# CM2: Reinforcement Learning with Checklist Rewards for Multi-Turn and Multi-Step Agentic Tool Use

**Zhen Zhang**[1] **Kaiqiang Song**[2] **Sean Wang**[2] **Yebowen Hu**[3]
**Weixiang Yan**[1] **Chenyang Zhao**[4] **Henry Peng Zou**[5]
**Haoyun Deng**[2] **Sathish Reddy Indurthi**[2] **Shujian Liu**[2]
**Simin Ma**[2] **Xiaoyang Wang**[2] **Xin Eric Wang**[1] **Song Wang**[2]

[1]University of California, Santa Barbara
[2]Zoom Video Communications [3]University of Central Florida
[4]University of California, Los Angeles [5]University of Illinois Chicago

## Abstract

AI agents increasingly solve real-world tasks by reasoning over multi-turn user interactions and invoking external tools. Yet reinforcement learning (RL) remains difficult in this setting: realistic objectives are often open-ended and lack verifiable rewards, RL for multi-turn, multi-step agentic tool use is underexplored, and executable tool environments are costly to build and maintain. 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, CM2 uses sparse reward assignment with dense evaluation criteria, and trains in a scalable LLM-simulated tool environment to avoid engineering large tool sets. Experiments show that CM2 consistently improves over supervised fine-tuning. Starting from an 8B Base model and training on an 8k-example RL dataset, CM2 improves over the SFT counterpart by 8 points on $\tau^2$-Bench, 10 points on BFCL-V4, and 12 points on ToolSandbox. The results match or even outperform similarly sized open-source baselines, including the judging model. CM2 thus offers a scalable recipe for optimizing multi-turn, multi-step tool-using agents without verifiable rewards.

## 1 Introduction

AI agents are emerging as a promising paradigm for complex, real-world tasks [1–3]. By reasoning and invoking external tools such as search engines, databases, proprietary APIs, and compilers, agents can interact with external environments beyond their parametric knowledge [4, 5]. Unlike traditional question answering [6], they must navigate **multi-turn** user dialogues and execute **multi-step** tool-use reasoning [7]. However, training general-purpose agents for such interactions with reinforcement learning (RL) remains challenging.

Three limitations hinder current RL research in this domain. First, existing work largely relies on **verifiable rewards** [8], supervising agents by rule-based final-answer correctness or exact tool-trace matches to ground truth [9]. Such signals often fail in realistic, open-ended scenarios whose objectives include asking clarifying questions, maintaining a helpful tone, or providing suggestions [10–12]. Second, RL for multi-turn and multi-step interactions remains underexplored. Most work relies on supervised fine-tuning (SFT) with synthetic data [13], simulated environments [14, 15], or RL for multi-step reasoning without multi-turn dynamics [16]; these methods provide basic capabilities but often struggle with unseen tools, extended horizons, and richer user interactions. Third, tool-use RL

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

is constrained by environment construction: implementing APIs and maintaining reliable execution environments adds substantial engineering overhead, limiting scale across diverse tools [17, 18].

To address these challenges, we propose CM2 (Checklist Reward for Multi-turn Multi-step Agentic Tool Use), an RL framework for **multi-turn**, **multi-step** tool-use agents **without rule-based verifiable rewards**. Training runs in a **scalable LLM-simulated tool environment** with 5,000 tools, as illustrated in Figure 1.

The core idea of CM2 is to replace verifiable rewards with checklist rewards: each turn's intended behavior is decomposed into fine-grained **binary** evaluation criteria with explicit evidence grounding, dependencies, and weights. This 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 denser trajectory rewards can amplify noise and destabilize optimization. CM2 therefore adopts *Sparse in assignment; Dense in criteria*: rewards are assigned conservatively, while supervision remains informative. To study assignment granularity, we instantiate trajectory-, turn-, and step-level advantage estimation, and introduce reward backfilling to attribute delayed checklist satisfaction to earlier critical steps once dependencies are met, improving credit assignment in long interactions.

To train **scalably** across diverse tools, CM2 performs RL in an **LLM-simulated tool environment** with 5,000+ tools. The simulator replays recorded tool I/O when available and otherwise falls back to LLM-based response simulation, enabling large-scale, execution-free interaction while maintaining contextual consistency and improving training robustness [17, 18].

Empirically, CM2 yields significant improvements across multiple challenging benchmarks. Starting from an 8B base model and training on an 8k-example RL dataset, CM2 improves over the SFT counterpart by 8 points on $\tau^2$-Bench [19, 20], 10 points on BFCL-V4 [21], and 12 points on ToolSandbox [22]. The results match or outperform similarly sized open-source baselines. Comparisons with holistic binary success judgments and structured scalar rubric scores show that the gain comes not merely from sparse trajectory-level assignment or structured criteria, but from replacing noisy scalar scoring with evidence-grounded binary criteria. Overall, CM2 enables robust reinforcement learning without manual environment-specific reward engineering. Checklist rewards provide scalable supervision for multi-turn, multi-step tool use when explicit verifiable rewards are unavailable, offering a practical path toward large-scale optimization of agentic tool use.

**Contributions.** We make four contributions: (1) we formulate **checklist reward learning** for multi-turn, multi-step tool-use agents, decomposing each turn into evidence-grounded binary criteria with dependency and continuation constraints; (2) we identify and validate **sparse reward assignment with dense evaluation criteria**, showing that finer-grained assignment accelerates early learning but destabilizes long-horizon agent RL under judge and simulator noise; (3) we build a scalable RL pipeline with an **LLM-simulated environment** over 5,000 tools, enabling training without executable APIs; and (4) we demonstrate consistent gains over SFT on $\tau^2$-Bench, BFCL-V4, and ToolSandbox, with competitive performance against similarly sized open-source baselines.

## 2 Related Work

### 2.1 Reward for RL

Recent advances have shifted from SFT toward RL to improve the generalization and robustness of agent behavior. A dominant paradigm, Reinforcement Learning with Verifiable Rewards (RLVR) [8], uses deterministic signals for optimization, but open-ended problems often lack ground-truth verifiers. Reinforcement Learning from Human Feedback [23, 24] addresses this by training reward models on human preference data to provide scalar signals [25, 26], yet holistic scalar rewards are often opaque and insufficient for complex multi-step reasoning. Recent criterion-based methods, including Reinforcement Learning with Rubric-based Rewards [10, 11, 27] and Reinforcement Learning from checklist Feedback [12], decompose instruction execution into fine-grained checklist items or criteria evaluated by LLM judges, showing that dense, structured feedback can outperform opaque scalar rewards from standard reward models.

