TOOLRM: Towards Agentic Tool-Use Reward Modeling

Renhao Li[1,4*], **Jianhong Tu**[2], **Yang Su**[2], **Yantao Liu**[2], **Fei Huang**[2],
Hamid Alinejad-Rokny[3], **Derek F. Wong**[1✉], **Junyang Lin**[2✉], **Min Yang**[4✉]

[1]University of Macau, [2]Qwen Team, Alibaba Inc., [3]UNSW Sydney
[4]Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences
li.renhao@connect.um.edu.mo, derekfw@um.edu.mo
junyang.ljy@alibaba-inc.com, min.yang@siat.ac.cn

🤗 Dataset   Code

Abstract

Reward models (RMs) play a critical role in aligning large language models (LLMs) with human preferences. Yet in the domain of tool learning, the lack of RMs specifically designed for function-calling tasks has limited progress toward more capable agentic AI. We introduce TOOLRM, a family of lightweight reward models tailored for general tool-use scenarios. To build these models, we propose a novel pipeline that constructs high-quality pairwise preference data using rule-based scoring and multidimensional sampling. This yields *ToolPref-Pairwise-30K*, a diverse, balanced, and challenging preference dataset that supports both generative and discriminative reward modeling. We also introduce TRBENCH$_{\text{BFCL}}$, a benchmark built on the agent evaluation suite BFCL to evaluate RMs on tool calling tasks. Trained on our constructed data, models from the Qwen3-4B/8B series achieve up to 17.94% higher accuracy, substantially outperforming frontier LLMs and RMs in pairwise reward judgments. Beyond training objectives, generative ToolRM generalizes to broader critique tasks, including Best-of-N sampling and self-correction. Experiments on ACEBench highlight its effectiveness and efficiency, enabling inference-time scaling while reducing output token usage by over 66%. Its support for downstream RL training further validates its practical utility. We release data to facilitate future research.

1 Introduction

Recent advances in agentic artificial intelligence have been driven in large part by the tool-use capabilities (Patil et al., 2024; OpenAI, 2025) of large language models (LLMs). By leveraging external tools, LLMs can recognize their limitations and extend their capabilities through environment interaction. The research focus has recently shifted from behavior cloning via supervised finetuning on curated trajectories (Schick et al., 2023; Tang et al., 2023) to trial-and-error approaches based on reinforcement learning from verifiable rewards (RLVR, Feng et al., 2025; Qian et al., 2025), enabling generalizable and robust tool-use behavior.

Despite these gains, the lack of reliable reward models (RMs) tailored to tool-use tasks remains a core limitation. Most existing methods depend on verified tool-call trajectories for feedback, which restricts scalability to domains lacking such annotations. At inference time, the absence of precise reward signals also makes it hard to leverage multiple sampled answers for test-time selection (Wang et al., 2023; Snell et al., 2025). We argue that developing a robust RM—capable of evaluating tool-use behavior without requiring ground-truth labels—is critical for advancing this field. Building effective RMs for tool-use presents three key challenges: **(C1)** Constructing high-quality preference pairs that reflect tool-use intent. **(C2)** Enabling generalizable critique beyond 3H-style modeling (Askell et al., 2021), as tool-use tasks often allow more objective, causal reasoning. **(C3)** Evaluating RM performance in this setting, which remains underexplored for frontier LLMs and specialized critics.

To address these challenges, we introduce TOOLRM, a family of lightweight RMs for general tool-use tasks, along with a two-stage pipeline for constructing high-quality preference data to train them. First, we curate and validate tool-calling trajectories from diverse open-source datasets, segment them into context–response pairs, and sample alternative responses using multiple LLMs. Instead of relying on ground-truth matches, we apply rule-based labeling to capture fine-grained preferences. A multidimensional sampling strategy ensures diverse scenarios, varied preference intensity, and high task complexity **(C1)**. To strengthen critique ability, we train generative ToolRM with a pairwise objective using unified instructions and verifiable

*Work done during an internship at Qwen Team.
✉Corresponding author.

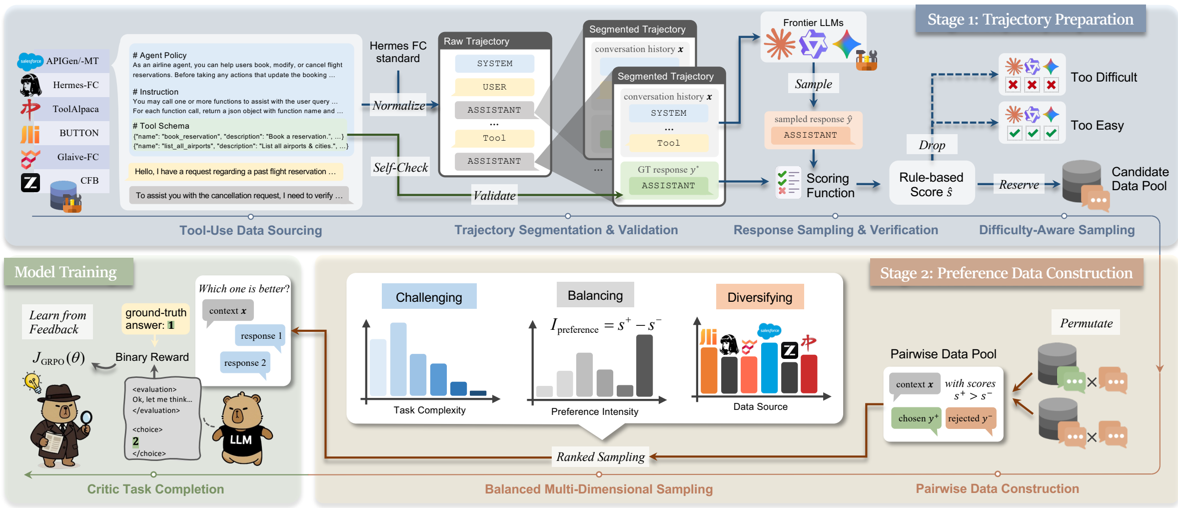


Figure 1: Overview of the proposed pipeline for training ToolRM.

supervision, enabling the model to learn robust reasoning without curated traces **(C2)**. We also introduce TRBench$_{\text{BFCL}}$, a benchmark based on BFCL (Patil et al., 2025), to systematically evaluate RM performance on tool-use tasks **(C3)**.

Our contributions are summarized as follows:

- We propose a novel pipeline for generating high-quality pairwise preference data for reward modeling in tool-use scenarios. Using seven open-source tool-calling datasets, we construct *ToolPref-Pairwise-30K*, a diverse and balanced set of 30,000 challenging preference pairs. This resource is publicly released to support future work in tool-oriented reward modeling.

- We train ToolRM on the Qwen3 families with different objectives, achieving substantial gains in pairwise reward judgments. The resulting models generalize to broader critique tasks, enabling efficient inference-time scaling, producing compact, high-quality critiques, and supporting effective downstream RL training.

- We introduce TRBench$_{\text{BFCL}}$, a dedicated benchmark for evaluating reward models in tool-use settings. Our analysis reveals that even state-of-the-art LLMs and specialized reward models show significant gaps on this benchmark, underscoring the need for targeted solutions.

