CM2: Reinforcement Learning via Checklist Rewards for Multi-Turn Multi-Step Agentic Tool Use

Anonymous ACL submission

Abstract

AI agents are increasingly used to solve real-world tasks by reasoning over multi-turn user interactions and invoking external tools. However, applying reinforcement learning to such settings remains difficult: realistic objectives often lack verifiable rewards and instead emphasize open-ended behaviors; moreover, RL for multi-turn, multi-step agentic tool use is still underexplored; and building and maintaining executable tool environments is costly, limiting scale and coverage. We propose **CM2**, an RL framework that replaces verifiable outcome rewards with checklist rewards. **CM2** decomposes each turn's intended behavior into fine-grained binary criteria with explicit evidence grounding and structured metadata, turning open-ended judging into more stable classification-style decisions. To balance stability and informativeness, our method adopts a strategy of sparse reward assignment but dense evaluation criteria. Training is performed in a scalable LLM-simulated tool environment, avoiding heavy engineering for large tool sets. Experiments show that **CM2** consistently improves over supervised fine-tuning. Starting from Qwen3-8B-Base and training on an 8k-example RL dataset, **CM2** improves over the SFT counterpart by **8** points on τ^2-Bench, by **10** points on BFCL-V4, and by **12** points on ToolSandbox. The results match or even outperform similarly sized open-source baselines, including the judging model. **CM2** thus provides a scalable recipe for optimizing multi-turn, multi-step tool-using agents without relying on verifiable rewards.

1 Introduction

AI Agents are emerging as a promising paradigm for solving complex, real-world tasks (Jimenez et al., 2023; Phan et al., 2025; Wei et al., 2025a). By reasoning and invoking external tools, such as search engines, databases, proprietary APIs, and compilers, an agent can interact with external environments to transcend the limitations of its parametric knowledge (Jin et al., 2025; Jain et al., 2025). Unlike traditional question answering (Kamalloo et al., 2023), these agents require the ability to navigate **multi-turn** dialogues with users and execute **multi-step** reasoning with tool use (Zhang et al., 2025). However, training general-purpose agents to master such interactions through reinforcement learning (RL) remains a huge challenge.

Three primary limitations hinder current RL research in this domain. First, existing work largely relies on **verifiable rewards** (Guo et al., 2025). Typical setups supervise agents based on the rule-based correctness of final answers or the exact match of the tool execution trace against ground-truth (Wei et al., 2025b). However, such signals are often not applicable in realistic, open-ended scenarios, where objectives may include asking clarifying questions, maintaining a helpful tone, or providing suggestions (Gunjal et al., 2025; Liu et al., 2025; Viswanathan et al., 2025). Second, RL for multi-turn and multi-step interactions is underexplored. Most current works rely heavily on supervised fine-tuning (SFT) with synthetic data (Qin et al., 2023) or RL limited to multi-step reasoning without multi-turn dynamics (Yu et al., 2024). While these methods endow models with basic capabilities, they often struggle to generalize to unseen tools, extended horizons, and richer user interactions. Third, scaling tool-use RL is fundamentally constrained by tool environment construction. Implementing tool APIs and maintaining reliable execution environments incurs substantial engineering overhead and makes it difficult to scale to large and diverse tools (Liu et al., 2024; Ruan et al., 2023).

To address these challenges, we propose **CM2** (Checklist Reward for Multi-turn Multi-step Agentic Tool Use), an RL training framework for **multi-turn** and **multi-step** tool-use agent, **without relying on rule-based verifiable rewards**. The RL training is performed in a **scalable LLM-**

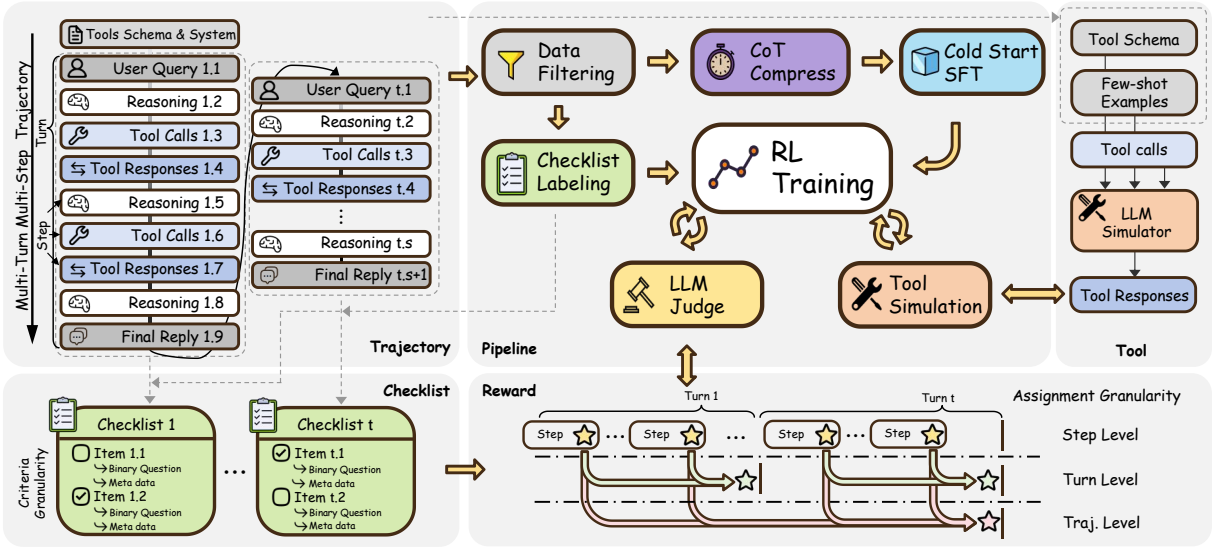


Figure 1: Overview of our **CM2**. Starting from multi-turn, multi-step tool-use trajectories, we perform data filtering, CoT compression, and cold-start SFT, then annotate a per-turn checklist with evidence-grounded binary criteria and structured metadata. RL training is carried out in an LLM-simulated tool environment, where a LLM simulator produces tool responses and an LLM-as-a-Judge evaluates checklist items to compute rewards. The bottom panel contrasts dense criteria granularity with sparse reward assignment at different assignment granularities.

simulated tool environment containing 5,000 tools. The workflow is illustrated in Figure 1.

The core idea of **CM2** is to replace the verifiable rewards with checklist rewards, decomposing the agent's intended behavior in each turn into fine-grained **binary** evaluation criteria, where each criterion is equipped with explicit evidence grounding, dependencies, and weights. This formulation turns open-ended judging into more stable classification-style decisions, while retaining interpretability and compositionality for complex objectives.

A central design question is how to trade off signal density and training stability under noisy tool simulations and LLM-based judging. We find that simply making rewards denser along the trajectory can amplify noise and destabilize optimization. **CM2** therefore adopts a strategy: *Sparse in assignment; Dense in criteria*. Rewards are assigned conservatively, while supervision remains informative. To study assignment granularity systematically, we instantiate three advantage estimation variants: trajectory-level, turn-level, and step-level. We also introduce a reward backfilling mechanism that attributes delayed checklist satisfaction to earlier critical steps when dependencies are met, improving credit assignment in long interactions.

To enable **scalable** training across diverse tools without heavy engineering, **CM2** performs RL in an **LLM-simulated tool environment** containing 5,000+ tools. The simulator supports hybrid execution by replaying recorded tool I/O when available and falling back to LLM-based tool response simulation otherwise. This method enables large-scale, execution-free interaction while maintaining contextual consistency, thereby improving training robustness (Liu et al., 2024; Ruan et al., 2023).

Empirically, **CM2** yields significant improvements across multiple challenging benchmarks. Starting from Qwen3-8B-Base and training on an 8k-example RL dataset, CM2 improves over the SFT counterpart by 8 points on τ^2-Bench (Yao et al., 2024; Barres et al., 2025), by 10 points on BFCL-V4 (Patil et al., 2025), and by 12 points on ToolSandbox (Lu et al., 2025). The results match or even outperform similarly sized open-source baselines. More importantly, **CM2** enables robust reinforcement learning for agentic systems without requiring manual environment-specific reward engineering, demonstrating that Checklist rewards provide an effective and scalable supervision signal for training general-purpose agents capable of multi-turn, multi-step tool use, particularly invaluable in domains where explicit and verifiable rewards are unavailable, offering a practical pathway toward large-scale optimization of agentic tool use capabilities.