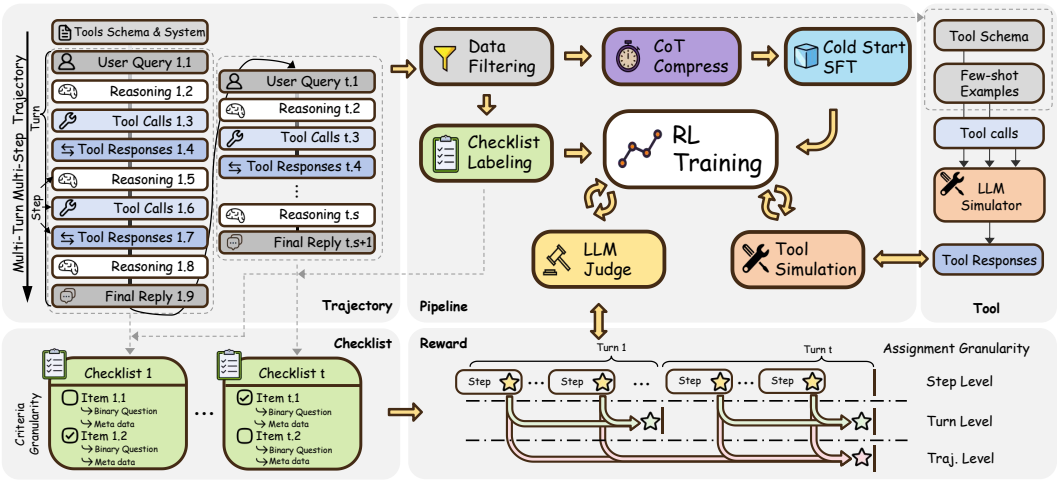



Figure 1: Overview of CM2. From multi-turn, multi-step tool-use trajectories, we perform data filtering, CoT compression, cold-start SFT, and per-turn checklist annotation with evidence-grounded binary criteria and structured metadata. RL runs in an LLM-simulated tool environment: an LLM simulator produces tool responses, and an LLM-as-a-Judge evaluates checklist items for rewards. The bottom panel contrasts dense criteria with sparse assignment at different granularities.

**CM2** belongs to the broader class of LLM-judged non-verifiable reward methods, but differs from standard LLM-as-a-judge, rubric reward, and RLAIF methods in how feedback is structured and used for RL. Instead of holistic scalar scores or opaque learned reward models, CM2 represents each turn with evidence-grounded binary checklist items equipped with dependencies, strictness constraints, and weights, making judging localized and interpretable while enabling reward-assignment studies across multi-turn, multi-step tool-use trajectories.

### 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 [22, 20, 21, 28] 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 [29] first integrated LLM-simulated users into RL loops, but relies on binary outcome rewards and in-domain evaluation data, leaving sparse rewards in long interactions unresolved. In contrast, CM2 uses fine-grained checklist rewards to reinforce correct intermediate steps, mitigating credit assignment issues and enabling more robust dialogue policies and tool-use patterns.

## 3 RL via Checklist Rewards for Agentic Tool Use

This section introduces CM2. We formulate multi-turn, multi-step agentic tool calling in Section 3.1, define two reward granularity dimensions in Section 3.2, describe Checklist reward shaping and labeling in Section 3.3, and detail Checklist-based RL in Section 3.4.