2 Related Work

Tool Learning in the Era of LLMs. Early work on agentic AI, such as Yao et al. (2023), combines chain-of-thought reasoning (Wei et al., 2022) with tool-augmented actions to elicit LLMs' tool-use capabilities. Later methods imitate curated tool-use trajectories via supervised fine-tuning (Schick et al., 2023; Liu et al., 2024b), but often struggle with out-of-distribution tasks. More recent works integrate verified rewards into tool-aware reasoning, with designs tailored for question answering (Jin et al., 2025; Song et al., 2025), math (Feng et al., 2025; Dong et al., 2025), and general tool-use (Qian et al., 2025; Zhang et al., 2025). Agarwal et al. (2025) evaluate reward models on function-calling tasks but omits multi-turn scenarios and rely on data pairs too simple for powerful models to distinguish.

Reward Modeling. Reward models guide large language models toward human-preferred outputs (Ouyang et al., 2022; Bai et al., 2022). They are typically either (1) discriminative (DiscRM), outputting scalar scores to rank responses (Cai et al., 2024; Liu et al., 2024a), or (2) generative (GenRM), producing textual rewards for domains such as chat (Shiwen et al., 2024), code (McAleese et al., 2024), and literary translation (Pombal et al., 2025). A recent trend views reward modeling as a reasoning process (Chen et al., 2025c; Guo et al., 2025b) to enhance reward quality. Recent work (Agarwal et al., 2025) examines RM performance on function-calling tasks, but the benchmark is not sufficiently challenging for powerful models.

3 Methodology

We introduce a pipeline for training ToolRM. As shown in Figure 1, we first label tool-calling trajectories with rule-based verifiers, then construct pairwise preference data via balanced multidimensional sampling. ToolRMs are trained as either GenRM or DiscRM under distinct objectives.

3.1 Trajectory Preparation

Task Sourcing. To build a diverse dataset, we collate function-calling tasks from seven open-source, tool-learning datasets, spanning a wide variety of task domains and trajectory patterns: APIGen (Liu et al., 2024b), APIGen-MT (Prabhakar et al., 2025), BUTTON (Chen et al., 2025b), ComplexFuncBench (Zhong et al., 2025), Hermes-Function-Calling (Teknium et al., 2025), Glaive-Function-Calling[1], and ToolAlpaca (Tang et al., 2023). To address format inconsistencies across these sources, we standardize all conversation records of raw tasks into format-aligned trajectories $\mathcal{T}_{\text{raw}} = \{\tau_i\}_{i=1}^{N}$, discarding any data with invalid role orders. The message format within each trajectory τ_i is normalized to adhere to the Hermes Function Calling standard, where special tags `<tools>`, `<tool_call>`, and `<tool_response>` are used to enclose tool schemas, calls, and responses, respectively. At the beginning of each τ_i, a function-calling prompt is uniformly included as the system message, along with the schemas of available tools in the task. Additional agent policies are prepended to this message for complex tasks from specific sources (e.g., APIGen-MT). See Appendix I for an example of a tool-use task trajectory.

Trajectory Segmentation and Validation. To enable subsequent rule-based verification of arbitrary trajectories against ground-truth answers, we first perform tool schema validation for each trajectory τ_i. Tool schemas are typically provided as dictionary objects, which we verify as valid JSON schemas describing tools compatible with OpenAI's tool-calling format. Invalid schemas are corrected, and duplicates are removed. The validated schemas are then wrapped into function-type JSON objects and incorporated into the aforementioned system message as tool descriptions.

Next, we partition each raw trajectory $\tau_i \in \mathcal{T}_{\text{raw}}$ into sub-trajectories ending in an assistant message, yielding segmented trajectories $\mathcal{T}_{\text{seg}} = \{\tau_j\}_{j=1}^{M}$. Each segment τ_j consists of a conversation history $\mathbf{x}_j$ (all messages preceding the assistant message) and the corresponding assistant response y_j. A preliminary filtering is then applied: we retain τ_j only if the message following y_j in the raw trajectory τ_i does not contain any unsuccessful tool response, ensuring the basic validity of tool calls in y_j.

A stricter tool-call validation is then applied to y_j for each retained trajectory τ_j. Each tool call must be parsable in the required format (e.g., `{"name":". . . ","arguments":{. . . }}`), and its function name and arguments must match the tool schema. Responses containing duplicate tool calls are also discarded. Only the trajectories $\tau_j = (\mathbf{x}_j, y_j)$ that pass all format and content checks are kept. For these validated trajectories, we treat the assistant response as the ground-truth response y_j^*, and the clean dataset is $\mathcal{T}_{\text{clean}} = \{(\mathbf{x}_j, y_j^*)\}_{j=1}^{M'}$. Data statistics are reported in Appendix B.

Response Sampling and Verification. In this phase, we begin by sampling multiple model responses for each conversation history. To ensure diversity in the outputs, we select five models from three different families with varying tool-calling capabilities: Claude-3.7-Sonnet, Gemini-2.5-Pro, Qwen2.5-Max, Qwen-32B, and Qwen3-8B. For each pair $(\mathbf{x}_j, y_j^*)$ in the cleaned dataset $\mathcal{T}_{\text{clean}}$, the context $\mathbf{x}_j$ is sent to all five models, yielding a set of new assistant responses $\{\hat{y}_{j,k}\}_{k=1}^{5}$. Each sampled response $\hat{y}_{j,k}$ is then scored using a rule-based function that compares it against its corresponding ground-truth response y_j^*, yielding a score between 0 and 1. Unlike prior rule-based TIR approaches (Qian et al., 2025), our method for training the reward model prioritizes the correctness of tool call content (reasoning ability) over strict format adherence (instruction-following ability), since downstream applications often use varying tool call structures. Consequently, we only score $\hat{y}$ that can be successfully parsed into the expected tool-call format and discard all others.

For a given ground-truth response y^* and a sampled response $\hat{y}$ (we drop indices j, k for simplicity), let $\mathcal{C}^* = \{c_i^*\}_{i=1}^{N_{\text{G}}}$ and $\hat{\mathcal{C}} = \{\hat{c}_l\}_{l=1}^{N_{\text{P}}}$ denote the lists of tool calls parsed from them, respectively. Each tool call is a JSON object containing a string-typed `name` and a dictionary of `arguments`. Scoring starts with two disqualifiers: if either applies, the final score $\hat{s}$ is set to 0:

- *Mismatched Number of Tool Calls.* The number of predicted tool calls does not match the number of ground-truth tool calls:

$$|\hat{\mathcal{C}}| \neq |\mathcal{C}^*| \Rightarrow \hat{s} = 0 \tag{1}$$

- *Duplicated Tool Calls.* The set of predicted tool calls contains identical duplicates (both name and

[1] We use a 5k cleaned glaive-function-calling subset in hermes-function-calling-v1.

arguments are the same). For $\hat{c}_l, \hat{c}_m \in \hat{\mathcal{C}}$:

$$\exists l \neq m \text{ s.t. } \texttt{is\_identical}(\hat{c}_l, \hat{c}_m) \Rightarrow \hat{s} = 0 \quad (2)$$