2 Related Work

2.1 Reward for RL

Recent advances have shifted from SFT toward RL to enhance the generalization and robustness of agent behavior. A dominant paradigm is Reinforcement Learning with Verifiable Rewards (RLVR) (Guo et al., 2025), which leverages de-

terministic signals to guide optimization. However, applying RLVR to open-ended problems remains challenging due to the absence of ground-truth verifiers. Traditionally, Reinforcement Learning from Human Feedback (Schulman et al., 2017; Rafailov et al., 2023) addresses this limitation by training reward models on human preference data to provide scalar signals (Hong et al., 2025; Mahan et al., 2024). Yet these holistic scalar rewards are often opaque and insufficient for guiding complex multi-step reasoning. To overcome this issue, recent work has turned to criterion-based rewards. Frameworks such as Reinforcement Learning with Rubric-based Rewards (Gunjal et al., 2025; Liu et al., 2025; Pathak et al., 2025) and Reinforcement Learning from checklist Feedback (Viswanathan et al., 2025) decompose instruction execution into fine-grained checklist items or criteria, which are then evaluated by LLMs serving as judges. These studies demonstrate that dense, structured feedback substantially outperforms opaque scalar rewards from standard reward models.

2.2 Multi-Turn Multi-Step Agent RL

The evolution from single-step to multi-turn, multi-step agent interactions poses significant challenges for state tracking and credit assignment in RL training. Recent benchmarks (Lu et al., 2025; Barres et al., 2025; Patil et al., 2025) emphasize the importance of stateful dynamics, requiring agents to maintain contextual consistency and execute coherent tool-calling sequences over extended horizons. While these benchmarks effectively evaluate multi-turn dialogue or multi-step reasoning capabilities, existing work largely treats these two aspects in isolation, with few studies using RL to simultaneously optimize the compositional complexity arising from multi-turn dialogue dynamics and multi-step tool-use trajectories. Recently, MUA-RL (Zhao et al., 2025) first integrated LLM-simulated users into RL loops but relies on binary outcome rewards and optimizes on in-domain evaluation data, failing to address sparse reward challenges in long interactions. In contrast, **CM2** employs fine-grained checklist rewards to explicitly reinforce correct intermediate steps, effectively mitigating credit assignment problems and enabling more robust dialogue policies and tool-use patterns.

2.3 LLM-Simulated Tool Environments

The fundamental limitation in extending RL to tool-use domains lies in the engineering overhead of maintaining real-world APIs (Lu et al., 2025; Jain et al., 2025). To address this challenge, LLM-based environment simulation has become the dominant paradigm. SynthAgent (Wang et al., 2025) proposes a fully synthetic supervision framework for web agents with trajectory optimization to enhance performance; ToolEmu (Ruan et al., 2023) demonstrates the effectiveness of LLM-simulated sandboxes in identifying risky behaviors, enabling safety evaluation without actual tool infrastructure. Simia (Li et al., 2025) shows that powerful LLMs can faithfully simulate environment feedback based on tool definitions and interaction history, while Generalist Tool Model (GTM) (Ren et al., 2025) introduces a specialized 1.5B parameter model to simulate the execution of over 20,000 tools. In contrast, **CM2** scales to arbitrary tools, enabling large-scale training across diverse domains and synthetic edge cases that improve robustness.

3 RL via Checklist Rewards for Agentic Tool Use

In this section, we introduce our **CM2** method. We first formulate the problem of multi-turn and multi-step agentic tool calling in Section 3.1 and then define two dimensions of granularity in reward modeling for agentic tasks in Section 3.2. Subsequently, Section 3.3 describes the shaping and labeling process of the Checklist rewards. Finally, we detail how to do RL training with Checklist rewards in Section 3.4.

3.1 Problem Formulation

As shown in the upper left part of Figure 1, we consider a **multi-turn and multi-step** dialogue $\mathcal{D}$ between a user u and an agent π_θ equipped with a set of tools $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$. A dialogue is composed of multiple **turns**: $\mathcal{D} = \{\tau_1, \tau_2, \ldots, \tau_L\}$, where each turn τ_t consists of a sequence of **steps**: $\tau_t = \{\sigma_{t,1}, \sigma_{t,2}, \ldots, \sigma_{t,M_t}\}$. Each step $\sigma_{t,s}$ is categorized into one of three types: **(1) User Query**, marking the initiation of a turn; **(2) Agent Action**, which comprises: (i) an internal *Reasoning process* $z_{t,s}$ that precedes an action, and (ii) an explicit *action* $a_{t,s}$, which may be tool calls or a final reply; **(3) Tool Responses**, which are the output returned by the tool invoked in the preceding agent action.

We employ **Interleaved Thinking** (Xie et al., 2025) to maintain context, and keep the thinking process from previous turns. The dialogue context $h_{t,s}$ is defined as the complete observable

```
{"id": "D3",
"evidence": [{
    "turn": 1, "step": 2,
    "from":
    ↪  "assistant.final_reply",
    "snippet": "which exceeds your
    ↪  $500 target.\n\n###
    ↪  Recommendations for
    ↪  Budget-Friendly
    ↪  Alternatives:"}],
"focus_on":
↪  "assistant.final_reply",
"question": "Does the assistant
↪  propose alternative
↪  budget-friendly van options or
↪  adjustments instead of
↪  generating a caption?",
"pass_condition": "The final reply
↪  offers at least one
↪  cost-lowering alternative
↪  (e.g., cheaper van, longer
↪  term, smaller vehicle) and does
↪  not proceed to
↪  caption/hashtags.",
"failure_examples": [
    "Assistant generates
    ↪  caption/hashtags despite
    ↪  payment > $500",
    "Assistant provides no
    ↪  alternative options"],
"strictness": true,
"dependency": ["D1"],
"weight": 0.2}
```


Figure 2: Example of One Checklist Item

history up to step $\sigma_{t,s}$: $h_{t,s} = \{\tau_1, \ldots, \tau_{t-1}\} \cup \{\sigma_{t,1}, \ldots, \sigma_{t,s}\}$. At each agent action step, the model first generates reasoning $z_{t,s} \sim \pi_\theta(z \mid h_{t,s})$, followed by an action $a_{t,s} \sim \pi_\theta(a \mid h_{t,s}, z_{t,s})$. If the action $a_{t,s}$ is a tool call, the tool environment executes the selected tool T_i with arguments and returns an observation $r^{\text{tool}}_{t,s} = T_i(a_{t,s})$, which is then appended to the history to form $h_{t,s+1}$. If the action is a final reply, the current turn terminates, and any subsequent user query initiates a new turn.

3.2 Two Types of Granularity in Reward Modeling

Before detailing our Checklist reward shaping, we define two orthogonal dimensions of reward granularity: *Assignment Granularity* and *Criteria Granularity*. These dimensions address two fundamental questions: *where* rewards are assigned along the trajectory, and *what* criteria are used for evaluation.

Assignment Granularity refers to the credit assignment of reward signals across the sequence of outputs. This dimension distinguishes between sparse and dense reward signals. At the coarse-grained level, the reward is assigned to the final state of a trajectory, treating the entire sequence as a single unit of evaluation. In contrast, the fine-grained level distributes reward signals across intermediate steps to evaluate the incremental progress of the generation.

Table 1: Components of a checklist item.

Component	Description
Evidence	Pointers to the specific segment(s) in the original trajectory that this item is annotated from.
Focus	The step type this item targets (e.g., tool calls, reasoning, final reply, or tool response), to help the judge localize the relevant context.
Question	A binary checklist question to be answered for this item.
Pass/Fail	Explicit criteria defining when the item passes or fails.
Strictness	A boolean flag (required_for_next_turn) indicating whether this item must pass for the conversation to proceed to the next turn since user query is fixed.
Dependency	Dependencies indicating whether this item can only be satisfied after other item(s) are satisfied.
Weight (w)	The item's relative weight within a turn, with $\sum_i w_i = 1$.

Criteria Granularity concerns the specificity of the evaluative metrics. Coarse-grained evaluation is holistic, where the reward reflects a single judgment, such as task completion or correctness. Fine-grained criteria decompose evaluation into multiple sub-dimensions (e.g., helpfulness, harmfulness, accuracy), each weighted according to a specific rubric.