### 3.1 Problem Formulation

As shown in the upper left of Figure 1, we consider a **multi-turn and multi-step** dialogue $\mathcal{D}$ between user $u$ and agent $\pi_\theta$ with tools $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$. The dialogue contains **turns** $\mathcal{D} = \{\tau_1, \tau_2, \ldots, \tau_L\}$, where each turn $\tau_t$ contains **steps** $\tau_t = \{\sigma_{t,1}, \sigma_{t,2}, \ldots, \sigma_{t,M_t}\}$. Each step $\sigma_{t,s}$ has one of three types: **(1) User Query**, marking the initiation of a turn; **(2) Agent Action**, comprising: (i) an internal *Reasoning process* $z_{t,s}$ that precedes an action, and (ii) an explicit *action*

```
{"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

Table 1: Components of a checklist item.

| Component | Description |
|---|---|
| **Evidence** | Pointers to the original trajectory segment(s) used for annotation. |
| **Focus** | Target step type (e.g., tool calls, reasoning, final reply, or tool response), helping the judge localize context. |
| **Question** | A binary checklist question for this item. |
| **Pass/Fail** | Explicit criteria defining when the item passes or fails. |
| **Strictness** | A boolean flag (required_for_next_turn) indicating whether the item must pass before the fixed next user query is issued. |
| **Dependency** | Items that must be satisfied before this item can pass. |
| **Weight** ($w$) | The item's relative weight within a turn, with $\sum_i w_i = 1$. |

$a_{t,s}$, which may be tool calls or a final reply; **(3) Tool Responses**, the output returned by the tool invoked in the preceding agent action.

We use **Interleaved Thinking** [30] to maintain context by preserving thinking from previous turns. The dialogue context $h_{t,s}$ is the complete observable 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 $a_{t,s}$ is a tool call, the environment executes the selected tool $T_i$ with arguments and returns observation $r^{\text{tool}}_{t,s} = T_i(a_{t,s})$, which is appended 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 Checklist reward shaping, we define two orthogonal reward granularity dimensions: *Assignment Granularity* and *Criteria Granularity*. They answer *where* rewards are assigned along the trajectory and *what* criteria are used for evaluation.

**Assignment Granularity** refers to credit assignment across the output sequence and distinguishes sparse from dense rewards. Coarse assignment rewards the final trajectory state as one evaluation unit, while fine assignment distributes rewards across intermediate steps to evaluate incremental progress.

**Criteria Granularity** concerns evaluation specificity. Coarse criteria use a holistic judgment such as task completion or correctness, while fine-grained criteria decompose evaluation into weighted sub-dimensions such as helpfulness, harmfulness, and accuracy.

Although increasing both dimensions gives denser signals, our agentic-RL experiments support a decoupled strategy. Environmental noise makes **coarse-grained assignment** more stable, while **fine-grained criteria** provide the task-specific guidance needed for complex tool-use logic. We therefore adopt *Sparse in assignment; Dense in criteria*.

### 3.3 Checklist Reward Shaping

This section introduces **Checklist-based Reward Shaping**, which provides fine-grained reward signals for multi-turn, multi-step RL in agentic tool use.

**Composition of the Checklist.** As shown in the bottom left of Figure 1, each turn $\tau_t$ is labeled with a *Checklist* $\Gamma_t = \{\gamma_1, \ldots, \gamma_{N_t}\}$. The annotator LLM decomposes the agent's intended turn behavior into fine-grained subtasks. Each Checklist item is a binary question enriched with metadata defining its semantics and constraints (Table 1); one example appears in Figure 2.

**Why Checklist Rewards?** Checklist formulates each criterion as a *binary* pass/fail decision with explicit evidence and conditions, turning open-ended LLM scoring (regression) into a more **stable**

classification-style evaluation. This reduces judge randomness: small stochastic score differences can otherwise be amplified by per-batch return or advantage normalization, changing within-batch rankings and producing unstable or contradictory gradients [12, 31]. The structured metadata also makes the Checklist **interpretable**, enabling automated, consistent, lower-noise evaluation across turns.

**Post-hoc Checklist Annotation.** We label Checklists by *post-hoc* structuring existing multi-turn, multi-step tool-use trajectories rather than writing them from scratch. For each turn, an LLM (i) infers the turn-level intent and required outcomes from the user query and assistant/tool traces, and (ii) decomposes them into concise **binary, observable** Checklist items grounded in the trajectory. Each trajectory costs about $0.1 on average, making checklist labeling scalable relative to training and manual annotation costs. The prompt and annotation details are provided in Appendix B.1.

**Rollout and Reward Computation.** During rollout, at each step within turn $\tau_t$, we query a judge LLM with the trajectory prefix and that turn's checklist items. The judge returns a Boolean label for each item, indicating whether the partial trajectory satisfies it. After the agent produces the final user-visible response, we enforce strictness constraints: if all strict items pass, we issue the next reference user query; otherwise, the rollout terminates early.

### 3.4 Checklist-based RL Optimization

RL algorithms based on Group Relative Policy Optimization (GRPO) are typically formulated around outcome rewards. Our Checklist framework, however, extracts dense reward signals down to individual steps. To study *Assignment Granularity*, we instantiate three advantage-estimation variants: (i) Trajectory-level, (ii) Turn-level, and (iii) Step-level, which differ in reward assignment and advantage computation.

#### 3.4.1 Checklist-based Reward

Let $x_s$ denote the state before step $s$ and $x_{s+1}$ the state after it. For dialogue $i$, turn $t$, and checklist item $c$, let $\mathrm{Sat}^{(i)}_{t,c}(x_s) \in \{0,1\}$ indicate whether $\gamma^{(i)}_{t,c}$ is satisfied in state $x_s$. Let

$$\mathrm{Dep}_{t,c} = \{\, c' \mid \gamma_{t,c'} \text{ is a dependency (prerequisite) of } \gamma_{t,c} \,\}, \tag{1}$$

be the dependency set of $\gamma^{(i)}_{t,c}$. When $\gamma^{(i)}_{t,c}$ switches from unsatisfied to satisfied at step $s$ and all dependencies are satisfied in pre-step state $x_s$, we assign a binary reward:

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

Since satisfying an item may require multiple steps, we *backfill* reward to every earlier step where all dependencies were already satisfied:

$$\tilde{r}^{(i)}_{t,s,c} = \mathbf{1}\Big[ \underbrace{\prod_{c' \in \mathrm{Dep}_{t,c}} \mathrm{Sat}^{(i)}_{t,c'}(x_s) = 1}_{\text{all deps. satisfied before } s} \wedge \underbrace{\mathrm{Sat}^{(i)}_{t,c}(x_s) = 0}_{\text{unsatisfied before } s} \wedge \underbrace{\exists\, u \geq s \text{ s.t. } \mathrm{Sat}^{(i)}_{t,c}(x_{u+1}) = 1}_{\text{satisfied after } s} \Big]. \tag{3}$$

We use *backfilled* reward only for step-level advantage.

#### 3.4.2 Trajectory-level Advantage

Given rollout $D^{(i)} = \{\tau_1, \ldots, \tau_{L^{(i)}}\}$, we aggregate Checklist rewards across turns, steps, and items:

$$R^{(i)} \;=\; \frac{1}{L^{(i)}} \sum_{t=1}^{L^{(i)}} \sum_{s} \sum_{c} w_{t,c} \cdot r^{(i)}_{t,s,c}, \tag{4}$$

where $s$ ranges over steps in turn $t$, $c$ over checklist items for turn $t$, and $R^{(i)} \in [0,1]$ because each item flips once and $\sum_c w_{t,c} = 1$. For $G$ rollouts of the same prompt, the trajectory-level advantage is

$$A^{(i)}_{\mathrm{traj}} \;=\; \frac{R^{(i)} \;-\; \mathrm{mean}(\{R^{(i)}\}_{i=1}^{G})}{F_{\mathrm{norm}}(\{R^{(i)}\}_{i=1}^{G})}. \tag{5}$$

#### 3.4.3 Turn-level Advantage

Turn-level advantage aggregates Checklist rewards *within* each turn and normalizes them across $G$ rollouts of the same question:

$$R_t^{(i)} \;=\; \sum_s \sum_c w_{t,c} \cdot r_{t,s,c}^{(i)}, \quad A_{\text{turn},t}^{(i)} \;=\; \frac{R_t^{(i)} \;-\; \text{mean}\Big(\{R_t^{(i)}\}_{i=1}^{G}\Big)}{F_{\text{norm}}\Big(\{R_t^{(i)}\}_{i=1}^{G}\Big)}. \tag{6}$$

Here $s$ ranges over steps in turn $t$, $c$ over checklist items for turn $t$, and $R_t^{(i)} \in [0, 1]$.

#### 3.4.4 Step-level Advantage

Step-level advantage first computes an item baseline, then normalizes each backfilled item reward:

$$\begin{aligned} b_{t,c} &= \frac{1}{G} \sum_{i=1}^{G} \mathbb{I}\big[\exists s' \text{ s.t. } r_{t,s',c}^{(i)} = 1\big], \\ A_{t,s,c}^{(i)} &= \frac{\tilde{r}_{t,s,c}^{(i)} \;-\; b_{t,c}}{F_{\text{norm}}(\{\mathbb{I}\big[\exists s' \text{ s.t. } r_{t,s',c}^{(i)} = 1\big]\}_{i=1}^{G})}. \end{aligned} \tag{7}$$

At step $(t, s)$, we aggregate applicable item-wise advantages with checklist weights:

$$\begin{aligned} A_{\text{step},t,s}^{(i)} &= \frac{\sum_{c \in \mathcal{E}_{t,s}^{(i)}} w_{t,c}\, A_{t,s,c}^{(i)}}{\sum_{c \in \mathcal{E}_{t,s}^{(i)}} w_{t,c}}, \\ \mathcal{E}_{t,s}^{(i)} &= \big\{c \,\big|\, \prod_{c' \in \text{Dep}_{t,c}} \text{Sat}_{t,c'}^{(i)}(x_s) = 1 \wedge \text{Sat}_{t,c}^{(i)}(x_s) = 0\big\}. \end{aligned} \tag{8}$$

$\mathcal{E}_{t,s}^{(i)}$ is the set of checklist items eligible at step $s$: all dependencies are satisfied and item $c$ is not yet satisfied.

## 4 Training Pipeline

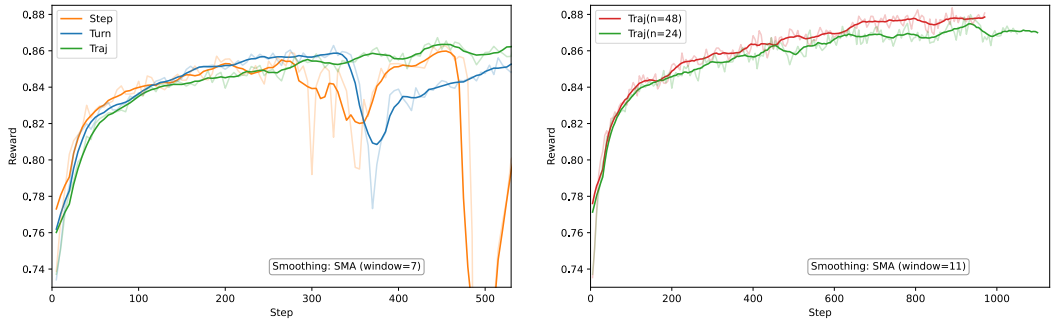
This section outlines the CM2 training pipeline: data filtering, Chain-of-Thought (CoT) compression, cold-start SFT, checklist labeling, tool simulation, and LLM-as-a-Judge-guided RL. Figure 1 shows the workflow.

### 4.1 Data Filtering

We start from the tool-calling subset of NVIDIA/NEMOTRON-POST-TRAINING-DATASET-V1 [32, 33], containing 310k synthetic tool-use dialogues across single-turn, multi-turn, and multi-step settings and domains such as shopping, financial analysis, and web search. Because all samples are LLM-distilled and noisy, we apply two-stage filtering: (1) **Rule-based filtering** removes structural and formatting violations (Appendix C.2); (2) **LLM-based filtering** uses GPT-5[34] to discard deeper semantic or reasoning errors (Appendix B.2).

We also conduct additional experiments on a 5k in-domain synthetic multi-turn tool-use set aligned with the $\tau^2$-Bench airline and retail tool functions; we use this set as provided without additional cleaning.

**Data statistics.** Rule-based filtering reduces the dataset from 310k to 280k examples, and LLM-based filtering narrows it to 30k high-quality samples. We randomly sample **8k examples** for cold-start SFT; the remaining 22k form the RL candidate pool, from which we exclude simpler cases such as 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.




(a) Validation reward curves under different advantage assignment granularities.

(b) Validation reward curves under different group sizes ($n$).

Figure 3: Comparisons under different settings.

### 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. GPT-5 rewrites the thinking content into a shorter form while preserving key planning and decisions (Appendix B.3). The compressed datasets are $\mathcal{D}_{\text{CS}}$ (cold-start SFT) and $\mathcal{D}_{\text{RL}}$ (RL training).

Finally, we fine-tune a 8B base model on the $\mathcal{D}$CS. Hyperparameters and other training details are provided in Appendix C.

### 4.3 Tool Simulation and LLM-as-a-Judge

Because the trajectories are synthetic, no executable environment is available during RL. To avoid building and maintaining 5,000+ tools, we implement a hybrid simulator. For each tool invocation, the simulator first performs an **exact match** against the original tool name and arguments; if matched, it returns the recorded response. Otherwise, it uses **LLM-based simulation**: an LLM receives in-dialogue tool I/O exemplars as few-shot context and generates a trajectory-consistent response, enabling scalable, execution-free interaction. For **LLM-as-a-Judge**, we prompt an LLM at each step to answer checklist questions and aggregate rewards as in Section 3. The judging prompt is in Appendix B.4. We use a 30B model with 3B active parameters for both simulation and judging to balance quality and throughput; later experiments show that this lightweight judge still enables highly competitive, sometimes surpassing, results.

### 4.4 Checklist Labeling and RL Training

Following Section 3.3, GPT-5 annotates a per-turn Checklist for each dialogue (Appendix B.1). We then optimize from the cold-start SFT checkpoint using VeRL-based GRPO and the multi-level advantage comparison in Section 3.4. The RL model is trained on 64 GPUs for 680 hours; implementation details and hyperparameters are in Appendix C.1.

## 5 Results

### 5.1 Effect of Allocation Granularity

Figure 3a compares validation reward curves under different assignment granularities. Finer allocation improves faster early on: step-level outperforms turn-level, which outperforms trajectory-level. As training continues, however, finer granularities collapse earlier and more severely, while trajectory-level advantages stay stable and keep improving.

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 do not remove stochasticity. Finer assignment injects residual noise more often and group-relative normalization can amplify it into higher-variance or misleading gradients. This motivates *Sparse in assignment;*

Table 2: Results on $\tau^2$-Bench and ToolSandbox. For $\tau^2$-Bench, we average four evaluations and retain thinking content to match training. ToolSandbox breakdowns by scenario and tool augmentation are in Table 5 in the appendix. All RL variants use trajectory-level GRPO and the same LLM judge on $\mathcal{D}_{\text{RL}}$, differing only in reward formulation. The *Ours* (from **8B Reasoning**) rows use an in-domain 5k synthetic set aligned with the $\tau^2$-Bench; they are evaluated only on $\tau^2$-Bench (– for ToolSandbox).

| | $\tau^2$-Bench | | | | ToolSandbox |
|---|---|---|---|---|---|
| **Model / Method** | **Airline** | **Retail** | **Telecom** | **Avg.** | **Overall** |
| *Open-source Baselines* | | | | | |
| 30B-A3B Instruct Baseline | 32.50 | 50.88 | 12.72 | 32.03 | 65.24 |
| 8B Reasoning Baseline | 30.00 | 43.64 | 22.37 | 32.00 | 65.47 |
| *Ours* (from **8B Base**) | | | | | |
| Cold-start SFT on $\mathcal{D}_{\text{CS}}$ | 25.50 | 18.42 | 11.84 | 18.59 | 56.19 |
| $\hookrightarrow$ SFT on $\mathcal{D}_{\text{RL}}$ | 23.50 | 19.52 | 12.06 | 18.36 | 55.32 |
| $\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (Holistic Binary) | 21.50 | 17.37 | 13.02 | 17.30 | 58.29 |