If a sampled response $\hat{y}$ passes the above initial checks, a match score s_i is calculated for each ground-truth tool call $c_i^* \in \mathcal{C}^*$. This score is determined by matching c_i^* with the predicted tool call of the same name that achieves the highest argument similarity. Specifically:

$$\begin{aligned} s_i = \max_{\hat{c} \in \hat{\mathcal{C}}} \mathbb{1}\left[c_i^*.\texttt{name} = \hat{c}.\texttt{name}\right] \\ \cdot\, \texttt{sim}\big(c_i^*.\texttt{arguments}, \hat{c}.\texttt{arguments}\big) \end{aligned} \quad (3)$$

where $\mathbb{1}[\cdot]$ is an indicator function equal to 1 if the tool names match and 0 otherwise, so arguments are only compared when tool names align. The argument similarity function $\texttt{sim}(\cdot)$ measures the ratio of identical key-value pairs to the total number of unique keys across both dictionaries. A key-value pair is considered identical only if the key appears in both dictionaries and the corresponding values match, with string comparisons performed in a case-insensitive manner. If both dictionaries are empty, the similarity is defined as 1. The final rule-based score $\hat{s}$ can then be calculated as the mean of all individual match scores s_i, with $\hat{s} = 1$ when both y^* and $\hat{y}$ contain no tool calls:

$$\hat{s} = \frac{1}{N_{\text{G}}} \sum_{i=1}^{N_{\text{G}}} s_i \quad (4)$$

Difficulty-Aware Down-Sampling. After collecting all rule-based scores for sampled responses, we perform difficulty-aware down-sampling. This is done by grouping all sampled responses by their original context $\mathbf{x}_j$. Empirically, tasks that are either too easy or too difficult are not ideal for model training: (1) contexts for which all sampled responses have a rule-based score of 1 are discarded, as they offer no meaningful variation for model critique; (2) contexts for which no sampled response receives a rule-based score of 1 are also removed, as such cases likely contain noise in either $\mathbf{x}_j$ or y_j^*. We retain the remaining candidates as:

$$\mathcal{D}_{\text{cand}} = \{(\mathbf{x}_j, y_j^*, \hat{y}_{j,k}, \hat{s}_{j,k}) \mid \text{context } j \text{ passes}\} \quad (5)$$

Each contains the conversation history, the ground-truth response, a sampled response, and the corresponding rule-based score. This pool serves as the source for constructing preference datasets.

3.2 Preference Data Construction

Pairwise Data Construction. This section describes how we construct data to train the RM as a critic. Such models typically evaluate data in either a *pointwise* or *pairwise* manner. In preliminary experiments, pointwise generative RMs supervised by rule-based scores exhibited superficial overfitting: it mimicked the training score distribution instead of developing genuine analytical ability, a form of reward hacking that hurt performance on out-of-distribution (OOD) tasks. To mitigate this, we train RMs on pairwise critique tasks, which rely on comparative judgments rather than direct scoring. The pairwise reward model is designed to distinguish a preferred response from a rejected one for a given context. To construct the training data for this, we sample pairs of responses from the preprocessed data pool $\mathcal{D}_{\text{cand}}$, where ground-truth preferences are determined by their rule-based scores. Each pair consists of a chosen response y^+ and a rejected response y^- that shares the same context but differs in score. We traverse $\mathcal{D}_{\text{cand}}$ and arrange the permutations according to the above rules to get a candidate pairwise data pool:

$$\begin{aligned} \mathcal{D}_{\text{pair-cand}} = \{(\mathbf{x}, y^*, y^+, y^-, s^+, s^-) \mid s^+ > s^-, \\ (\mathbf{x}, y^*, y^+, s^+), (\mathbf{x}, y^*, y^-, s^-) \in \mathcal{D}_{\text{cand}}\} \end{aligned} \quad (6)$$

Balanced Multi-Dimensional Sampling. To enable efficient training with fewer data, we then adopt a balanced, multi-dimensional sampling strategy to select samples from $\mathcal{D}_{\text{pair-cand}}$ where we focus on the following three dimensions of data:

- *Diversity of Data Sources.* Incorporating a diverse range of tool schemas and user queries enhances the generalizability of trained models. To this end, we aim to sample contexts from different sources in a balanced manner. For each context $\mathbf{x}$ in data, we denote its source as $\mathbf{x}.\texttt{source}$.
- *Coverage of Preference Intensity.* For each pair of chosen and rejected responses, the difference in their rule-based scores reflects the intensity of the preference signal: a large difference signifies a strong preference, while a small difference suggests a weak one. To train robust reward models, our data sampling process is designed to cover this full spectrum of preference signals, from weak to strong. For each pairwise data point, we measure its preference intensity by:

$$I_{\text{preference}} = s^+ - s^- \quad (7)$$

- *Complexity of Tasks.* Challenging the reward model with more complex tasks is essential for enhancing its analytical capabilities. We calculate the complexity score of one candidate data point according to its ground-truth response y^*:

$$S_{\text{complex}} = |\mathcal{C}^*| + \sum_{i=1}^{N_G} |c_i^*.\texttt{arguments}| \quad (8)$$

where $\mathcal{C}^*$ is the set of tool calls parsed from y^*. Both the number of tool calls and arguments are accumulated to measure the task complexity. Over-complicated samples ($S_{\text{complex}} > 50$) are filtered out for a higher success rate of rollout trajectory in the model training stage.

Guided by the above principles, we use a heuristic algorithm to select samples from $\mathcal{D}_{\text{pair-cand}}$ that are more efficient for model training. Specifically, we prioritize samples with higher complexity scores S_{complex} while ensuring that the data source `x.source` and preference intensity $I_{\text{preference}}$ are as balanced as possible, resulting in a subset of pairwise data $\mathcal{D}_{\text{pair-sampled}} \subseteq \mathcal{D}_{\text{pair-cand}}$ for subsequent model training. Details of the heuristic algorithm are provided in Appendix H.

3.3 Model Training

Critique Task Design. We train generative ToolRM by prompting models with critique tasks. Given a conversation history and two candidate responses, the model must thoroughly evaluate each one and select the better response, outputting its name within `<choice>` tags. We adapt instructions to each model's native style: reasoning models use a *think-mode* template, embedding evaluations in their reasoning process, while non-reasoning models use a *no-think-mode* template, placing evaluations explicitly in `<evaluation>` tags. We define unified evaluation criteria to ensure consistent, comprehensive critiques and to specify which tool-invocation errors should be penalized. For each sampled data $(\mathbf{x}, y^*, y^+, y^-, s^+, s^-) \in \mathcal{D}_{\text{pair-sampled}}$, we format the conversation history $\mathbf{x}$ into a single string. This string is then concatenated with the two assistant responses y^+ and y^- to form the final input query q. To reduce position bias and prevent reward hacking during training, we randomly swap the order of the assistant responses in 50% of the queries, recording the position of y^+ as the ground-truth answer a. The resulting dataset $\mathcal{D}_{\text{pref}} = \{(q, y^+, y^-, a)_i\}_{i=1}^K$ is then used to train the reward model. See Appendix J for detailed prompt templates.

Training Objectives. We train generative ToolRM (ToolRM-Gen) in a RLVR paradigm using Group Relative Policy Optimization (GRPO, Shao et al., 2024), a variant of Proximal Policy Optimization (PPO, Schulman et al., 2017) that improves efficiency and reduces computational cost by replacing the critic network with grouped relative advantages. Given an input query q and its ground-truth answer a, let $\mathcal{O} = \{o_1, o_2, \ldots, o_G\}$ denote the set of rollout trajectories generated by the old policy $\pi_{\theta_{\text{old}}}$. Our goal is to optimize the policy π_θ by maximizing the following objective:

$$\begin{aligned} J_{\text{GRPO}}(\theta) &= \mathbb{E}_{(q,a)\sim\mathcal{D}_{\text{pref}}, \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)} \\ &\Bigg[\frac{1}{G}\sum_{i=1}^G \frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Bigg[\min\Bigg(\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q, o_{i,<t})}A_{i,t}, \\ &\text{clip}\left(\frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q, o_{i,<t})}, 1-\epsilon, 1+\epsilon\right) A_{i,t}\Bigg)\Bigg]\Bigg] \end{aligned} \quad (9)$$

where ϵ is a clipping-related hyper-parameter for stabilizing training. $A_{i,t}$ denotes the relative advantage calculated on outputs of each rollout group:

$$A_{i,t} = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_G\})}{\text{std}(\{r_1, r_2, \ldots, r_G\})} \quad (10)$$

Here, r_i denotes the binary reward assigned to the rollout trajectory o_i. It is determined by whether a valid choice can be successfully extracted from o_i and whether it accurately answers q:

$$r_i = \begin{cases} 1, \text{if } \texttt{is\_equivalent}(a, \texttt{extract\_choice}(o_i)) \\ 0, \text{otherwise.} \end{cases}$$

Following Qian et al. (2025), we omit the KL penalty term from the original GRPO objective to encourage more effective exploitation of reward signals during policy updates. Building on this, we design a verifiable reward system for training generative reward models in the tool-use scenario.

