 4	(0.15 ± 0.01)	**(0.23 ± 0.03)**	(0.22 ± 0.01)
Claude 3.7	(0.16 ± 0.04)	(0.18 ± 0.03)	(0.23 ± 0.01)
Claude 3.5	(0.08 ± 0.02)	(0.09 ± 0.03)	(0.08 ± 0.03)
DeepSeek-R1	(0.12 ± 0.02)	(0.14 ± 0.01)	(0.11 ± 0.04)
Qwen2.5 3B	(0 ± 0)	(0 ± 0)	(0.02 ± 0.01)
Qwen2.5 7B	(0.02 ± 0.01)	(0.03 ± 0.02)	(0.03 ± 0.02)

4.2 Analysis of the Graph-based Reward Shaping in OrchDAG

As discussed in Section 3.2, the reward signal from ToolRL [19] can be sparse in our data., which makes it difficult for reinforcement learning algorithms such as GRPO [22], DAPO [35], and GiGPO [9] to efficiently improve the policy LLM, even with large rollouts. We apply ToolRL on the training single-turn dataset and evaluate it on the test set in the single-turn setting. In this setup, we train Qwen2.5 with GRPO using ToolRL, and convert the ToolCalls it generates into predicted DAGs, since the outputs follow a standardized JSON format. We use 8×100 A100 GPUs with Verl [23] to complete the training. We evaluate performance using two metrics: Accuracy/step and Accuracy/user_query. Accuracy/step measures correctness at the step level, where each individual action in a turn is assessed independently; a step may be correct even if the final tool execution graph is incorrect. Accuracy/user_query measures correctness at the full query level, requiring the entire tool execution graph to be correct.

Table 3: ToolRL Performance on OrchDAG Single-turn Test Dataset (The definitions of fine-grained and coarse rewards are given in ToolRL

Model (Qwen2.5)	Acc/step	Acc/user_query
3B Coarse	0.517	0
3B Finegrained	0.540	0
7B Coarse	**0.609**	0
7B Finegrained	0.594	0

From Table 3, we observe that ToolRl performs reasonably well on certain steps within a single turn; however, it struggles to maintain a coherent overview of the entire execution. In contrast, as shown in Table 2, Qwen2.5-7B achieves 2% accuracy in predicting the DAG. This indicates that for complex tool executions, it may be advantageous to first establish a high-level plan, such as a DAG, to guide the subsequent execution. We subsequently fine-tune Qwen2.5 using GRPO on the training single-turn dataset, guided by the proposed graph-based reward. We evaluate different hyperparameter settings: the use of entropy regularization and the KL loss, the choice of rollout number, and the number of training steps for the optimizer.

Table 4: Performance of Graph-Based Reward on OrchDAG Single-turn Test Dataset (We report results using Acc/user_query as the evaluation metric. The columns indicate the number of training steps, and n denotes the rollout number.)

Model (Qwen2.5) / Steps	15	30	45	60
3B KL n=4	0	-	-	-
7B n=4	0.184	0.253	0.241	-
7B KL n=4	0.184	0.276	0.276	-
7B KL Entropy n=4	0.195	0.276	0.253	-
7B KL n=8	**0.23**	**0.33**	**0.402**	**0.391**

In Table 4, we observe that model size has a clear impact on performance. Moreover, the rollout number plays a crucial role, consistent with the intuition that larger rollout numbers enable greater exploration [24], thereby increasing the likelihood of reaching the correct DAG. To evaluate the effectiveness of the GED-based reward design, we conduct an ablation study using a coarser reward:

the predicted DAG receives a reward of 1 if it exactly matches the ground-truth DAG, and 0 otherwise. Using the 7B model with KL and $n = 8$, the accuracy remains 0 even after 15 training steps.

We then extend our experiments to the multi-turn setting. In this setting, we train Qwen2.5 on the entire training set with GRPO, using both the information from previous turns and the new user query as input, and evaluate performance on the full OrchDAG test dataset. Based on Table 4, we report performance only at the 45th training step.

Table 5: Performance of Graph-Based Reward on OrchDAG Single/Multi-turn Test Dataset (We report results using Acc/user_query as the evaluation metric. The columns indicate the three multi-turn scenarios defined by 2)

Model (Qwen2.5) / Steps	scenario 1	scenario 2	scenario 3
7B KL Entropy n=4	0.112	0.125	0.218
7B KL n=8	**0.156**	**0.203**	**0.352**

From Table 5, we observe that in the multi-turn setting, performance decreases across both experimental setups. The largest drops occur in Scenario 1 (tool-calling error) and Scenario 2 (requiring information from the previous turn), since these tasks depend not only on the new user query and the system prompt but also on information carried over from earlier turns. In contrast, the drop in Scenario 3 is smaller, as the new user query is completely independent of prior turns. To demonstrate generalizability, we further evaluate the trained model on StableToolBench [11], measuring solvable pass rates across L1, L2, and L3 categories. A task is considered successful when the predicted DAG matches the ground truth. We choose StableToolBench for its inherent complexity in tool interactions. In StableToolBench, GPT-4-0613 (CoT) achieves solvable pass rates of 45.5 (L1 instruction), 57.4 (L1 category), 48.8 (L1 tool), 43.0 (L2 instruction), 46.5 (L2 category), and 48.1 (L3 instruction). Under the same evaluation, our model attains 47.1, 56.4, 47.2, 41.3, 44.8, and 50.7, respectively.