While increasing granularity in both dimensions theoretically provides denser signals, our empirical observations in agentic scenarios suggest a decoupled strategy. Due to the inherent noise in the environment, **coarse-grained assignment** yields a more stable training curve. Concurrently, **fine-grained criteria** deliver the essential, task-specific guidance required to navigate complex tool-use logic. Consequently, we adopt a strategy characterized as *Sparse in assignment; Dense in criteria*.

3.3 Checklist Reward Shaping

In this section, we introduce the **Checklist-based Reward Shaping** that can provide two types of fine-grained reward signals for multi-turn and multi-step RL training for agentic tool use.

Composition of the Checklist. As shown in the bottom left of Figure 1, for each turn τ_t, we label a *Checklist* Γ_t that contains several items $\{\gamma_1, \ldots, \gamma_{N_t}\}$. The annotator LLM is prompted to decompose the agent's intended behavior in each

turn into multiple fine-grained subtasks. Each subtask, which is called a Checklist item, has one binary question and is enriched with detailed metadata that defines its semantics and constraints as shown in Table 1. The example of one Checklist item is illustrated in Figure 2.

Why Checklist Rewards? Checklist formulates each criterion as a *binary* pass/fail decision with explicit evidence and conditions, turning LLM judging from open-ended scoring (regression) into a more **stable** and easy classification-style evaluation. This substantially reduces judge randomness; otherwise, small stochastic score differences can be amplified by per-batch return or advantage normalization in RL, changing within-batch rankings and leading to unstable or even contradictory gradients (Viswanathan et al., 2025). Besides, this structured metadata ensures that the Checklist is **interpretable**, allowing automated and consistent evaluation across turns with less noise.

Post-hoc Checklist Annotation. In practice, we label the Checklist by *post-hoc* structuring an existing multi-turn and multi-step tool use trajectory rather than from scratch. For each turn, we prompt an LLM to (i) infer the turn-level intent and required outcomes from the user query and the assistant/tool traces, and (ii) decompose them into a concise set of **binary, observable** Checklist items grounded in the trajectory. Each trajectory only costs approximately $0.1 on average, making it practical to scale checklist labeling to large datasets without significant overhead compared with training costs and manual annotation. The prompt and annotation details are provided in Appendix A.1.1.

Rollout and Reward Computation. During rollout, at each step within turn τ_t, we query a judge LLM with the trajectory prefix (history so far) together with the checklist items for that turn. The judge returns a Boolean label for each item, indicating whether it is currently satisfied by the partial trajectory. After the agent produces the final user-visible response for the turn, we enforce the strictness constraints: if all strictness items are satisfied, we issue the next user query from the reference trajectory; otherwise, we terminate the rollout early.

3.4 Checklist-based RL Optimization

RL algorithm based on Group Relative Policy Optimization (GRPO) are typically formulated around outcome rewards. However, our Checklist-based framework enables the extraction of dense reward signals down to the individual step level. To systematically investigate the impact of *Assignment Granularity*, we instantiate three distinct advantage estimation variants: (i) Trajectory-level, (ii) Turn-level, and (iii) Step-level. These variants differ primarily in how to assign the reward and calculate the advantage accordingly.

3.4.1 Checklist-based Reward

Let x_s denote the state before step s and x_{s+1} the state after step s. For dialogue i, turn t, and checklist item c, let $\mathrm{Sat}^{(i)}_{t,c}(x_s) \in \{0,1\}$ denote whether $\gamma^{(i)}_{t,c}$ is satisfied in state x_s. Let $\mathrm{Dep}_{t,c} = \{\, c' \mid \gamma_{t,c'}$ is a dependency (prerequisite) of $\gamma_{t,c} \,\}$ be the set of dependency items of $\gamma^{(i)}_{t,c}$. Once $\gamma^{(i)}_{t,c}$ switches from unsatisfied to satisfied at step s, and all its dependencies are already satisfied in the pre-step state x_s, we assign a binary reward to that step:

$$r^{(i)}_{t,s,c} = \mathbf{1}\Big[\underbrace{\prod_{c' \in \mathrm{Dep}_{t,c}} \mathrm{Sat}^{(i)}_{t,c'}(x_s) = 1}_{\text{all deps. satisfied in } x_s} \wedge \underbrace{\mathrm{Sat}^{(i)}_{t,c}(x_s) = 0}_{\text{unsatisfied in } x_s} \wedge \underbrace{\mathrm{Sat}^{(i)}_{t,c}(x_{s+1}) = 1}_{\text{satisfied in } x_{s+1}} \Big]. \tag{1}$$

Since satisfying an item may require multiple steps, we further *backfill* the reward to every earlier step where all the dependencies were already satisfied. The backfilled reward is defined as

$$\tilde{r}^{(i)}_{t,s,c} = \mathbf{1}\Big[\underbrace{\prod_{c' \in \mathrm{Dep}_{t,c}} \mathrm{Sat}^{(i)}_{t,c'}(x_s) = 1}_{\text{all deps. satisfied before } s} \wedge \underbrace{\mathrm{Sat}^{(i)}_{t,c}(x_s) = 0}_{\text{unsatisfied before } s} \wedge \underbrace{\exists\, u \geq s \text{ s.t. } \mathrm{Sat}^{(i)}_{t,c}(x_{u+1}) = 1}_{\text{satisfied after } s} \Big]. \tag{2}$$

Note that we only use *backfilled* reward in step-level advantage.

3.4.2 Trajectory-level Advantage

Given a dialogue (rollout) $D^{(i)} = \{\tau_1, \ldots, \tau_{L^{(i)}}\}$, we first aggregate all Checklist-based rewards across turns, steps, and items as

$$R^{(i)} = \frac{1}{L^{(i)}} \sum_{t=1}^{L^{(i)}} \sum_{s} \sum_{c} w_{t,c} \cdot r^{(i)}_{t,s,c}, \tag{3}$$

where s ranges over steps in turn t and c ranges over checklist items for turn t and $R^{(i)} \in [0,1]$ since $\sum_s r^{(i)}_{t,s,c} \le 1$ (it only flips once) and $\sum_s \sum_c w_{t,c} \cdot r^{(i)}_{t,s,c} \le \sum_c w_{t,c} = 1$. For the group of G rollouts of the same prompt, we define the trajectory-level advantage as

$$A^{(i)}_{\text{traj}} = \frac{R^{(i)} - \text{mean}(\{R^{(i)}\}_{i=1}^{G})}{F_{\text{norm}}(\{R^{(i)}\}_{i=1}^{G})}. \quad (4)$$

3.4.3 Turn-level Advantage

To get the turn-level advantage, we aggregate Checklist-based rewards *within* each turn. For dialogue i and turn t, we define the turn reward as

$$R^{(i)}_t = \sum_s \sum_c w_{t,c} \cdot r^{(i)}_{t,s,c}, \quad (5)$$

where s ranges over steps in turn t and c ranges over checklist items for turn t and $R^{(i)}_t \in [0,1]$. Given a group of G rollouts of the same question, we compute a turn-level GRPO advantage as

$$A^{(i)}_{\text{turn},t} = \frac{R^{(i)}_t - \text{mean}\left(\{R^{(i)}_t\}_{i=1}^{G}\right)}{F_{\text{norm}}\left(\{R^{(i)}_t\}_{i=1}^{G}\right)}. \quad (6)$$

3.4.4 Step-level Advantage

For the step-level reward baseline, we first calculate a baseline in one group satisfy a certain Checklist item:

$$b_{t,c} = \frac{1}{G} \sum_{i=1}^{G} \mathbb{I}\left[\exists s' \text{ s.t. } r^{(i)}_{t,s',c} = 1\right]. \quad (7)$$

At step (t,s) in rollout i, multiple checklist items may be applicable simultaneously. We first compute an item-wise step advantage:

$$A^{(i)}_{t,s,c} = \frac{\tilde{r}^{(i)}_{t,s,c} - b_{t,c}}{F_{\text{norm}}(\{\mathbb{I}\left[\exists s' \text{ s.t. } r^{(i)}_{t,s',c} = 1\right]\}_{i=1}^{G})}, \quad (8)$$

and then aggregate them using the checklist weights:

$$A^{(i)}_{\text{step},t,s} = \frac{\sum_{c \in \mathcal{E}^{(i)}_{t,s}} w_{t,c}\, A^{(i)}_{t,s,c}}{\sum_{c \in \mathcal{E}^{(i)}_{t,s}} w_{t,c}}. \quad (9)$$

Here $\mathcal{E}^{(i)}_{t,s} = \{c \mid \prod_{c' \in \text{Dep}_{t,c}} \text{Sat}^{(i)}_{t,c'}(x_s) = 1 \wedge \text{Sat}^{(i)}_{t,c}(x_s) = 0\}$ denotes the set of checklist items that are eligible to be satisfied at step s (i.e., all dependencies are already satisfied and item c is not yet satisfied).