We train discriminative ToolRM (ToolRM-Disc) with a pairwise ranking loss based on the Bradley-Terry model (Bradley and Terry, 1952), following Ouyang et al. (2022):

$$\mathcal{L}_{\text{ranking}} = -\log(\sigma(r_\theta(\mathbf{x}, y^+) - r_\theta(\mathbf{x}, y^-))) \quad (11)$$

where $r_\theta(\mathbf{x}, y^+)$ and $r_\theta(\mathbf{x}, y^-)$ are the scalar rewards assigned by the reward model θ to the chosen response y^+ and the rejected response y^- for a given prompt $\mathbf{x}$.

Table 1: Evaluation results of reward models on TRBench$_{BFCL}$. Higher accuracy indicates a stronger ability to distinguish better responses. GenRMs and DiscRMs trained on *ToolPref-Pairwise-30K* are highlighted in green. The best result in each group is **bolded**, and the second-best is underlined. (◇): evaluated with the *think-mode* template; (♡): evaluated with the *no-think-mode* template; (♣): evaluated with the official template. (⚖️): pairwise inputs; (🎯): pointwise inputs; (💬): critique as output; (✅): choice as output; (🔢): scalar reward as output.

Models	Classification Accuracy (%) S	M	P	PM	LS	LM	LP	LPM	MTB	Avg.	W-Avg.
Proprietary & Open-source General LLMs											
⚖️💬 Meta/Llama-3.2-3B-Instruct♡	34.31	33.80	24.58	34.87	26.89	29.54	8.82	30.00	20.20	27.00	28.09
⚖️💬 Meta/Llama-3.1-8B-Instruct♡	45.99	52.11	46.84	62.31	33.02	39.44	23.53	40.00	28.69	41.33	41.38
⚖️💬 Qwen/Qwen3-4B-Thinking-2507◇	67.88	70.42	85.71	87.69	61.79	46.61	**85.29**	85.00	33.54	69.33	57.59
⚖️💬 Qwen/Qwen3-4B (w/ thinking)◇	70.07	73.24	89.70	87.69	56.60	48.09	79.41	81.67	39.80	69.59	59.34
⚖️💬 Qwen/Qwen3-8B (w/ thinking)◇	71.53	61.97	89.37	90.26	58.49	48.09	**85.29**	76.67	39.19	68.98	59.44
⚖️💬 Qwen/Qwen3-4B-Instruct-2507♡	71.53	64.79	90.37	89.23	51.42	50.66	70.59	**86.67**	36.57	67.98	59.67
⚖️💬 DeepSeek-AI/DeepSeek-R1-0528◇	68.61	70.42	87.71	85.64	64.62	46.45	76.47	75.00	36.77	67.97	57.93
⚖️💬 OpenAI/GPT-4o-2024-11-20♡	69.34	66.20	86.71	86.67	50.47	50.82	67.65	78.33	38.38	66.06	59.00
⚖️💬 OpenAI/o3-2025-04-16◇	70.80	69.01	85.71	84.87	55.19	50.43	67.65	76.67	**41.21**	66.84	59.40
⚖️💬 Google/Gemini-2.5-Flash (w/ thinking)◇	64.23	66.20	89.70	89.49	56.13	51.13	79.41	80.00	36.77	68.12	59.87
⚖️💬 Google/Gemini-2.5-Pro (w/ thinking)◇	75.18	67.61	88.04	91.79	58.96	48.32	82.35	73.33	39.80	69.49	59.94
⚖️💬 Qwen/Qwen3-235B-A22B-Thinking-2507◇	71.53	69.01	86.05	90.26	**67.92**	51.52	**85.29**	76.67	34.55	70.31	60.64
⚖️💬 DeepSeek-AI/DeepSeek-V3-0324♡	75.18	66.20	88.70	89.74	58.02	53.86	70.59	73.33	37.17	68.09	61.45
⚖️💬 Qwen/Qwen2.5-Max♡	77.37	73.24	89.04	90.00	58.02	**55.18**	67.65	70.00	37.98	68.72	62.39
⚖️💬 Anthropic/Claude-3.7-Sonnet (w/ thinking)◇	76.64	67.61	**91.69**	**92.82**	60.85	52.77	73.53	78.33	39.19	70.38	62.45
⚖️💬 **Anthropic/Claude-4-Sonnet (w/ thinking)**◇	**81.02**	**77.46**	91.36	91.28	62.74	54.95	82.35	83.33	41.01	**73.95**	**64.23**
Generative Reward Models											
🎯💬🔢 Databricks/CLoud-RM-Llama-3-8B♣	25.55	35.21	33.22	32.82	31.60	37.88	32.35	25.00	49.90	33.73	37.34
⚖️💬 Unbabel/M-Prometheus-7B♡	54.74	54.93	71.43	74.87	43.87	46.69	38.24	53.33	34.14	52.47	51.19
⚖️💬 Microsoft-Research/RRM-7B◇	65.69	56.34	82.06	84.62	43.40	49.65	44.12	68.33	36.36	58.95	56.05
⚖️💬 UIUC/RM-R1-DeepSeek-Distilled-Qwen-32B◇	75.18	76.06	68.44	80.51	61.79	49.18	52.94	53.33	38.18	61.73	56.25
⚖️💬 Unbabel/M-Prometheus-14B♡	64.96	57.75	88.37	87.44	44.34	46.38	64.71	61.67	39.39	61.67	56.32
⚖️✅ Skywork/Skywork-Critic-Llama-3.1-8B♣	54.74	59.15	86.05	83.59	47.17	45.75	67.65	61.67	50.30	61.79	56.92
⚖️💬 ToolRM-Gen-Llama-3.2-3B-Instruct♡	54.01	57.75	87.04	78.97	44.34	54.95	64.71	61.67	45.45	60.99	59.27 (+31.18)
⚖️💬 ToolRM-Gen-Llama-3.1-8B-Instruct♡	62.04	61.97	88.70	86.15	47.64	52.30	82.35	68.33	41.21	65.63	59.57 (+18.19)
⚖️✅ Skywork/Skywork-Critic-Llama-3.1-70B♣	64.23	67.61	87.38	88.21	44.34	51.68	70.59	66.67	47.47	65.35	60.31
⚖️💬 Microsoft-Research/RRM-32B◇	76.64	76.06	87.38	89.23	67.92	56.90	67.65	75.00	42.83	71.07	64.50
⚖️💬 ToolRM-Gen-Qwen3-4B-Instruct-2507♡	70.80	74.65	**91.03**	89.49	55.66	60.41	**94.12**	81.67	49.90	74.19	66.85 (+7.18)
⚖️💬 ToolRM-Gen-Qwen3-4B◇	81.02	78.87	89.04	88.97	63.21	62.12	91.18	86.67	52.32	77.04	68.89 (+9.55)
⚖️💬 ToolRM-Gen-Qwen3-8B◇	81.02	76.06	89.70	91.03	64.62	61.50	91.18	80.00	**52.73**	76.43	68.92 (+9.48)
⚖️💬 **ToolRM-Gen-Qwen3-4B-Thinking-2507**◇	**83.21**	**80.28**	90.03	**92.56**	**71.23**	**66.02**	**94.12**	**88.33**	52.12	**79.77**	**71.87** (+14.28)
Discriminative Reward Models											
🎯🔢 Skywork/Skywork-Reward-Llama-3.1-8B-v0.2♣	83.21	70.42	92.36	**92.31**	59.91	62.51	67.65	75.00	59.80	73.68	70.23
🎯🔢 InternLM/InternLM2-7B-Reward♣	80.29	80.28	88.04	89.74	63.68	65.16	67.65	73.33	61.21	74.38	71.17
🎯🔢 Skywork/Skywork-Reward-V2-Llama-3.1-8B♣	**88.32**	77.46	90.70	91.03	68.40	64.54	70.59	**90.00**	60.81	77.98	72.28
🎯🔢 InternLM/InternLM2-20B-Reward♣	87.59	**84.51**	91.69	91.54	64.15	68.67	88.24	76.67	55.35	78.71	73.08
🎯🔢 Skywork/Skywork-Reward-V2-Qwen3-4B♣	85.40	80.28	**94.35**	92.05	70.28	67.58	76.47	83.33	55.76	78.39	73.25
🎯🔢 ToolRM-Disc-Qwen3-4B-Thinking-2507	81.75	**84.51**	93.69	90.26	**72.17**	**74.51**	97.06	71.67	60.61	80.69	76.80 (+19.21)
🎯🔢 **ToolRM-Disc-Qwen3-4B-Instruct-2507**	84.67	**84.51**	94.02	90.00	**72.17**	73.97	**100.00**	80.00	**64.85**	**82.69**	**77.61** (+17.94)