4.3 Training Insight Analysis

As shown in Table 4, performance generally improves as the number of training steps increases. However, when we extend training beyond this range, we observe a significant drop in performance around step 51. Inspired by DAPO [35], We hypothesize that the performance drop may be caused by a low-entropy situation, where the model becomes overly confident and thus fails to explore sufficiently.

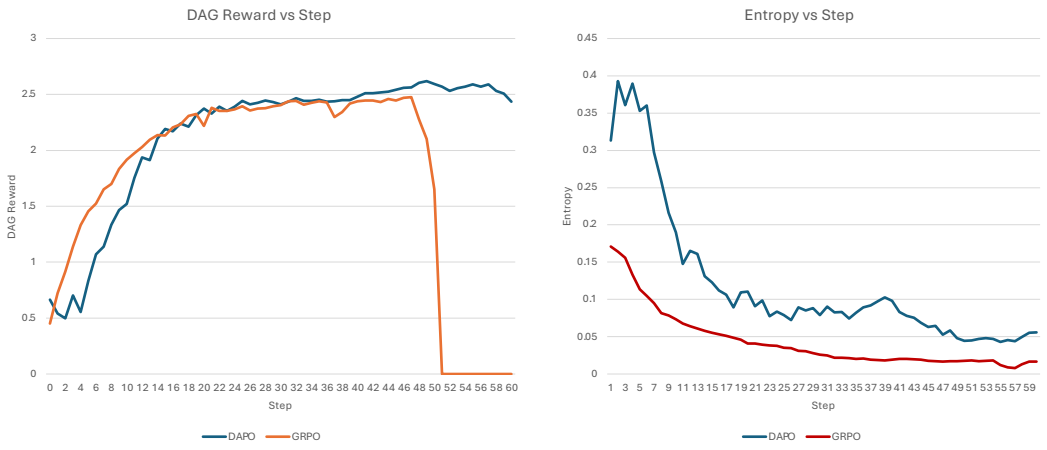

Figure 3: Performance and Entropy Analysis with DAPO and GRPO

After applying DAPO, we observe that the performance collapse no longer occurs, and the entropy remains at a relatively higher level in the later stages of training.

5 CONCLUSION

In summary, we introduce OrchDAG, a synthetic multi-turn data generation pipeline that models tool execution as DAGs with controllable complexity. Leveraging this dataset, we evaluate model performance and propose a graph-based reward to enhance RLVR training. Extensive experiments validate the effectiveness of our approach, underscoring the importance of exploiting the topological structure of tool execution graphs and managing data complexity in multi-turn tool use.

Nonetheless, our method remains limited in that it does not yet address multi-turn scenarios involving implicit dependencies, such as file operations in computer-use tasks. In future work, we aim to extend our framework to capture these implicit dependency cases.

References

[1] Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment, 2025.

[2] Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, Gerald Shen, Ameya Sunil Mahabaleshwarkar, Bilal Kartal, Yoshi Suhara, Olivier Delalleau, Zijia Chen, Zhilin Wang, David Mosallanezhad, Adi Renduchintala, Haifeng Qian, Dima Rekesh, Fei Jia, Somshubra Majumdar, Vahid Noroozi, Wasi Uddin Ahmad, Sean Narenthiran, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Siddhartha Jain, Igor Gitman, Ivan Moshkov, Wei Du, Shubham Toshniwal, George Armstrong, Branislav Kisacanin, Matvei Novikov, Daria Gitman, Evelina Bakhturina, Prasoon Varshney, Makesh Narsimhan, Jane Polak Scowcroft, John Kamalu, Dan Su, Kezhi Kong, Markus Kliegl, Rabeeh Karimi, Ying Lin, Sanjeev Satheesh, Jupinder Parmar, Pritam Gundecha, Brandon Norick, Joseph Jennings, Shrimai Prabhumoye, Syeda Nahida Akter, Mostofa Patwary, Abhinav Khattar, Deepak Narayanan, Roger Waleffe, Jimmy Zhang, Bor-Yiing Su, Guyue Huang, Terry Kong, Parth Chadha, Sahil Jain, Christine Harvey, Elad Segal, Jining Huang, Sergey Kashirsky, Robert McQueen, Izzy Putterman, George Lam, Arun Venkatesan, Sherry Wu, Vinh Nguyen, Manoj Kilaru, Andrew Wang, Anna Warno, Abhilash Somasamudramath, Sandip Bhaskar, Maka Dong, Nave Assaf, Shahar Mor, Omer Ullman Argov, Scot Junkin, Oleksandr Romanenko, Pedro Larroy, Monika Katariya, Marco Rovinelli, Viji Balas, Nicholas Edelman, Anahita Bhiwandiwalla, Muthu Subramaniam, Smita Ithape, Karthik Ramamoorthy, Yuting Wu, Suguna Varshini Velury, Omri Almog, Joyjit Daw, Denys Fridman, Erick Galinkin, Michael Evans, Shaona Ghosh, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, Chris Alexiuk, Joey Conway, Trisha Saar, Ann Guan, Krzysztof Pawelec, Shyamala Prayaga, Oleksii Kuchaiev, Boris Ginsburg, Oluwatobi Olabiyi, Kari Briski, Jonathan Cohen, Bryan Catanzaro, Jonah Alben, Yonatan Geifman, and Eric Chung. Llama-nemotron: Efficient reasoning models, 2025.