4 Training Pipeline

In this section, we outline the training pipeline of **CM2**, which encompasses data filtering, Chain-of-Thought (CoT) compression, cold-start SFT, checklist labeling, tool simulation, and RL training guided by an LLM-as-a-Judge. The overall workflow is illustrated in the top right of Figure 1.

4.1 Data Filtering

We start from the tool-calling subset of the NVIDIA/NEMOTRON-POST-TRAINING-DATASET-V1 dataset (Nathawani et al., 2025; Bercovich et al., 2025), which contains 310k synthetic tool-use dialogues spanning single-turn, multi-turn, and multi-step settings across diverse domains (e.g., shopping, financial analysis, and web search). Since all samples are distilled from Qwen3-235B-A22B (Team, 2025), the data contains substantial noise. We therefore apply a two-stage filtering pipeline to ensure quality: (1) **Rule-based filtering** removes examples with structural and formatting violations (criteria in Appendix A.4); (2) **LLM-based filtering** uses GPT-5(OpenAI, 2025) to further discard samples with deeper semantic or reasoning errors. The prompt and details are provided in Appendix A.1.2.

Data statistics. Rule-based filtering reduces the dataset from 310k to 280k examples, and LLM-based filtering further narrows it to 30k high-quality samples. From this set, we randomly sample **8k examples** for cold-start SFT, and the remaining 22k form the candidate pool for RL, from which we additionally exclude simpler cases (e.g., single-turn or single-tool interactions) and retain another **8k complex multi-turn, multi-step dialogues** for RL training, with 500 held out for validation.

4.2 CoT Compression and Cold Start

Before finalizing the training sets, we compress the original chain-of-thought (CoT) to improve inference efficiency and reduce context length. Specifically, we use GPT-5 to rewrite the thinking content into a shorter form while preserving the key planning and decisions (prompt in Appendix A.1.3). After compression, the resulting datasets are denoted as $\mathcal{D}_{\text{CS}}$ (cold-start SFT) and $\mathcal{D}_{\text{RL}}$ (RL training), respectively.

Finally, we fine-tune **Qwen3-8B-Base** on the $\mathcal{D}$CS. Hyperparameters and other training details are provided in Appendix A.2.

4.3 Tool Simulation and LLM-as-a-Judge

Because trajectories are synthetic, there is no executable environment available during RL. To avoid building and maintaining 5,000+ unique tools, we implement a hybrid tool simulator. Upon a tool invocation, the simulator first performs an **exact match** against the original tool name and arguments; if matched, it returns the recorded tool response. Otherwise, we fall back to **LLM-based simulation**: we prompt an LLM with in-dialogue tool I/O exemplars as few-shot learning to generate a response that remains consistent with the trajectory context, enabling scalable, execution-free interaction. For **LLM-as-a-Judge** for checklist rewards, we prompt an LLM at each step to answer each question in checklist. Then we aggregate reward as in Section 3. The judging prompt is provided in Appendix A.1.4. We use **Qwen3-30B-A3B** for both tool simulation and LLM-as-a-Judge, chosen to balance quality and throughput. Later experiments show that even a lightweight judge with merely 3B active parameters enables the model to attain highly competitive or even surpassing results.

4.4 Checklist Labeling and RL Training

Following Section 3.3, we use GPT-5 to annotate a per-turn Checklist for each dialogue (prompts in Appendix A.1.1). We then optimize from the cold-start SFT checkpoint using GRPO based on VeRL, and apply the multi-level advantage comparison described in Section 3.4. The RL model is trained on 64 GPUs for 680 hours. Additional implementation details and hyperparameters are deferred to Appendix A.3.

5 Results

5.1 Effect of Allocation Granularity

Figure 3a compares the reward curve on validation set under different assignment granularities. Finer-grained allocation yields faster early improvements: step-level advantages outperform turn-level, which in turn outperform trajectory-level in the initial phase. As training continues, however, finer granularities exhibit earlier and more severe training collapse, while trajectory-level advantages remain more stable and continue to improve.

We attribute this trade-off to noise amplification in agentic RL. Checklist rewards reduce judge variance by turning open-ended scoring into binary, evidence-grounded decisions, but they do not eliminate stochasticity. With finer-grained assignment, this residual noise enters optimization more frequently and can be amplified by group-relative normalization, yielding higher-variance or sometimes misleading gradients. This motivates our principle of *Sparse in assignment; Dense in criteria*: we keep evaluation criteria fine-grained for informative supervision, while assigning rewards at coarser granularity to average out residual noise and improve stability.

Model / Method	Airline	Retail	Telecom	Avg.
Open-source Baselines				
Qwen3-30B-A3B-Instruct-2507	32.50	50.88	12.72	32.03
Qwen3-8B-Thinking	30.00	43.64	22.37	32.00
Ours (from **Qwen3-8B-Base**)				
Cold-start SFT on $\mathcal{D}_{\text{CS}}$	25.50	18.42	11.84	18.59
$\hookrightarrow$ SFT on $\mathcal{D}_{\text{RL}}$	23.50	19.52	12.06	18.36
$\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (**CM2**)	27.00	36.40	16.89	26.76

Table 2: Results on the τ^2-Bench benchmark. We run evaluation four times and report the average accuracy

	Multi-Turn					Web Search		
Model / Method	Base	Miss Func	Miss Param	Long Ctx	Overall	Base	No Snippet	Overall
Open-source Baselines								
Qwen3-30B-A3B-Instruct-2507	45.0	28.0	21.0	42.5	*34.25*	24.00	17.00	20.50
Qwen3-8B-Thinking	42.5	38.5	31.5	35.5	*37.00*	19.00	11.00	15.00
Ours (from **Qwen3-8B-Base**)								
Cold-start SFT on $\mathcal{D}_{\text{CS}}$	24.5	19.0	14.5	19.5	*19.37*	18.00	10.00	14.00
$\hookrightarrow$ SFT on $\mathcal{D}_{\text{RL}}$	30.0	27.5	24.5	25.0	*26.75*	18.00	9.00	13.50
$\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (**CM2**)	44.5	32.0	35.0	34.5	***36.50***	41.00	14.00	**27.50**

Table 3: Results on the BFCL-V4 benchmark (Multi-Turn and Web Search subset).

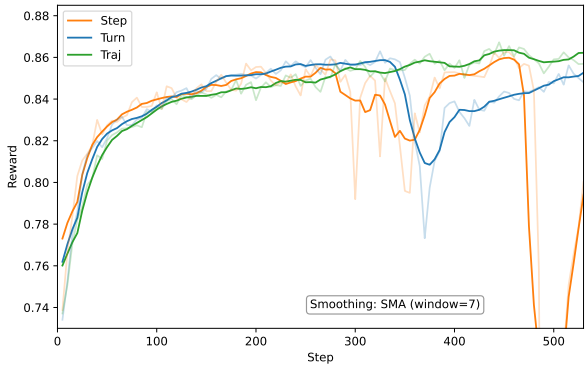
5.2 Effect of Group Size

Figure 3b shows the impact of group size G (e.g., G=24 vs. G=48) with trajectory-level Checklist rewards. A larger group size consistently achieves higher rewards. Intuitively, for multi-turn, multi-step trajectories, increasing G provides more samples for later turns, leading to a lower-variance advantage estimate more reliable gradient updates.