4 Experiments

4.1 Do ToolRM Provide Precise Rewards?

Benchmark Construction. We evaluate reward models on an improved benchmark adapted from Agarwal et al. (2025), based on BFCL. The original benchmark pairs correct function calls with incorrect ones generated by 25 permissively licensed models but has two main limitations: (1) it only covers single-turn tasks, and (2) its data pairs are too trivial for powerful RMs to differentiate. To address this, we build a more challenging benchmark using the *multi_turn_base* split from BFCL-v3 and curate harder rejected responses from seven top-performing function-calling models.

The resulting benchmark, TRBench$_{BFCL}$, comprises 2,983 samples from 1,397 unique tasks across 9 splits: simple (S), multiple (M), parallel (P), parallel multiple (PM), live sample (LS), live multiple (LM), live parallel (LP), live parallel multiple (LPM), and multi-turn base (MTB). It covers 20 distinct error types with rejected responses from 7 different models. Since BFCL tasks and their synthetic data are excluded from training, **TRBench$_{BFCL}$ serves as a strong OOD evaluation set for ToolRM**. Additional statistics and implementation details are in Appendix D.2.

Evaluation Metric. We assess model performance via pairwise preference classification. To minimize position bias, each sample is evaluated twice, swapping the response order on the second pass. A sample is correct only if both orders yield the correct prediction. For scalar-output RMs, we compute scores for chosen and rejected responses and mark the result correct if the score order matches the preference label. We report average accuracy (**Avg.**) across splits and weighted-average accuracy (**W-Avg.**), based on sample counts.

Model Training. We train ToolRM on three reasoning models (Qwen3-4B, Qwen3-8B, and Qwen3-4B-Thinking-2507) and three non-reasoning models (Qwen3-4B-Instruct-2507, Llama-3.2-3B-Instruct, and Llama-3.1-8B-Instruct (Dubey et al., 2024)). At both training and inference for generative ToolRM, we apply the appropriate *think-mode* or *no-think-mode* templates. Our preference dataset, *ToolPref-Pairwise-30K*, contains 30,000 samples (29,500 for training, 500 for validation), built with our proposed pipeline. See training details in Appendix D.1 and impact of data scaling on ToolRM in Appendix E.

Baseline Models. We benchmark ToolRM against strong LLMs in the LLM-as-a-judge setup, including GPT, Gemini, Claude, DeepSeek, and Qwen. Specialized reward models are also tested: generative (Skywork-Critic (Shiwen et al., 2024), M-Prometheus (Pombal et al., 2025), RM-R1 (Chen et al., 2025c), RRM (Guo et al., 2025b)), discriminative (Skywork-Reward (Liu et al., 2024a, 2025a), InternLM2-Reward (Cai et al., 2024)), and hybrid (Cloud-RM (Ankner et al., 2024)).

Main Results. Table 1 presents evaluation results on TRBENCH$_{\text{BFCL}}$ across all splits. Training on *ToolPref-Pairwise-30K* significantly boosts performance, with Qwen3 models gaining 12.94% on average and up to 17.94% in weighted accuracy. ToolRM, trained on Qwen3-4B-Thinking-2507 and Qwen3-4B-Instruct-2507, achieves the best results within each model group. Notably, ToolRM also improves on the *multi-turn-base* split, despite being trained on step-wise critiques. Since BFCL scoring for multi-turn tasks relies on state- and response-based signals rather than rule-matching, these gains demonstrate that **ToolRM acquires robust, generalizable analytical capabilities rather than overfitting to rule-based labels.**

In LLM-as-a-judge evaluations, Claude-4 outperforms other general-purpose LLMs, consistent with its stronger tool-use capabilities. Among specialized reward models, the Skywork-Reward-V2 series performs best, likely due to training on 26M diverse preference pairs. Notably, Skywork-Reward-Llama-3.1-8B-v0.2 exceeds its generative counterpart, Skywork-Critic, despite similar training

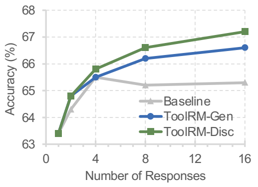


Figure 2: Model BoN sampling on ACEBench.

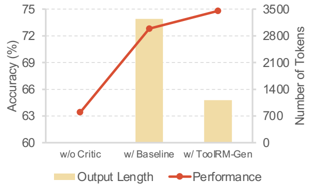


Figure 3: Model self-correction on ACEBench.

data—a pattern also observed in ToolRM. This suggests that discriminative reward models may generalize better to pairwise classification tasks than generative critics (see Appendix G for discussion).

Lastly, reasoning models show greater gains from pairwise RL training than instruction-tuned counterparts, and models with longer initial reasoning patterns (e.g., Qwen3-4B-Thinking-2507 vs. Qwen3-4B) benefit the most. This highlights that **even with weaker initial performance, a greater capacity for exploration can ultimately lead to stronger outcomes through RL.** See the ablation on training data construction in Appendix F.

4.2 Do ToolRM Aid Inference-Time Scaling?

Setup. We assess whether ToolRM improves tool-call inference using 823 samples from the *Normal* split of ACEBENCH (Chen et al., 2025a), a benchmark for tool-use evaluation. For each sample, we apply Best-of-N (BoN) sampling with Qwen3-4B-Instruct-2507 (temperature = 1.0), and use trained RMs to select the best response. We compare three judges: the baseline Qwen3-4B-Thinking-2507 (*Base*), the trained ToolRM-Gen-Qwen3-4B-Thinking-2507 (*ToolRM-Gen*), and ToolRM-Disc-Qwen3-4B-Instruct-2507 (*ToolRM-Disc*). Performance is measured by average accuracy across all samples.