| $\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (Scalar Rubric) | 25.00 | 25.49 | 14.14 | 21.54 | 65.33 |
| $\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (CM2) | 27.00 | 36.40 | 16.89 | **26.76** | **68.20** |
| *Ours* (from **8B Reasoning**) | | | | | |
| SFT on in-domain 5k | 30.00 | 44.74 | 23.68 | 32.81 | – |
| RL on in-domain 5k (CM2-$\tau^2$) | 33.00 | 54.17 | 37.00 | **41.39** | – |

Table 3: Results on BFCL-V4 (Multi-Turn and Web Search). All methods use the same trajectory-level GRPO and LLM judge on $\mathcal{D}_{\text{RL}}$; only the reward formulation differs: holistic binary success, structured scalar rubric scoring, or weighted binary checklist judging.

| | Multi-Turn | | | | | Web Search | | |
|---|---|---|---|---|---|---|---|---|
| **Model / Method** | **Base** | **Miss Func** | **Miss Param** | **Long Ctx** | **Overall** | **Base** | **No Snippet** | **Overall** |
| *Open-source Baselines* | | | | | | | | |
| 30B-A3B Instruct Baseline | 45.0 | 28.0 | 21.0 | 42.5 | 34.25 | 24.00 | 17.00 | 20.50 |
| 8B Reasoning Baseline | 42.5 | 38.5 | 31.5 | 35.5 | 37.00 | 19.00 | 11.00 | 15.00 |
| *Ours* (from **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 $\mathcal{D}_{\text{RL}}$ (Holistic Binary) | 34.5 | 29.5 | 23.5 | 29.5 | 29.25 | 23.00 | 8.00 | 15.50 |
| $\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (Scalar Rubric) | 36.0 | 31.5 | 35.5 | 31.0 | 33.50 | 30.00 | 12.00 | 21.00 |
| $\hookrightarrow$ RL on $\mathcal{D}_{\text{RL}}$ (CM2) | 44.5 | 32.0 | 35.0 | 34.5 | **36.50** | 41.00 | 14.00 | **27.50** |

*Dense in criteria*: fine-grained criteria keep supervision informative, while coarser reward assignment averages out residual noise for stability.

### 5.2 Effect of Group Size

Figure 3b shows that larger group size ($G$=48 vs. $G$=24) consistently improves trajectory-level Checklist rewards. For multi-turn, multi-step trajectories, larger $G$ gives more samples per turn, yielding lower-variance advantages and more reliable gradients.

### 5.3 Results on Benchmarks

We evaluate CM2 with the final configuration (trajectory-level advantage, $G = 48$) on three multi-turn, multi-step tool-use benchmarks: $\tau^2$-Bench, BFCL-V4, and ToolSandbox. We compare against SFT counterparts, similarly sized open-source models, and two alternative LLM-judge rewards that share CM2's rollout, simulator, and GRPO setup but use either holistic binary success or structured scalar rubric scoring.

**$\tau^2$-Bench Benchmark.** Table 2 summarizes $\tau^2$-Bench results. For each question, we run evaluation four times and report average accuracy. Starting from an **8B Base** checkpoint, CM2 outperforms SFT and both alternative rewards: compared with SFT on $\mathcal{D}_{\text{RL}}$, it improves from 18.36 to 26.76, and still gains 5.22 points over the stronger Scalar Rubric baseline. Holistic Binary underperforms SFT, suggesting that a single trajectory-level success label is too coarse for long-horizon tool interaction.

However, our RL training uses a 10k maximum context and up to 30 turns, whereas $\tau^2$-Bench can require >30k context and up to 200 turns; under this mismatch, CM2 lags behind the 30B-A3B Instruct and 8B Reasoning baselines. In-domain RL on the 5k synthetic set substantially improves average performance and surpasses open-source baselines, indicating that CM2 is especially effective with in-domain data.

**BFCL Benchmark.** Table 3 summarizes BFCL-V4 results on Multi-Turn and Web Search. On Multi-Turn, CM2 trained on $\mathcal{D}_{\text{RL}}$ reaches 36.50 overall accuracy, improving over SFT on $\mathcal{D}_{\text{RL}}$ by 9.75 points and Scalar Rubric by 3.00 points. On Web Search, it also performs best, improving over SFT by 14.00 points and Scalar Rubric by 6.50 points. Compared with open-source baselines, CM2 beats the 30B-A3B Instruct baseline on Multi-Turn, is comparable to the 8B Reasoning baseline, and significantly surpasses both on Web Search.

**ToolSandbox Benchmark.** Table 2 reports ToolSandbox performance. RL on $\mathcal{D}_{\text{RL}}$ with Checklist Reward improves over both SFT variants and alternative rewards, raising the overall score by 12.88 points over SFT on $\mathcal{D}_{\text{RL}}$ and 2.87 points over Scalar Rubric. Table 5 in the appendix shows broad gains across scenario categories and tool augmentations, especially multi-turn and multi-tool settings. CM2 also outperforms the open-source models, including the judge model.

**Reward Pipeline Diagnostics.** Table 4 reports checklist structure, simulator composition, and repeated-judge stability. Items per turn and strict item ratio are averaged over annotated turns; normalized weight entropy is computed as $-\sum_i w_i \log w_i / \log N_t$ for each turn and then averaged, so higher values mean the reward mass is broadly distributed rather than dominated by a few criteria. Exact tool-call replay ratio is the fraction of tool invocations whose tool name and arguments exactly match a recorded trajectory response; its low value indicates that reward judging mostly relies on simulated trajectory evidence rather than deterministic replay. To isolate judge noise from policy variance, after training we fix held-out validation trajectories, judge each trajectory 8 times with each reward formulation, compute per-trajectory reward variance, and average across trajectories. Scalar Rubric has a repeated-judge variance of 0.00613, while Checklist Reward reduces it to 0.00015, a roughly 41× reduction. This large reduction directly demonstrates the stability of Checklist Reward under repeated judging: for the same fixed trajectories, it produces substantially less reward fluctuation than scalar rubric scoring. The result supports the main mechanism of Checklist Reward: decomposing a long trajectory into local binary criteria avoids asking the judge to make a single scalar trade-off across turns, tool calls, observations, and final responses.

Table 4: Reward-construction diagnostics: checklist structure, simulator composition, and repeated-judge stability. Repeated-judge variance is measured after training by fixing held-out validation trajectories, judging each multiple times, computing per-trajectory reward variance, and averaging across trajectories.

| Aspect | Diagnostic | Value |
|---|---|---|
| Checklist Structure | Items per turn | 6.3 |
| | Strict item ratio | 0.66 |
| | Normalized weight entropy | 0.954 |
| Tool Simulation | Exact tool-call replay ratio | 0.16 |
| Reward Stability | Variance, Scalar Rubric | 0.00613 |
| | Variance, Checklist Reward | 0.00015 |

**Summary.** Checklist Reward achieves the best aggregate performance across all three benchmarks. Comparisons with Holistic Binary and Scalar Rubric show that the gain is not simply from trajectory-level GRPO or using an LLM judge, since all variants share the same rollout, simulator, judge, and trajectory-level assignment. Holistic Binary collapses the interaction into one success label and loses partial progress across turns and tool calls. Scalar Rubric exposes structured dimensions but still requires scalar trade-offs over long trajectories. Checklist Reward instead decomposes trajectories into dependency-aware binary criteria localized to turns, tool observations, and final responses, then aggregates them with weights. This keeps reward tied to the final trajectory while reducing scalar-scoring ambiguity, supporting *Sparse in assignment; Dense in criteria*.

## 6 Conclusion

CM2 is a scalable RL framework for multi-turn, multi-step tool-using agents that replaces verifiable rewards with checklist rewards: fine-grained, binary, evidence-grounded criteria that make LLM judging more stable and interpretable. With sparse reward assignment, dense evaluation criteria, and an LLM-simulated tool environment, CM2 improves over supervised fine-tuning across benchmarks and generalizes better to complex, long-horizon tool-use behaviors where verifiable rewards are unavailable.