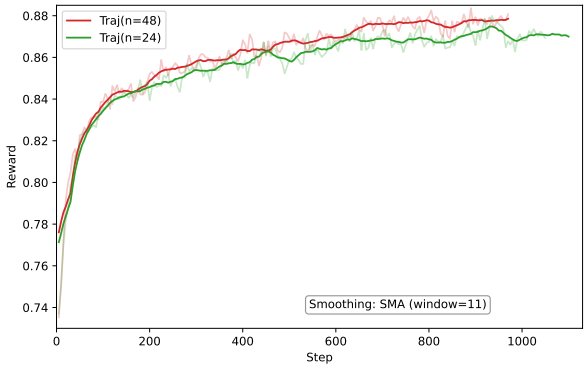
5.3 Results on Benchmarks

We evaluate our proposed **CM2** using our final configuration (trajectory-level advantage estimation with group size $G = 48$) on three challenging multi-turn, multi-step tool-use benchmarks: τ^2-Bench, BFCL-V4, and ToolSandbox. We compare against the SFT counterparts and open-source models of similar size.

τ^2-Bench Benchmark. The results on τ^2-Bench are summarized in Table 2. For each question, we run evaluation four times and report average accuracy. As shown in Table 2, starting from **Qwen3-8B-Base** (Team, 2025), our RL model outperforms SFT by over 8 points, demonstrating the effective-



(a) Reward curves on the validation set for different advantage assignment granularities.



(b) Reward curves on the validation set between different group sizes (n).

Figure 3: Comparison results under different settings.

Model / Method	Overall Score ↑	Scenario Categories							Tool Augmentations							
		STC	MTC	SUT	MUT	SD	C	II	0-DT	3-DT	10-DT	AT	TNS	TDS	ADS	ATS
Open-source Baselines																
Qwen3-30B-A3B-Instruct-2507	65.24	84.18	69.14	74.52	65.33	75.11	66.95	40.97	64.29	68.23	60.98	66.62	68.56	63.32	66.17	63.74
Qwen3-8B-Thinking	65.47	77.12	58.91	64.07	57.82	60.65	56.71	76.77	70.96	67.69	64.55	60.79	69.00	56.98	65.08	68.71
Ours (from **Qwen3-8B-Base**)																
Cold-start SFT on $\mathcal{D}_{\text{CS}}$	56.19	71.65	47.89	54.91	45.72	63.08	45.93	69.97	55.44	56.68	56.03	53.86	60.34	55.81	52.82	58.53
↪ SFT on $\mathcal{D}_{\text{RL}}$	55.32	74.89	46.66	55.71	42.22	59.25	44.35	67.41	55.69	54.09	55.23	50.42	62.42	55.27	53.42	56.04
↪ RL on $\mathcal{D}_{\text{RL}}$ (**CM2**)	**68.20**	78.46	66.12	69.23	63.40	67.36	63.41	70.31	69.82	63.97	65.89	65.25	74.06	67.03	67.78	71.81

Table 4: Performance on various scenarios and tool augmentations. Our models are trained from **Qwen3-8B-Base**; we do not report results for the base checkpoint since it is not instruction-tuned under this evaluation protocol. Here, $\mathcal{D}_{\text{CS}}$ denotes the 8k cold-start SFT set, and $\mathcal{D}_{\text{RL}}$ denotes the 8k complex multi-turn, multi-step RL training set.

ness of **CM2**. However, our RL training uses a maximum context length of 10k and up to 30 turns, whereas τ^2-Bench can require >30k context and up to 200 turns. Under this mismatch, **CM2** lags behind some open-source models such as Qwen3-30B-A3B-Instruct-2507 and Qwen3-8B-Thinking.

BFCL Benchmark. Table 3 summarizes the results on BFCL-V4 (Multi-Turn and Web Search). Overall, our RL model trained on $\mathcal{D}_{\text{RL}}$ (**CM2**) substantially improves over SFT variants: on Multi-Turn, it achieves 36.50 overall accuracy, outperforming cold-start SFT and further SFT on $\mathcal{D}_{\text{RL}}$ by 10 points. On Web Search, RL also yields the best overall performance, improving over cold-start SFT and SFT on $\mathcal{D}_{\text{RL}}$ by 13.5 and 14 points, respectively. Compared with open-source baselines, our RL model performs better than Qwen3-30B-A3B-Instruct-2507 (judging model) on Multi-Turn and is comparable to Qwen3-8B-Thinking, while significantly surpassing both baselines on Web Search.

ToolSandbox Benchmark. Table 4 reports performance on ToolSandbox Benchmark. RL on $\mathcal{D}_{\text{RL}}$ (**CM2**) yields a large improvement over both SFT variants, increasing the overall score by more than 12 points. It also improves consistently across nearly all scenario categories, with particularly notable gains on multi-turn and multi-tool settings. Our RL model (**CM2**) also outperforms the open-source models, including the judging model.

Summary. Our method consistently yields substantial gains over SFT, with improvements that are stable across benchmarks. Notably, the resulting policy matches and often surpasses the LLM-as-a-judge model on most evaluation measures, while remaining competitive with or exceeding similarly sized open-source baselines. We further find that a lightweight judge is sufficient to drive strong RL improvements, and the learned behavior generalizes well to previously unseen benchmarks.

6 Conclusion

CM2 presents a scalable reinforcement learning framework for multi-turn, multi-step tool-using agents by replacing verifiable rewards with checklist rewards, which is fine-grained, binary, evidence-grounded criteria that make LLM judging more stable and interpretable. By adopting a "sparse in reward assignment, dense in evaluation criteria" strategy and training within an LLM-simulated tool environment, **CM2** improves over supervised fine-tuning across multiple benchmarks and shows stronger generalization to complex, long-horizon tool-use behaviors where verifiable rewards are unavailable.

Limitations

Long-context and very long-horizon training remains constrained. Our training setup is bounded by the model context window and a maximum turn budget, which limits the amount of history the policy can condition on during RL. When evaluation tasks require substantially longer interaction histories (larger contexts and more turns), this train–test mismatch can cap performance, especially for problems that demand persistent memory, delayed credit assignment, or recovery after many intermediate steps.

Simulated tool environments may diverge from real execution. While using an LLM to simulate tool responses greatly improves scalability across many tools, the simulator can produce "plausible" outputs that violate real API constraints (e.g., strict schemas, parameter bounds, error modes, rate limits, or edge-case behavior). As a result, policies trained in simulation may overfit to simulator idiosyncrasies and transfer imperfectly to real tool backends, with larger degradation on tools that have rigid interfaces or complex boundary conditions. Nevertheless, our experiments show strong generalization to multiple held-out benchmarks that are not seen during training, suggesting that this simulation-to-real gap has limited impact in our evaluated settings.

These limitations can likely be mitigated with better data and stronger models. For long-context and long-horizon settings, collecting or constructing trajectories that explicitly cover longer histories—and training with larger context windows and higher turn budgets—would reduce train–test mismatch and improve robustness on delayed-credit tasks. For simulation-to-real gaps, higher-quality tool-interaction data (including real API traces, failure cases, and edge conditions) and more capable simulator/judge models can better capture true tool constraints and error distributions, improving fidelity and downstream transfer to real tool execution.

Ethical Considerations

Our training data is derived from publicly available datasets and does not contain personally identifiable information. The LLM-simulated tool environment and LLM-as-a-Judge annotation pipeline may inherit biases from their underlying models, potentially affecting reward quality and policy behavior. We did not employ human annotators for checklist labeling, thus avoiding direct labor concerns; however, automated annotation may introduce systematic errors that are harder to audit. We encourage future work to examine potential biases in both the simulator and judge components before deployment in high-stakes applications.

References

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. 2025. tau^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*.

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, and 114 others. 2025. Llama-nemotron: Efficient reasoning models. *Preprint*, arXiv:2505.00949.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. 2025. Group-in-group policy optimization for llm agent training. *arXiv preprint arXiv:2505.10978*.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as rewards: Reinforcement learning beyond verifiable domains. *arXiv preprint arXiv:2507.17746*.

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*.

Ilgee Hong, Changlong Yu, Liang Qiu, Weixiang Yan, Zhenghao Xu, Haoming Jiang, Qingru Zhang, Qin Lu, Xin Liu, Chao Zhang, and 1 others. 2025. Think-rm: Enabling long-horizon reasoning in generative reward models. *arXiv preprint arXiv:2505.16265*.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *arXiv preprint arXiv:2504.07164*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2023. Swe-bench: Can language models resolve real-world github issues? *arXiv preprint arXiv:2310.06770*.