Main Results. Figure 2 shows that ToolRM (-Disc/-Gen) consistently matches or outperforms the baseline, improving by 3.8/3.2 points without BoN and 1.9/1.3 points with BoN-16. These gains validate the quality of *ToolPref* and indicate that ToolRM-Gen generalizes beyond its RL training objective. Notably, performance remains stable as the candidate pool grows, demonstrating **robustness to long-context reasoning** and **effectiveness for inference-time scaling in tool-use tasks**.

4.3 Are Model Critiques Helpful?

Setup. We evaluate how critiques generated by ToolRM guide policy model self-correction

Table 2: Policy model accuracy (%) on BFCL-v3 before and after RL training with ToolRM-Gen.

Model	Single-Turn AST	Multi-Turn
Qwen3-4B-Instruct-2507	73.25	19.88
- RL w/ ToolRM-Gen	**77.89** (+4.64)	**25.50** (+5.62)

(SC). For each sample in the *Normal* subset of ACEBench, Qwen3-4B-Instruct-2507 first produces a function-calling response. A GenRM then critiques the output with concise feedback, which the policy model uses to refine its response. We compare two critics here: the baseline Qwen3-4B-Thinking-2507 (*Baseline*) and the trained ToolRM-Gen-Qwen3-4B-Thinking-2507 (*ToolRM-Gen*). Performance is measured by average accuracy over all samples.

Main Results. As shown in Figure 3, ToolRM-Gen leads to notable gains in self-correction accuracy: +11.4 points over *w/o Critic* and +2.0 over *w/ Baseline*, indicating more reliable critiques in tool use tasks. It also lowers decoding cost, reducing average output length from 3,211 to 1,111 tokens, demonstrating **efficient reasoning without sacrificing critique quality.** See Appendix K for qualitative examples and Appendix G for a detailed comparison of ToolRM-Gen and ToolRM-Disc.

4.4 Do ToolRM Facilitate Policy RL Training?

Setup. We test whether ToolRM can improve another policy model's tool use by serving as a reliable reward model during RL training. Using 15,000 unlabeled tool-call queries, we train Qwen3-4B-Instruction-2507 with GRPO. During training, the policy model's rollouts are paired and scored by ToolRM-Gen, which assigns a reward of 1 to the chosen response and 0 to the other. We then evaluate the resulting policy model on BFCL-v3.

Main Results. Table 2 reports policy-model accuracy on BFCL-v3 across splits and shows substantial gains after supervised RL with ToolRM-Gen. The fact that applying the reward model to downstream RL enhances tool-calling agents further **underscores the practical utility of ToolRM**. See Appendix D.3 for detailed settings and results.

4.5 Error Analysis of ToolRM

We further analyze the thinking process of generative ToolRM in cases where its final judgments are inconsistent with the ground-truth preferences. Our investigation indicates that these errors primarily fall into two categories: (i) when the description of tool schema or parameters lacks concrete examples, the model is unable to infer the most appropriate tool invocation from the candidates, given the available tool information and the user's query; (ii) the originally annotated chosen response contains minor errors, while the rejected response has more fundamental and severe errors. The model correctly identifies all errors but fails to distinguish primary errors from secondary ones, leading to an incorrect pairwise reward. We believe the first type of error is constrained by the base model's inherent reasoning capability and is therefore more difficult to improve. The second type, however, is more tractable and can be mitigated through targeted optimization using higher-quality, non-perfect preference pairs, of which the chosen response still contains minor errors. Examples from $\text{TRBench}_{\text{BFCL}}$ corresponding to the two typical error types are provided in Appendix K for detailed reference.

5 Conclusion

This paper introduces a framework of data, models and benchmarks for agentic tool-use reward modeling. At its core is a novel data construction pipeline that combines rule-based labeling with balanced multi-dimensional sampling to automatically generate fine-grained pairwise preference data. The resulting dataset is diverse, well-balanced, and intentionally challenging, enabling efficient RL training and promoting nuanced reasoning beyond superficial signal matching.

Extensive evaluations across multiple benchmarks demonstrate the value of ToolRM in: (i) delivering high-fidelity reward signals that align with human preferences and outperform frontier baselines; (ii) enabling inference-time scaling by reliably selecting optimal outputs from diverse candidate pools; (iii) producing efficient, pointwise critiques that improve self-correction with minimal decoding overhead; and (iv) effectively supporting downstream RL training.

Together, these results suggest that reward models trained on high-quality data with suitable objectives can serve as effective judges and critics to support downstream decision-making in general tool-use settings. Future work could extend this approach to more open-ended agentic tasks, including RM-guided multi-agent coordination and planning. We hope this work provides new perspectives on efficient tool learning for LLMs.

Limitations

Although the trained ToolRM generalizes across tasks, it still has limitations. In agentic tool-use scenarios, whether ToolRM should be viewed as an outcome reward model (ORM) or process reward model (PRM) depends on the application context: (i) Single-action view: ToolRM scores each action's final output without access to intermediate chain-of-thought, so it effectively serves as an ORM. (ii) Multi-turn view: In longer trajectories, ToolRM primarily evaluates each step independently and is not intended to judge the overall task outcome or provide holistic trajectory-level feedback, aligning it more with a PRM. Extending ToolRM to evaluate full trajectories is important but technically challenging; because it would require different objectives and substantial additional work, we leave it to future research.

References

Mayank Agarwal, Ibrahim Abdelaziz, Kinjal Basu, Merve Unuvar, Luis A Lastras, Yara Rizk, and Pavan Kapanipathi. 2025. ToolRM: Outcome reward models for tool-calling large language models. *arXiv preprint arXiv:2509.11963*.

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D Chang, and Prithviraj Ammanabrolu. 2024. Critique-out-loud reward models. *arXiv preprint arXiv:2408.11791*.

Anthropic. 2025. Claude 3.7 sonnet and claude code.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, and 1 others. 2021. A general language assistant as a laboratory for alignment. *arXiv preprint arXiv:2112.00861*.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, and 1 others. 2022. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

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

Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, and 1 others. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, and 1 others. 2025a. Acebench: Who wins the match point in tool usage? *arXiv preprint arXiv:2501.12851*.

Mingyang Chen, sunhaoze, Tianpeng Li, Fan Yang, Hao Liang, KeerLu, Bin CUI, Wentao Zhang, Zenan Zhou, and weipeng chen. 2025b. Facilitating multi-turn function calling for LLMs via compositional instruction tuning. In *The Thirteenth International Conference on Learning Representations*.

Xiusi Chen, Gaotang Li, Ziqi Wang, Bowen Jin, Cheng Qian, Yu Wang, Hongru Wang, Yu Zhang, Denghui Zhang, Tong Zhang, and 1 others. 2025c. RM-R1: Reward modeling as reasoning. *arXiv preprint arXiv:2505.02387*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, and 1 others. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in LLMs. *arXiv preprint arXiv:2504.11536*.