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[4] Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. Acebench: Who wins the match point in tool usage?, 2025.

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[6] DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang,

Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Shengfeng Ye, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanhong Xu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025.

[7] Lutfi Eren Erdogan, Nicholas Lee, Siddharth Jha, Sehoon Kim, Ryan Tabrizi, Suhong Moon, Coleman Hooper, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Tinyagent: Function calling at the edge, 2024.

[8] Lutfi Eren Erdogan, Nicholas Lee, Sehoon Kim, Suhong Moon, Hiroki Furuta, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks, 2025.

[9] Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training, 2025.

[10] Xinbo Gao, Bing Xiao, Dacheng Tao, and Xuelong Li. A survey of graph edit distance. *Pattern Anal. Appl.*, 13(1):113–129, February 2010.

[11] Zhicheng Guo, Sijie Cheng, Hao Wang, Shihao Liang, Yujia Qin, Peng Li, Zhiyuan Liu, Maosong Sun, and Yang Liu. Stabletoolbench: Towards stable large-scale benchmarking on tool learning of large language models, 2024.

[12] Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W. Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling, 2024.

[13] Shengjie Liu, Alex Lu, Li Dong, Jason Zhu, Manish Gawali, and Alice Zhou. Toposem: In-context planning with semantically-informed tooling graph similarity. 2025.

[14] Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen. Toolace: Winning the points of llm function calling, 2025.

[15] Yanming Liu, Xinyue Peng, Jiannan Cao, Shi Bo, Yuwei Zhang, Xuhong Zhang, Sheng Cheng, Xun Wang, Jianwei Yin, and Tianyu Du. Tool-planner: Task planning with clusters across multiple tools, 2025.

[16] Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets, 2024.

[17] Chang Ma, Haiteng Zhao, Junlei Zhang, Junxian He, and Lingpeng Kong. Non-myopic generation of language models for reasoning and planning, 2024.

[18] Shishir G. Patil, Huanzhi Mao, Charlie Cheng-Jie Ji, Fanjia Yan, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

[19] Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs, 2025.

[20] Cheng Qian, Zuxin Liu, Akshara Prabhakar, Zhiwei Liu, Jianguo Zhang, Haolin Chen, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. Userbench: An interactive gym environment for user-centric agents, 2025.

[21] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.

[22] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[23] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, page 1279–1297. ACM, March 2025.

[24] Saksham Sahai Srivastava and Vaneet Aggarwal. A technical survey of reinforcement learning techniques for large language models, 2025.

[25] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025.

[26] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023.

[27] Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently, 2025.

[28] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents, 2024.

[29] Jiaxin Wen, Jian Guan, Hongning Wang, Wei Wu, and Minlie Huang. Unlocking reasoning potential in large langauge models by scaling code-form planning, 2024.

[30] Binfeng Xu, Zhiyuan Peng, Bowen Lei, Subhabrata Mukherjee, Yuchen Liu, and Dongkuan Xu. Rewoo: Decoupling reasoning from observations for efficient augmented language models, 2023.

[31] Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates, 2025.

[32] Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains, 2024.

[33] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[34] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models, 2023.

[35] Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Mu Qiao, Yonghui Wu, and Mingxuan Wang. Dapo: An open-source llm reinforcement learning system at scale, 2025.

[36] Aohan Zeng, Xiao Liu, Zhengxiao Du, Zihan Wang, Hanyu Lai, Ming Ding, Zhuoyi Yang, Yifan Xu, Wendi Zheng, Xiao Xia, Weng Lam Tam, Zixuan Ma, Yufei Xue, Jidong Zhai, Wenguang Chen, Peng Zhang, Yuxiao Dong, and Jie Tang. Glm-130b: An open bilingual pre-trained model, 2023.

[37] Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, Zhiwei Liu, Yihao Feng, Tulika Awalgaonkar, Rithesh Murthy, Eric Hu, Zeyuan Chen, Ran Xu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. xlam: A family of large action models to empower ai agent systems, 2024.