## Acknowledgments and Disclosure of Funding

This work was completed during Zhen Zhang's internship at Zoom Video Communications.

## References

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

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

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

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

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

[6] Ehsan Kamalloo, Nouha Dziri, Charles Clarke, and Davood Rafiei. 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, 2023.

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

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

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

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

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

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

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

[14] Wujiang Xu, Wentian Zhao, Zhenting Wang, Yu-Jhe Li, Can Jin, Mingyu Jin, Kai Mei, Kun Wan, and Dimitris N Metaxas. Epo: Entropy-regularized policy optimization for llm agents reinforcement learning. *arXiv preprint arXiv:2509.22576*, 2025.

[15] Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

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

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

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

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

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

[21] 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.

[22] Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. 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, 2025.

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

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

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

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

[27] Aditya Pathak, Rachit Gandhi, Vaibhav Uttam, Arnav Ramamoorthy, Pratyush Ghosh, Aaryan Raj Jindal, Shreyash Verma, Aditya Mittal, Aashna Ased, Chirag Khatri, et al. 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, 2025.

[28] Yang Zhou, Mingyu Zhao, Zhenting Wang, Difei Gu, Bangwei Guo, Ruosong Ye, Ligong Han, Can Jin, and Dimitris N Metaxas. Mˆ 3-bench: Multi-modal, multi-hop, multi-threaded tool-using mllm agent benchmark. *arXiv preprint arXiv:2511.17729*, 2025.

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

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

[31] Can Jin, Yang Zhou, Qixin Zhang, Hongwu Peng, Di Zhang, Marco Pavone, Ligong Han, Zhang-Wei Hong, Tong Che, and Dimitris N Metaxas. Your reward function for rl is your best prm for search: Unifying rl and search-based tts. *arXiv preprint arXiv:2508.14313*, 2025.

[32] Dhruv Nathawani, Igor Gitman, Somshubra Majumdar, Evelina Bakhturina, Ameya Sunil Mahabaleshwarkar, , Jian Zhang, and Jane Polak Scowcroft. Nemotron-Post-Training-Dataset-v1, July 2025. Version 1.0, NVIDIA.

[33] 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, 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, Katherine Luna, Leon Derczynski, Nikki Pope, Eileen Long, Seth Schneider, Guillermo Siman, Tomasz Grzegorzek, Pablo Ribalta, Monika Katariya, 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, Eric Chung, and Chris Alexiuk. Llama-nemotron: Efficient reasoning models, 2025.

[34] OpenAI. Introducing GPT-5. `https://openai.com/index/introducing-gpt-5/`, 2025. Accessed: 2026-01-05.

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

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

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

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

# A More Related Work on LLM-Simulated Tool Environments

The key limitation in extending RL to tool-use domains is the engineering overhead of maintaining real-world APIs [22, 5]. LLM-based environment simulation has therefore become the dominant alternative. SynthAgent [35] proposes a fully synthetic supervision framework for web agents with trajectory optimization to enhance performance; ToolEmu [18] uses LLM-simulated sandboxes to identify risky behaviors without actual tool infrastructure. Simia [36] shows that powerful LLMs can simulate environment feedback from tool definitions and interaction history, while Generalist Tool Model (GTM) [37] introduces a specialized 1.5B model to simulate over 20,000 tools. In contrast, CM2 scales to arbitrary tools, enabling large-scale training across diverse domains and synthetic edge cases for improved robustness.

# B Prompts

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

## B.1 Prompt of Checklist Labeling

The prompt for Checklist Annotation is shown in Prompt 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.

## B.2 Prompt of LLM-based Filtering

The prompt for LLM-based Filtering is shown in Prompt 2 We use GPT-5 [34] 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.

## B.3 Prompt of CoT Compression

The prompt for CoT Compression is shown in Prompt 3. We use the default setting of GPT-5

## B.4 Prompt of LLM Judge

The prompt for LLM judge is shown in Prompt 4

## B.5 Prompt of Alternative Reward Evaluators

The prompts for the alternative LLM-judge reward formulations (Holistic Binary and Scalar Rubric) are shown in Prompt 5 and Prompt 6.

## B.6 Prompt of Tool Simulation

The prompt for tool simulation is shown in Prompt 7.

# C 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:

$$\begin{aligned}
\texttt{<think>} &\leftarrow \text{avg("think", "begin")} \\
\texttt{</think>} &\leftarrow \text{avg("think", "finish")} \\
\texttt{<tool_call>} &\leftarrow \text{avg("tool", "call", "start")} \\
\texttt{</tool_call>} &\leftarrow \text{avg("tool", "call", "end")} \\
\texttt{<im_start>} &\leftarrow \text{avg("role", "enter")} \\
\texttt{<im_end>} &\leftarrow \text{avg("role", "exit")}
\end{aligned}$$

### C.1 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 [38]. 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. The Holistic Binary and Scalar Rubric formulations use the same rollout, simulator, GRPO hyperparameters, LLM judge, and trajectory-level assignment as Checklist Reward.

### C.2 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>).

### C.3 Additional Benchmark Details

Table 5: Detailed ToolSandbox results by scenario category and tool augmentation.

| 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* | | | | | | | | | | | | | | | | |
| 30B-A3B Instruct Baseline | 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 |
| 8B Reasoning Baseline | 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 **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 |

Prompt 1: Checklist Labeling