Arduin Findeis, Floris Weers, Guoli Yin, Ke Ye, Ruoming Pang, and Tom Gunter. 2025. Can external validation tools improve annotation quality for LLM-as-a-judge? In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15997–16020, Vienna, Austria. Association for Computational Linguistics.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025a. Deepseek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Jiaxin Guo, Zewen Chi, Li Dong, Qingxiu Dong, Xun Wu, Shaohan Huang, and Furu Wei. 2025b. Reward reasoning model. *arXiv preprint arXiv:2505.14674*.

Jian Hu, Xibin Wu, Wei Shen, Jason Klein Liu, Weixun Wang, Songlin Jiang, Haoran Wang, Hao Chen, Bin Chen, Wenkai Fang, Xianyu, Yu Cao, Haotian Xu, and Yiming Liu. 2025. OpenRLHF: A ray-based easy-to-use, scalable and high-performance RLHF framework. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 656–666, Suzhou, China. Association for Computational Linguistics.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. GPT-4o system card. *arXiv preprint arXiv:2410.21276*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. OpenAI o1 system card. *arXiv preprint arXiv:2412.16720*.

Yuxuan Jiang and Francis Ferraro. 2026. Scribe: Structured mid-level supervision for tool-using language models. *arXiv preprint arXiv:2601.03555*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan O Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training LLMs to reason and leverage search engines with reinforcement learning. In *Second Conference on Language Modeling*.

Seungone Kim, Jamin Shin, Yejin Cho, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2025. Tülu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Ren Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. 2024. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. In *Forty-first International Conference on Machine Learning*.

Lei Li, Yekun Chai, Shuohuan Wang, Yu Sun, Hao Tian, Ningyu Zhang, and Hua Wu. 2024. Tool-augmented reward modeling. In *The Twelfth International Conference on Learning Representations*.

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. 2024a. Skywork-reward: Bag of tricks for reward modeling in LLMs. *arXiv preprint arXiv:2410.18451*.

Chris Yuhao Liu, Liang Zeng, Yuzhen Xiao, Jujie He, Jiacai Liu, Chaojie Wang, Rui Yan, Wei Shen, Fuxiang Zhang, Jiacheng Xu, and 1 others. 2025a. Skywork-reward-v2: Scaling preference data curation via human-ai synergy. *arXiv preprint arXiv:2507.01352*.

Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Wang Xinzhi, Yong Liu, Yasheng Wang, and 8 others. 2025b. ToolACE: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations*.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. 2025c. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*.

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, and 1 others. 2024b. APIGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. 2024. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, and 1 others. 2021. WebGPT: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

OpenAI. 2025. Deep research system card.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

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*.

Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2024. Gorilla: Large language model connected with massive APIs. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

José Pombal, Dongkeun Yoon, Patrick Fernandes, Ian Wu, Seungone Kim, Ricardo Rei, Graham Neubig, and André FT Martins. 2025. M-prometheus: A suite of open multilingual llm judges. *arXiv preprint arXiv:2504.04953*.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, and 1 others. 2025. APIGen-MT: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

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

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

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.

Tu Shiwen, Zhao Liang, Chris Yuhao Liu, Liang Zeng, and Yang Liu. 2024. Skywork critic model series.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2025. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. 2025. R1-searcher: Incentivizing the search capability in LLMs via reinforcement learning. *arXiv preprint arXiv:2503.05592*.

Qiaoyu Tang, Ziliang Deng, Hongyu Lin, Xianpei Han, Qiao Liang, Boxi Cao, and Le Sun. 2023. ToolAlpaca: Generalized tool learning for language models with 3000 simulated cases. *arXiv preprint arXiv:2306.05301*.

Ryan Teknium, Roger Jin, Jai Suphavadeeprasit, Dakota Mahan, Jeffrey Quesnelle, Joe Li, Chen Guang, Shannon Sands, and Karan Malhotra. 2025. Hermes 4 technical report. *arXiv preprint arXiv:2508.18255*.

Chenglong Wang, Yang Gan, Yifu Huo, Yongyu Mu, Qiaozhi He, MuRun Yang, Bei Li, Tong Xiao, Chunliang Zhang, Tongran Liu, and JingBo Zhu. 2025. GRAM: A generative foundation reward model for reward generalization. In *Forty-second International Conference on Machine Learning*.

Jize Wang, Ma Zerun, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. 2024. GTA: A benchmark for general tool agents. In *NeurIPS 2024 Workshop on Open-World Agents*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, pages 24824–24837.

Chenxi Whitehouse, Tianlu Wang, Ping Yu, Xian Li, Jason Weston, Ilia Kulikov, and Swarnadeep Saha. 2025. J1: Incentivizing thinking in LLM-as-a-judge via reinforcement learning. *arXiv preprint arXiv:2505.10320*.

Ran Xu, Jingjing Chen, Jiayu Ye, Yu Wu, Jun Yan, Carl Yang, and Hongkun Yu. 2025. Incentivizing agentic reasoning in llm judges via tool-integrated reinforcement learning. *arXiv preprint arXiv:2510.23038*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, and 1 others. 2024. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik R Narasimhan. 2025. τ-bench: A benchmark for \underline{T}ool-\underline{A}gent-\underline{U}ser interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*.

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

Yue Yu, Zhengxing Chen, Aston Zhang, Liang Tan, Chenguang Zhu, Richard Yuanzhe Pang, Yundi Qian, Xuewei Wang, Suchin Gururangan, Chao Zhang, Melanie Kambadur, Dhruv Mahajan, and Rui Hou. 2025. Self-generated critiques boost reward modeling for language models. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 11499–11514, Albuquerque, New Mexico. Association for Computational Linguistics.

Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025. Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning. *arXiv preprint arXiv:2505.00024*.

Lucen Zhong, Zhengxiao Du, Xiaohan Zhang, Haiyi Hu, and Jie Tang. 2025. ComplexFuncBench: Exploring multi-step and constrained function calling under long-context scenario. *arXiv preprint arXiv:2501.10132*.

Table 3: Statistics for each constituent dataset. *Raw* and *Filtered* are reported by the number of original tasks, while *Segmented* counts the number of segmented trajectories, with *Msg* indicating their average number of messages. Trajectory patterns in each dataset are characterized from turn, step, and order perspectives: 'ST' and 'MT' denote 'single-turn' and 'multi-turn'; 'SS' and 'MS' denote 'single-step' and 'multi-step'; 'P' and 'S' denote 'parallel' and 'sequential', respectively.

Data Source	#Raw	#Filtered	#Segmented	#Msg	#Schemas	Pattern of Trajectory			Task Domain
						Turn	Step	Order	
APIGen	60,000	60,000	59,960	3.00	4,205	ST	SS/MS	P	Finance/Sports/Technology/Travel ...
APIGen-MT	5,000	4,874	20,055	11.75	26	MT	SS/MS	P/S	Airline/Retail
BUTTON	8,000	8,000	20,811	5.19	22,101	ST	SS/MS	P/S	Daily Life
ComplexFuncBench	1,000	1,000	3,259	5.43	40	ST	MS	S	Hotel/Flight/Attraction/Car Rental/Taxi
Glaive-Function-Calling	5,209	4,344	6,747	4.82	1,565	MT	SS/MS	P	Stocks and Orders/Movie/Flight Services ...
Hermes-Function-Calling	1,893	1,724	1,724	3.00	2,383	ST	SS/MS	P	Information Extraction/API Call/Software ...
ToolAlpaca	4,098	2,510	6,194	4.24	2,040	ST	SS/MS	P/S	News/Jobs/Finance/Entertainment ...