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*.

Ehsan Kamalloo, Nouha Dziri, Charles Clarke, and Davood Rafiei. 2023. Evaluating open-domain question answering in the era of large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5591–5606.

Yuetai Li, Huseyin A Inan, Xiang Yue, Wei-Ning Chen, Lukas Wutschitz, Janardhan Kulkarni, Radha Poovendran, Robert Sim, and Saravan Rajmohan. 2025. Simulating environments with reasoning models for agent training. *arXiv preprint arXiv:2511.01824*.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. 2025. Openrubrics: Towards scalable synthetic rubric generation for reward modeling and llm alignment. *arXiv preprint arXiv:2510.07743*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, and 1 others. 2024. Toolace: Winning the points of llm function calling. *arXiv preprint arXiv:2409.00920*.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, and 1 others. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 1160–1183.

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. 2024. Generative reward models. *arXiv preprint arXiv:2410.12832*.

Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. 2025. Nemotron-Post-Training-Dataset-v1.

OpenAI. 2025. Introducing GPT-5. https://openai.com/index/introducing-gpt-5/. Accessed: 2026-01-05.

Aditya Pathak, Rachit Gandhi, Vaibhav Uttam, Arnav Ramamoorthy, Pratyush Ghosh, Aaryan Raj Jindal, Shreyash Verma, Aditya Mittal, Aashna Ased, Chirag Khatri, and 1 others. 2025. Rubric is all you need: Improving llm-based code evaluation with question-specific rubrics. In *Proceedings of the 2025 ACM Conference on International Computing Education Research V. 1*, pages 181–195.

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

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, Sean Shi, and 1 others. 2025. Humanity's last exam. *arXiv preprint arXiv:2501.14249*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.

Zhenzhen Ren, Xinpeng Zhang, Zhenxing Qian, Yan Gao, Yu Shi, Shuxin Zheng, and Jiyan He. 2025. Gtm: Simulating the world of tools for ai agents. *arXiv preprint arXiv:2512.04535*.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J Maddison, and Tatsunori Hashimoto. 2023. Identifying the risks of lm agents with an lm-emulated sandbox. *arXiv preprint arXiv:2309.15817*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Vijay Viswanathan, Yanchao Sun, Shuang Ma, Xiang Kong, Meng Cao, Graham Neubig, and Tongshuang Wu. 2025. Checklists are better than reward models for aligning language models. *arXiv preprint arXiv:2507.18624*.

Zhaoyang Wang, Yiming Liang, Xuchao Zhang, Qianhui Wu, Siwei Han, Anson Bastos, Rujia Wang, Chetan Bansal, Baolin Peng, Jianfeng Gao, and 1 others. 2025. Adapting web agents with synthetic supervision. *arXiv preprint arXiv:2511.06101*.

Jason Wei, Zhiqing Sun, Spencer Papay, Scott McKinney, Jeffrey Han, Isa Fulford, Hyung Won Chung, Alex Tachard Passos, William Fedus, and Amelia Glaese. 2025a. Browsecomp: A simple yet challenging benchmark for browsing agents. *arXiv preprint arXiv:2504.12516*.

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

Roy Xie, David Qiu, Deepak Gopinath, Dong Lin, Yanchao Sun, Chong Wang, Saloni Potdar, and Bhuwan Dhingra. 2025. Interleaved reasoning for large language models via reinforcement learning. *arXiv preprint arXiv:2505.19640*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *Preprint*, arXiv:2406.12045.

Yuanqing Yu, Zhefan Wang, Weizhi Ma, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. 2024. Steptool: Enhancing multi-step tool usage in llms through step-grained reinforcement learning. *arXiv preprint arXiv:2410.07745*.

Chen Zhang, Xinyi Dai, Yaxiong Wu, Qu Yang, Yasheng Wang, Ruiming Tang, and Yong Liu. 2025. A survey on multi-turn interaction capabilities of large language models. *arXiv preprint arXiv:2501.09959*.

Weikang Zhao, Xili Wang, Chengdi Ma, Lingbin Kong, Zhaohua Yang, Mingxiang Tuo, Xiaowei Shi, Yitao Zhai, and Xunliang Cai. 2025. Mua-rl: Multi-turn user-interacting agent reinforcement learning for agentic tool use. *arXiv preprint arXiv:2508.18669*.

A Appendix

A.1 Prompts

We force models to generate output in JSON format to ensure instruction following and the output can be parsed.

A.1.1 Prompt of Checklist Labeling

The prompt for Checklist Annotation is shown in Prompt A.1.1. We use GPT-5 to label the Checklist. The parameter "effort" is set to "high" for high quality. Each trajectory only costs approximately $0.1 on average, making it practical to scale checklist labeling to large datasets without significant overhead.

```
# Instruction

You are an evaluation designer for
↪  **multi-turn, multi-step
↪  tool-use** dialogues.

## Your Task

Given a reference message list
↪  containing user, assistant, and
↪  tool steps, **produce a
↪  concise, per-turn checklist**
↪  of binary, observable criteria
↪  for judgment.
The checklist is used to judge
↪  whether another assistant meets
↪  the user's requirements.
One checklist per turn.


## Target of the assistant
The assistant needs to resolve the
↪  user's query in each turn.
It must analyze the user's intent
↪  in the private thinking, use
↪  tools to gather new information
↪  if necessary, plan the next
↪  steps based on the updated
↪  information and provide an
↪  user-visible reply to user.

## Input Format

### Conversation structure
↪  (multi-turn, multi-step)

* The conversation is chronological
↪  and split into **turns**.
* In each **turn**, there may be
↪  several steps from user,
↪  assistant, and tool:
  1. The **user** message appears
  ↪  **once** with questions or
  ↪  requirements.
  2. The **assistant** may think
  ↪  privately (Note: assistant
  ↪  content includes private
  ↪  thinking between <think> and
  ↪  </think>) and then either:
     * call one single tool or call
     ↪  multiple tools, **or**
     * generate a user-visible
     ↪  reply directly without
     ↪  calling tools.
  3. **Tool** messages return
  ↪  results to the preceding
  ↪  assistant message with tool
  ↪  calls.
  4. Repeat steps 2 and 3 until the
  ↪  turn ends.
* A turn **ends** when the
↪  assistant produces a
↪  user-visible reply after
↪  thinking.
* Only the **user-visible reply**
↪  is seen by the user.

### Candidate tools

You will also be given the schema
↪  of candidate tools for
↪  conversation. The tool calling
↪  should follow the schema
↪  (function name, required
↪  parameters, type of parameter)

### Message JSON schema (per step)

```json
{
  "role": "user|assistant|tool",
  "turn": 0,
  "step": 0,
  "content": "string containing
  ↪  either hidden thinking,
  ↪  user-visible reply, or tool
  ↪  output",
  "tool_calls": [
    {
      "id": "",
      "type": "function",
      "function": {
```
```

```
        "name": "TOOL_NAME",
        "arguments": { "Param":
        ↪  "Value", "...": "..." }
      }
    }
  ] # or None and []
}
```

* `turn` indexes start at **0**;
↪ `step` indexes start at **0**
↪ within each turn.

Rules for the Checklist

1. Each item must be a **YES/NO**
↪ question with an **objective
↪ pass condition**.
2. Items must be **observable**
↪ from user messages, assistant
↪ private thinking/tool
↪ calls/user-visible reply, and
↪ tool responses.
3. For each item, specify
↪ **evidence pointers** that
↪ reference specific assistant or
↪ tool step, not user at step 0.
4. If the task has prerequisite
↪ tool response (e.g., "search
↪ before analyze"), encode them
↪ via **`depends_on`**. The
↪ dependence must be a tool step.
5. Within a turn, the checklist
↪ should cover **key
↪ requirements** implied by that
↪ turn's user request, tool
↪ usage, constraints, and final
↪ reply (correctness,
↪ comprehensiveness, no
↪ hallucination, constraints,
↪ formatting, key reasoning
↪ steps, etc.).
6. Keep items atomic: ensure each
↪ checklist item evaluates a
↪ single, independent condition
↪ without combining multiple
↪ actions or operations.
7. Avoid purely stylistic or format
↪ checks; focus on key step to
↪ solve the user's requirements.
8. The question should focus on a
↪ specific part of the response,
↪ such as assistant.tool_calls,
↪ assistant.content.thinking,
↪ assistant.content.user_visible」
↪ _reply, or tool.content
↪ (focus_on).
9. Allow procedurally different
↪ operations, intermediate
↪ conclusions, or derived facts
↪ **as long as they produce the
↪ same verifiable result and
↪ strictly follow the user's
↪ requirements**.
10. Provide a **weight** for every
↪ item (0-1) and normalize
↪ weights so they **sum to 1.0
↪ per turn**, reflecting each
↪ requirement's contribution and
↪ necessity to the final
↪ user-visible reply.
11. For each item, include a
↪ must_pass_to_continue boolean.
↪ True means this item must pass;
↪ otherwise the conversation
↪ should not proceed to the next
↪ turn (critical failure). False
↪ means non-critical; failure is
↪ tolerable but counted against
↪ quality.
12. The reference messages may
↪ contain some failed attempts.
↪ The checklist should not
↪ mention anything about those
↪ unsuccessful attempts or
↪ self-correction.
13. Assume there is no error in
↪ tool calling.