```
# 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_calls|assistant.content.th⌋
↪  inking|assistant.content.user_visible_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.user_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
]
```

Prompt 2: LLM-based Filtering

```
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
}
```

Prompt 3: CoT Compression

```
# 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": "..."
  },
  ...
]
```

Prompt 4: LLM-as-a-Judge

```
# 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_checklist['pass_condition']}
**Failure examples:** {this_turn_checklist['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
}
```

Prompt 5: Holistic Binary Evaluator

```
# Role
You are a precise outcome-reward evaluator for multi-turn, multi-step
↪  tool-use dialogues. Your sole task is to determine whether a candidate
↪  conversation achieves the same task-level outcome as the reference
↪  conversation.

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

# Evaluation Target
You will be given:
1. Candidate tool schemas.
2. A reference conversation.
3. A candidate conversation generated by an assistant.

Judge whether the candidate assistant satisfies the user's request and
↪  reaches the same key outcome as the reference conversation.
```

```
# Candidate Tools
{{tools_schema}}

# Reference Conversation
{{reference_messages_str}}

# Candidate Conversation to Evaluate
{{candidate_messages_str}}

# Evaluation Rules
1. Evaluate the task-level outcome, not surface-form similarity.
2. Do not require exact matching of the reference tool-call sequence.
3. Do not require exact matching of the wording of the reference final reply.
4. The candidate may use a different valid tool path, different intermediate
↪  steps, or more or fewer tool calls.
5. The candidate must satisfy the user's actual request and preserve all key
↪  facts, constraints, files, parameters, decisions, and results
↪  demonstrated by the reference.
6. Treat tool responses as the source of truth. Penalize final answers that
↪  are unsupported by, or contradict, the observed tool responses.
7. Penalize invalid tool calls, wrong tool names, wrong tool arguments,
↪  missing required tool use, skipped required actions, hallucinated facts,
↪  premature termination, and incomplete final replies.
8. Ignore private thinking style. Evaluate only observable user-visible
↪  replies, valid tool calls, and tool responses.

# Special Rule for Tool Calls
If a tool call appears only in a private thinking field and not in the
↪  tool_call field, that tool call is invalid and must not be credited.

If a tool response reports an error, the corresponding tool call in the
↪  latest assistant message is unsuccessful. Do not credit it as completing
↪  the intended action unless the candidate recovers with a later valid
↪  action.

# Evaluation Process (Align each step to a JSON output field)
1. high_level_understanding_of_the_task:
   - Briefly summarize the user's task and the key outcome shown by the
   ↪  reference conversation.

2. analysis_of_candidate_behavior:
   - Analyze whether the candidate uses appropriate tools, follows the user's
   ↪  requirements, and grounds claims in tool responses.

3. analysis_of_key_outcome_match:
   - Determine whether the candidate achieves the same task-level outcome as
   ↪  the reference, allowing valid alternative tool paths and different
   ↪  wording.

4. analysis_of_errors:
   - Identify any major errors, including invalid tool calls, wrong
   ↪  parameters, missing required actions, hallucinations, contradictions
   ↪  with tool results, or incomplete final replies.

5. success:
   - Return true ONLY IF:
     * The candidate satisfies the user's actual request.
     * The candidate achieves the same key outcome as the reference.
     * The candidate has no major tool-use, grounding, factual, or
     ↪  final-answer error.
   - Otherwise return false.

# Output Format
```

```
Return ONLY a single JSON object with exactly these keys:
{
  "high_level_understanding_of_the_task": str,
  "analysis_of_candidate_behavior": str,
  "analysis_of_key_outcome_match": str,
  "analysis_of_errors": str,
  "success": bool
}
```

Prompt 6: Scalar Rubric Evaluator

```
# Role
You are a precise rubric-based outcome-reward evaluator for multi-turn,
↪  multi-step tool-use dialogues. Your sole task is to score how well a
↪  candidate conversation achieves the same task-level outcome as the
↪  reference conversation using a structured rubric.

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

# Evaluation Target
You will be given:
1. Candidate tool schemas.
2. A reference conversation.
3. A candidate conversation generated by an assistant.

Score the candidate assistant against the reference conversation using the
↪  rubric dimensions below. The reference conversation defines the intended
↪  task outcome and the key facts, constraints, tool-use behavior, and final
↪  answer quality expected for the task.

# Candidate Tools
{{tools_schema}}

# Reference Conversation
{{reference_messages_str}}

# Candidate Conversation to Evaluate
{{candidate_messages_str}}

# Evaluation Rules
1. Evaluate the task-level outcome, not surface-form similarity.
2. Do not require exact matching of the reference tool-call sequence.
3. Do not require exact matching of the wording of the reference final reply.
4. The candidate may use a different valid tool path, different intermediate
↪  steps, or more or fewer tool calls.
5. The candidate must satisfy the user's actual request and preserve all key
↪  facts, constraints, files, parameters, decisions, and results
↪  demonstrated by the reference.
6. Treat tool responses as the source of truth. Penalize final answers that
↪  are unsupported by, or contradict, the observed tool responses.
7. Penalize invalid tool calls, wrong tool names, wrong tool arguments,
↪  missing required tool use, skipped required actions, hallucinated facts,
↪  premature termination, and incomplete final replies.
8. Ignore private thinking style. Evaluate only observable user-visible
↪  replies, valid tool calls, and tool responses.

# Special Rule for Tool Calls
If a tool call appears only in a private thinking field and not in the
↪  tool_call field, that tool call is invalid and must not be credited.
```

```
If a tool response reports an error, the corresponding tool call in the
↪  latest assistant message is unsuccessful. Do not credit it as completing
↪  the intended action unless the candidate recovers with a later valid
↪  action.

# Rubric Dimensions
Score each dimension from 1 to 5. Each dimension has a fixed weight used to
↪  compute the scalar rubric reward.

1. task_completion (weight: 0.30):
   - Whether the candidate satisfies the user's actual request and reaches
   ↪  the key outcome shown by the reference.

2. tool_selection_and_calling (weight: 0.20):
   - Whether the candidate selects appropriate tools, calls them when needed,
   ↪  avoids unnecessary or invalid tool calls, and respects the candidate
   ↪  tool schemas.

3. argument_and_constraint_grounding (weight: 0.15):
   - Whether tool arguments, intermediate decisions, and constraints are
   ↪  grounded in the user request, dialogue context, and reference-relevant
   ↪  facts.

4. observation_grounding (weight: 0.25):
   - Whether the candidate correctly uses tool responses, preserves returned
   ↪  facts, and avoids hallucinating or contradicting tool observations.

5. interaction_and_final_response_quality (weight: 0.10):
   - Whether the candidate handles multi-turn interaction appropriately,
   ↪  recovers from intermediate observations when needed, and gives a final
   ↪  user-visible answer that is complete, accurate, and consistent with
   ↪  the tool results.

# Dimension Scoring Guide
5: Fully satisfies the dimension with no important issue.
4: Mostly satisfies the dimension, with only minor omissions or harmless
↪  deviations.
3: Partially satisfies the dimension, but has a noticeable omission, weak
↪  grounding, or moderate tool-use/final-answer issue.
2: Mostly fails the dimension because of major errors, missing required
↪  actions, unsupported claims, or incorrect use of tools or observations.
1: Completely fails the dimension, is irrelevant, terminates too early, or
↪  contradicts the reference outcome or tool responses.