Supplementary rules
1. Do not limit the number of tool
↪ calling.
2. Determine whether the value must
↪ match exactly or if a certain
↪ tolerance is acceptable.
3. Determine whether the parameter
↪ of tool calling must match
↪ exactly or if a certain
↪ tolerance is acceptable
4. The question about tool should
↪ align with the schema of
↪ candidate tool, e.g., argument
↪ with default value is not
↪ necessary.
5. Do not make any assumptions in
↪ the question, e.g., using if or
↪ when is question.
6. turn and step index should not
↪ appear or be refered to in
↪ checklist focus_on, question,
↪ pass_condition or
↪ failure_examples.

How the Checklist Will Be Used

We evaluate **every assistant step
↪ with possible following tool
↪ response steps** within a turn
↪ to determine which checklist
↪ items become newly satisfied
↪ **relative to the previous
↪ assistant step** (for `step=0`
↪ there is no previous step). We
↪ **do not** require the model to
↪ complete items at specific,
↪ pre-ordained steps from the
↪ input log; instead, we assess
↪ whether **all requirements for
↪ that turn** are satisfied **by
↪ the end of the turn**,
↪ regardless of which assistant
↪ step achieved them or how
↪ assistant achieved them.

```
## Examples


from should be one of
↪  user.content|assistant.tool_ca⌋
↪  lls|assistant.content.thinking⌋
↪  |assistant.content.user_visibl⌋
↪  e_reply|tool.content
[
  {
    "turn": 0,
    "checklist": [
      {
        "id": "C0", # start from 0
        ↪  in each turn
        "evidence": [{
          "turn": TURN_INDEX,
          "step": STEP_INDEX,
          "from": "...",
          "snippet": "..."
        }],
        "focus_on":
        ↪  "assistant.tool_calls",
        "question": "Did the
        ↪  assistant call the
        ↪  required tool TOOL_NAME
        ↪  with the correct
        ↪  parameter
        ↪  Param=Value?",
        "pass_condition": "There
        ↪  exists an assistant
        ↪  tool call with
        ↪  name=TOOL_NAME and
        ↪  arguments.Param ==
        ↪  Value or similar
        ↪  value.",
        "failure_examples": [
          "No tool call observed",
          "Wrong parameter value"
        ],
        "required_for_next_turn":
        ↪  true,
      },
      {
        "id": "C1",
        "evidence": [{
          "turn": TURN_INDEX,
          "step": STEP_INDEX,
          "from": "...",
          "snippet": "..."
        }],
        "focus_on": "tool.content",
        "question": "Did the
        ↪  assistant get xxx by
        ↪  calling the tool
        ↪  TOOL_NAME?",
        "pass_condition": "The
        ↪  assistant gets xxx from
        ↪  the tool response",
        "failure_examples": [
          "No tool response
          ↪  observed",
          "Wrong information from
          ↪  the tool"
        ],
        "required_for_next_turn":
        ↪  true,
      },
      {
        "id": "C2",
        "evidence": [{
          "turn": TURN_INDEX,
          "step": STEP_INDEX,
          "from": "...",
          "snippet": "..."
        }],
        "focus_on":
        ↪  "assistant.content.use⌋
        ↪  r_visible_reply",
        "question": "Does the final
        ↪  user-visible answer
        ↪  mentioned xxx?",
        "pass_condition": "The
        ↪  assistant's final reply
        ↪  content mentions xxx
        ↪  that answers user's
        ↪  question.",
        "failure_examples": [
          "Assistant does not
          ↪  mention xxx",
          "Numeric/text mismatch
          ↪  between answer and
          ↪  tool output"
        ],
        "required_for_next_turn":
        ↪  true,
      }
      // ...more items
    ],
    "dependence": {
      "C0": [], // if no
      ↪  dependence, use a empty
      ↪  list
      "C1": [],
      "C2": ["C1"] // dependence
      ↪  (e.g. C1 here) must focus
      ↪  on tool.content
    },
    "weight": {
      "C0": 0.3,
      "C1": 0.3,
      "C2": 0.4
    }    // ... must match item
    ↪  weights and sum to 1.0
  },
  // ... next turn checklist
]
```

A.1.2 Prompt of LLM-based Filtering

The prompt for LLM-based Filtering is shown in Prompt A.1.2 We use GPT-5 (OpenAI, 2025) as the filter model. To reduce API costs while maintaining high filtering quality, we adopt an aggressive, progressive evaluation strategy: each sample is sequentially evaluated twice at `low` effort, twice at `medium` effort, and twice at `high` effort. If any single evaluation flags the sample as problematic, it is immediately discarded without further processing. This aggressive early-exit mechanism ensures that only high-confidence, high-quality samples survive the filtering pipeline, at the cost of potentially discarding some borderline cases.

```
This is a simulated set of messages among a user, an assistant, and tools. The tools listed are the candidate tools. The assistant will first think (inside <think> and </think>; this part will be removed in post-processing and not shown to the user, and it should not be considered when judging whether there is an error), then decide whether to call a tool or produce a final response.

Does the logic for tool calling by the assistant follow the user's query, have no ambiguities, and can realistically occur in real scenarios? Are there any mistakes or flaws? Is it something that could exist in reality?
If there is no problem, answer true; if there is a problem, answer false.
You should be very strict.

Response format:
{
"Reasoning": string,
"NoError": true or false
}
```

A.1.3 Prompt of CoT Compression

The prompt for CoT Compression is shown in Prompt A.1.3. We use the default setting of GPT-5

```
# Instruction
You are given a multi-turn multi-step conversation consisting of messages. Each message has:
- role: one of [system | user | assistant | tool]
- content: textual content
- thinking (for assistant messages, there is a thinking section)
- tool_call / tool_result

# Task:
For every assistant message that contains a thinking section, produce a concise rewritten thinking section that preserves all essential reasoning, decisions, constraints, and references needed to justify the reply, while removing verbosity, filler, repetitions, speculative or unneeded self-talk.

# Important Requirements:
1. Preserve Meaning: Do not change conclusions, assumptions, selected tools, or rationale ordering unless reordering improves clarity without altering logic.
2. Keep Necessary Steps: Retain key logical steps, intermediate conditions, disambiguations, and any constraints that influence the final answer or tool choice.
3. Do NOT invent new facts or reasoning not present in the original thinking.
4. Do NOT shorten so aggressively that causal links or justification for tool calls become unclear.
5. Maintain references to tool names, parameters, or required outputs if they affect the final answer.
6. If the original thinking is already minimal, keep it (possibly with tiny clarity edits).
7. If a thinking section is empty or missing, output one for that item.
8. Output Format must be strict JSON as described below (no extra commentary).

# Input JSON Schema (example):
{
  "tools": [
    // tool schemas
  ]
  "messages": [
    {
      "role": "user",
      "content": "...",
      "tool_calls": []
    },
    {
      "role": "assistant",
      "content": {
        "thinking": "Reason for choosing tool ...",
        "reply": "(May be empty if just a tool call step)"
      }
      "tool_calls": [ ... ],
    },
    {
      "role": "tool",
      "content": "Tool result ...",
      "tool_calls": []
    },
    {
      "role": "assistant",
      "content": {
        "thinking": "LONG INTERNAL REASONING TEXT ...",
        "reply": "Visible answer to user ..."
      }
    }
```

```
  ]
}

# Output format:
Return a JSON array aligned with
↪  assistant messages order. Each
↪  element corresponds to one
↪  assistant message.
The total number should be the
↪  same.

[
  {
    "thinking": "..."
  },
  ...
]
```