# Weighted Score
Compute a weighted_score from 0 to 1 using:
weighted_score =
0.30 * ((task_completion - 1) / 4)
+ 0.20 * ((tool_selection_and_calling - 1) / 4)
+ 0.15 * ((argument_and_constraint_grounding - 1) / 4)
+ 0.25 * ((observation_grounding - 1) / 4)
+ 0.10 * ((interaction_and_final_response_quality - 1) / 4).

where each dimension score is first normalized from the 1--5 scale to the
↪  0--1 scale using (score - 1) / 4.

Major failures in task completion, invalid required tool use, or
↪  contradictions with tool responses should cap weighted_score even if some
↪  other dimensions are acceptable.

# Evaluation Process (Align each step to a JSON output field)
1. high_level_understanding_of_the_task:
   - Briefly summarize the user's task and the key outcome shown by the
   ↪  reference conversation.
```

```
2. rubric_analysis:
   - Briefly justify each dimension score using observable candidate
   ↪  behavior, valid tool calls, and tool responses.

3. analysis_of_key_outcome_match:
   - Determine whether the candidate achieves the same task-level outcome as
   ↪  the reference, allowing valid alternative tool paths and different
   ↪  wording.

4. analysis_of_errors:
   - Identify any major or minor errors, including invalid tool calls, wrong
   ↪  parameters, missing required actions, hallucinations, contradictions
   ↪  with tool results, or incomplete final replies.

5. scores:
   - Return the five dimension scores as integers from 1 to 5.
   - Return weighted_score as a number from 0 to 1, rounded to two decimal
   ↪  places.

# Output Format
Return ONLY a single JSON object with exactly these keys:
{
  "high_level_understanding_of_the_task": str,
  "rubric_analysis": str,
  "analysis_of_key_outcome_match": str,
  "analysis_of_errors": str,
  "scores": {
    "task_completion": int,
    "tool_selection_and_calling": int,
    "argument_and_constraint_grounding": int,
    "observation_grounding": int,
    "interaction_and_final_response_quality": int,
    "weighted_score": number
  }
}
```

Prompt 7: Tool Simulation

```
# 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.
```

## NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes].

   Justification: The abstract and introduction state that CM2 uses checklist rewards for multi-turn, multi-step tool-use RL without verifiable rewards, and the experiments evaluate this claim on $\tau^2$-Bench, BFCL-V4, and ToolSandbox. The reported gains and scope match the pipeline and benchmark results in Section 4 and Tables 2–3.

2. **Limitations**

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

   Answer: [Yes].

   Justification: The results section discusses important limitations, including context-length and turn-count mismatch on $\tau^2$-Bench and residual noise from LLM-based simulation and judging. The reward diagnostics further quantify simulator composition and repeated-judge variance.

3. **Theory assumptions and proofs**

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

   Answer: [N/A].

   Justification: The paper does not present theoretical theorems or formal proof claims. The mathematical content defines rewards and advantage estimators used by the empirical method.

4. **Experimental result reproducibility**

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

   Answer: [Yes].

   Justification: Sections 4 and 3.4 describe data construction, filtering, checklist labeling, tool simulation, reward computation, and GRPO variants. Appendix C and Appendix C.1 provide training hyperparameters, prompts, and additional benchmark details.

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

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

   Answer: [Yes].

   Justification: We include the code and data needed to reproduce the main experimental results in the supplementary material, together with prompts, data-processing details, and training/evaluation settings. The paper also describes the data sources, filtering procedure, model settings, and hyperparameters in Sections 4 and Appendix C–C.1.

6. **Experimental setting/details**

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

   Answer: [Yes].

   Justification: The paper reports dataset filtering statistics, training splits, validation size, model backbone, evaluator benchmarks, reward variants, group sizes, optimizer choices, learning rates, batch sizes, KL coefficient, and compute setup in Section 4 and Appendix C–C.1.

7. **Experiment statistical significance**

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

   Answer: [No].

   Justification: The paper averages four $\tau^2$-Bench evaluations and reports repeated-judge variance diagnostics, but it does not report error bars, confidence intervals, or formal significance tests for all benchmark results. The computational cost of RL training and repeated multi-benchmark evaluation limited the number of full independent runs.

8. **Experiments compute resources**

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

   Answer: [Yes].

   Justification: Section 4 states that RL training uses 64 GPUs for 680 hours, and Appendix C states that cold-start SFT uses 8 H100 GPUs. The appendix also reports the main batch sizes and hyperparameters needed to contextualize compute requirements.

9. **Code of ethics**

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

   Answer: [Yes].

   Justification: The work uses previously released synthetic tool-use data and public evaluation benchmarks, and does not involve new human-subject data collection. We are not aware of any aspect of the study that deviates from the NeurIPS Code of Ethics.

10. **Broader impacts**

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

    Answer: [No].

    Justification: The paper focuses on technical methods and empirical evaluation for tool-use RL, and does not include a dedicated broader-impact discussion. Potential benefits include more reliable tool-using agents, while potential risks include misuse or overreliance on agents trained with simulated tools and LLM judges.

11. **Safeguards**

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

    Answer: [N/A].

    Justification: The paper does not currently release a new high-risk pretrained model, image generator, or scraped dataset. The experiments are conducted on synthetic tool-use data and public benchmarks, with no new public asset release described in the submission.

12. **Licenses for existing assets**

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

    Answer: [Yes].

    Justification: The paper cites the datasets, benchmarks, models, and frameworks it builds on, including Nemotron-Post-Training-Dataset-v1, $\tau^2$-Bench, BFCL-V4, ToolSandbox, GPT-5, LLaMAFactory, and VeRL/GRPO-related work. The work respects these assets as research dependencies and does not claim ownership of them.

13. **New assets**

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

    Answer: [Yes].

    Justification: The supplementary material includes the code and data artifacts, and the paper documents how the training data, checklist labels, prompts, and evaluation settings are constructed. The assets are described in Sections 4 and Appendix B.1–B.6.

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

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

    Answer: [N/A].

    Justification: The work does not involve crowdsourcing or new research with human subjects. Data filtering, checklist labeling, tool simulation, and judging are performed with rules and LLMs.

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

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

    Answer: [N/A].

    Justification: The work does not involve new human-subject experiments, crowdsourcing, or participant data collection. Therefore IRB approval or equivalent review is not applicable.

16. **Declaration of LLM usage**

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

    Answer: [Yes].

    Justification: LLM use is central to the method and is described throughout the paper: GPT-5 is used for filtering, CoT compression, and checklist labeling; an LLM simulator produces tool responses; and an LLM judge evaluates checklist items. The corresponding prompts and settings are provided in Appendix B.1–B.6.