A.1.4 Prompt of LLM Judge

The prompt for LLM judge is shown in Prompt A.1.4

```
# Role
You are a precise checklist
↪  evaluator. Your sole task is to
↪  judge whether the messages
↪  between user, assistant and
↪  tool satisfie the provided
↪  criteria.

# Objective
Produce a strict JSON verdict (no
↪  extra text) based on the
↪  instructions below.

# Criteria
**Question:** {this_turn_checklist」
↪  ['question']}
**Focus on:** {this_turn_checklist」
↪  ['focus_on']}
**Pass condition:** {this_turn_che」
↪  cklist['pass_condition']}
**Failure examples:** {this_turn_c」
↪  hecklist['failure_examples']}
**Reference snippet:**
↪  {reference_snippet}

# Previous Messages
{{messages_str_before_this_turn}}
# Current Messages to Evaluate
{{messages_str_in_this_turn}}

# Special rule of tool call
If there is no tool call in
↪  tool_call part but there are
↪  some tool calls in
↪  content.thinking part, it means
↪  these tools' format are not
↪  correct and all tool calls are
↪  not valid.If there is error in
↪  tool response. The previous
↪  tool calls in latest assistant
↪  (only the latest one) are not
↪  valid.# Evaluation Process
↪  (Align each step to a JSON
↪  output field)
1. high_level_understanding_of_the」
↪  _question:
   - Briefly restate what is being
   ↪  evaluated (the intent of the
   ↪  question + what compliance
   ↪  means here).
2. analysis_of_if_focus_on:
   - Check whether Focus on part
   ↪  presents in the Current
   ↪  Messages.
3. analysis_of_pass_condition:
   - Determine if the 'Pass
   ↪  condition' is fully
   ↪  satisfied.
4. analysis_of_failure_examples:
   - For EACH failure example
   ↪  pattern: state clearly
   ↪  'triggered' or 'not
   ↪  triggered' with a brief
   ↪  justification.
5. answer:
   - Return true ONLY IF:
     * Focus on part is present.
     * The 'Pass condition' is
     ↪  fully met.
     * No failure example pattern
     ↪  is triggered.
   - Otherwise return false.

# Output Format
Return ONLY a single JSON object
↪  with exactly these keys:
{
  "high_level_understanding_of_the」
  ↪  _question": str,
  "analysis_of_if_focus_on": str,
  "analysis_of_pass_condition":
  ↪  str,
  "analysis_of_failure_examples":
  ↪  str,
  "answer": bool
}
```

A.1.5 Prompt of Tool Simulation

The prompt for LLM judge is shown in Prompt A.1.5

```
# SYSTEM PROMPT
You are a precise tool executor
↪  that learns from examples.
You will be given:
- Tool call JSON Schema
- Few-shot examples showing tool
↪  calls and their execution
↪  results
- A new tool call with specific
↪  arguments

Your task:
1) Learn the OUTPUT FORMAT from the
↪  provided examples - follow the
↪  exact structure, data types,
↪  and response patterns
2) Ensure FACTUAL CONSISTENCY -
↪  your output should align with
↪  the factual information
↪  demonstrated in the examples
```

```
3) For the new tool call:
   - Apply the learned format to
   ↪  the new arguments
   - Maintain factual consistency
   ↪  with example patterns
   - If arguments are similar to
   ↪  examples, adapt the example
   ↪  results appropriately
   - If arguments are significantly
   ↪  different, generate new
   ↪  results following the
   ↪  learned format and factual
   ↪  patterns
   - May need to fix some type or
   ↪  error in the examples
4) Handle errors gracefully - if
↪  arguments are invalid or
↪  missing, return error messages
↪  in the same format as examples

Critical constraints:
- Act as a silent function executor
↪  - NO explanations, suggestions,
↪  or hints
- NO guidance on how to fix errors
↪  or improve calls
- NO references to examples or
↪  comparisons
- Return ONLY the raw execution
↪  result as valid JSON
- For errors, return minimal error
↪  information without
↪  instructional content

Output requirements:
- First do some analysis on how to
↪  mock the execution results.
↪  Then return ONLY the execution
↪  result as valid JSON array or
↪  object
- No explanations, markdown, or
↪  code fences
- Follow the exact output structure
↪  learned from examples
- Maintain factual consistency with
↪  the example patterns
Format:
{
  "analysis": str,
  "execution_result": JSON array or
   ↪  object,
}


# USER PROMPT
{"\n".join(examples_lines)}

Current tool name: {tool.name}
Current tool input schema (JSON
↪  Schema):
{schema_str}
Current arguments (JSON):
{parameters}
Generate tool execution result in
↪  JSON format.
```

A.2 Cold Start hyper-parameters

For cold-start training, we utilize the **LLaMAFactory** framework. The model is trained on the cold-start dataset for 2 epochs. We adopt the AdamW optimizer and a cosine learning rate schedule. The learning rate is set to `1e-6` with a warmup ratio of `0.1`. The batch size is `64`. The cold-start training is conducted on 8 H100 GPUs.

To better handle special tokens (e.g., `<think>`, `<tool_call>`) that are not trained during pre-training, we explicitly initialize their embeddings using the average of semantically related tokens. For example, the embedding of `<think>` is initialized as the mean of the embeddings for `think` and `begin`. This stabilizes optimization and speeds up convergence.

The initialization is as follows:

$$
\begin{aligned}
\texttt{<think>} &\leftarrow \text{avg}(\text{"think"}, \text{"begin"}) \\
\texttt{</think>} &\leftarrow \text{avg}(\text{"think"}, \text{"finish"}) \\
\texttt{<tool\_call>} &\leftarrow \text{avg}(\text{"tool"}, \text{"call"}, \text{"start"}) \\
\texttt{</tool\_call>} &\leftarrow \text{avg}(\text{"tool"}, \text{"call"}, \text{"end"}) \\
\texttt{<im\_start>} &\leftarrow \text{avg}(\text{"role"}, \text{"enter"}) \\
\texttt{<im\_end>} &\leftarrow \text{avg}(\text{"role"}, \text{"exit"})
\end{aligned}
$$

The training loss curve is shown in Figure **??**

A.3 RL Training Details

For reinforcement learning, we set the mini-batch size to 128 and the learning rate to 3e-6. The KL divergence loss coefficient is set to 0.001, and we sample 24 or 48 trajectories for one question as a group size. We adopt GRPO as our RL algorithm with the standard deviation term in the denominator set to 1, following (Feng et al., 2025). This improves the stability of the policy updates during training as we use a larger group size to ensure that later turns also receive a sufficient number of samples for sampling. We set the group number of 48 and use trajectory level reward for our final **CM2** model.

A.4 Rule-based Filtering Criteria

The criteria include: (1) violations of tool schemas; (2) incorrect role ordering; (3) mismatches between tool calls and subsequent responses; (4) tool responses erroneously placed within assistant messages; (5) invalid JSON formatting; (6) duplicate tool schemas or names; and (7) missing or redundant thinking tags (`<think>`).

A.5 Discussion: Scaling Up.

There are several natural axes to scale up **CM2**. First, we can increase the *number of checklists per turn* by generating multiple, independently instantiated checklists for the same turn (e.g., with different paraphrases or decompositions). Aggregating their outcomes (e.g., averaging or majority voting) can further reduce residual stochasticity and improve robustness to occasional missing or ambiguous criteria, at the cost of additional judging compute. Second, we can reduce judge noise more directly via *majority vote* (or other ensembling schemes) over multiple independent judgments of the same checklist. Third, **CM2** can benefit from *stronger judge models*, which provide more reliable evidence grounding and more consistent binary decisions. Beyond checklist-specific knobs, standard scaling strategies also apply, including using a *stronger base model* and a *larger group size* for advantage estimation, both of which typically improve optimization stability.

We expect these scaling directions to further stabilize training by suppressing residual stochasticity in tool use and judging. With sufficiently low-noise rewards, finer-grained reward assignment (e.g., step-level) may become viable, potentially retaining its fast early learning while avoiding premature collapse.