A The Use of Large Language Models

During the completion of this work, we employed Gemini 2.5 Pro (Comanici et al., 2025) to identify grammatical errors and refine the text in the preliminary draft stage. The data construction pipeline code was initially developed by the human authors and then verified using Qwen3-Coder (Yang et al., 2025). All suggestions from the LLMs were manually reviewed and confirmed for accuracy.

B Statistics of Data Sources

In this work, we utilize seven high-quality tool-use datasets with diverse distributions. Table 3 summarizes statistics for each data source, including the number of unique tool schemas and the distribution of tool-call trajectory patterns, measured by turn-, step-, and order-wise occurrences.

C Full Related Work

C.1 Tool Learning in the Era of LLMs

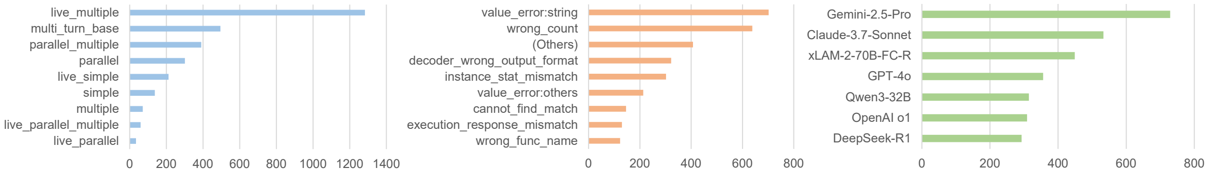
The emergence of foundational capabilities in large language models (LLMs) has enabled them to identify and use appropriate tools in a human-like manner. Yao et al. (2023) unlock this ability by combining chain-of-thought reasoning (Wei et al., 2022) with tool-augmented actions. Another line of approaches clones behaviors from completed tool-calling trajectories using supervised fine-tuning (Schick et al., 2023; Tang et al., 2023; Liu et al., 2024b, 2025b), while these methods may face challenges generalizing to complex and out-of-distribution tasks. To address this limitation, other approaches employ reinforcement learning with human preference data to learn via trial-and-error (Nakano et al., 2021). Building on recent successes in reasoning models (Lambert et al., 2025; Shao et al., 2024), utilizing verified rewards to facilitate tool-integrated reasoning has become a promising direction. Reward designs based on the format and correctness of the final answer have proven effective in tasks like question-answering (Jin et al., 2025; Song et al., 2025), math (Feng et al., 2025; Dong et al., 2025), and general tool-calling (Qian et al., 2025; Zhang et al., 2025; Jiang and Ferraro, 2026), leading to generalized model improvements through reinforcement learning.

C.2 Evaluation of LLM Tool-Use

Numerous tool-calling benchmarks have been proposed in recent years. To enable realistic and reliable evaluation, tasks are either drawn from real-world domains (Wang et al., 2024; Patil et al., 2024; Zhong et al., 2025; Yao et al., 2025; Barres et al., 2025) or generated via well-designed data-synthesis pipelines (Qin et al., 2024; Chen et al., 2025a). Among these, BFCL (Patil et al., 2025) covers diverse and complex patterns of tool usage and serves as a comprehensive benchmark for evaluating LLMs' tool-use capabilities. Nevertheless, there remains a lack of a benchmark that assesses whether current models can provide accurate feedback on LLM actions in tool-use scenarios. Recent work, FC-Reward-Bench (Agarwal et al., 2025), investigates reward model performance on function-calling tasks. However, it does not evaluate multi-turn tool-use scenarios, and its data pairs are too simplistic for powerful RMs to effectively distinguish between them.

C.3 Reward Modeling of Human Preferences

Reinforcement learning has proven effective for aligning LLMs with human preferences, using feedback from humans (Ouyang et al., 2022) or other capable LLMs (Bai et al., 2022; Lee et al., 2024). Central to this process are reward models (RMs),



(a) Distribution of data patterns. (b) Distribution of error types. (c) Distribution of response sources.

Figure 4: Statistics of the enhanced reward model benchmark TRBench$_{\mathrm{BFCL}}$.

which are primarily developed in two ways. The first is discriminative modeling, where RMs output a scalar score to differentiate between preferred and rejected responses (Yang et al., 2024; Cai et al., 2024; Liu et al., 2024a, 2025c). The second is generative modeling, where models provide textual rewards as natural language critiques for tasks like chat (Shiwen et al., 2024; Kim et al., 2024; Yu et al., 2025), code (McAleese et al., 2024), and literary machine translation (Pombal et al., 2025). Hybrid approaches combine critiques with scalar rewards to better capture nuanced preferences (Ankner et al., 2024; Wang et al., 2025), while recent work frames reward modeling as reasoning tasks (Chen et al., 2025c; Wang et al., 2025; Guo et al., 2025b; Whitehouse et al., 2025). In this paper, we extend reward modeling to the agentic tool-use scenario.

Notably, there is also a line of work on tool-augmented reward modeling (Li et al., 2024; Findeis et al., 2025; Xu et al., 2025), which is conceptually distinct from ToolRM, with different motivations and inference procedures. In our setting, ToolRM is trained to evaluate another policy model's behavior on agentic tool-use tasks, and it relies solely on internal reasoning rather than invoking external tools during evaluation. By contrast, tool-augmented RMs are primarily designed for target tasks such as general QA, writing, and coding, where the policy model can complete the task without invoking any tools, and tools are instead called at evaluation time to improve the reliability of reward estimates. Consequently, these tool-augmented RMs do not apply to the scenario studied in this paper and are not directly comparable to our approach.

D Experiment Details

D.1 Reward Model Training

We train generative ToolRM on eight NVIDIA A100 80G GPUs. We perform one epoch of GRPO training using veRL (Sheng et al., 2025), with a learning rate of `1e-6` and a clip ratio of $\epsilon = 0.2$. At each training step, we sample a batch of 128 queries and generate 8 trajectories per query. Trajectory generation is handled by the vllm backend (Kwon et al., 2023), employing sampling hyper-parameters of `temperature=1.0`, `top_p=1.0`, and `top_k=-1`. Due to resource constraints, we limit the maximum prompt length to 16,384 tokens and the maximum response length to 4,096 tokens for model training.

All discriminative reward models in this paper are trained using OpenRLHF (Hu et al., 2025) with a learning rate of `4e-6` and a training batch size of 256. For each dataset, we train for 2 epochs using the BT objective and set the maximum prompt length to 16,384 tokens during training.

D.2 Benchmark Implementation

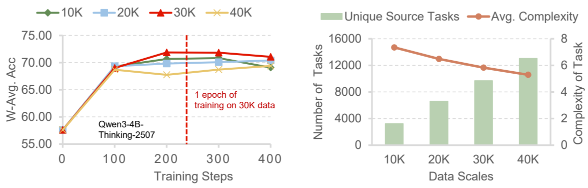
In constructing TRBENCH$_{\mathrm{BFCL}}$, all rejected LLM responses are extracted from the official evaluation archive[2], including xLAM-2-70B-FC-R (Prabhakar et al., 2025), GPT-4o (Hurst et al., 2024), OpenAI o1 (Jaech et al., 2024), Qwen3-32B (Yang et al., 2025), DeepSeek-R1 (Guo et al., 2025a), Gemini-2.5-Pro (Comanici et al., 2025), and Claude-3.7-Sonnet (Anthropic, 2025). We prepare preference pairs for each data task according to its turn-wise trajectory pattern. For single-turn tasks (splits originally introduced in BFCL v1 and v2), evaluation is based on the Abstract Syntax Tree (AST), which compares a model-generated function against its function documentation and a set of possible correct answers. In these cases, we source the oracle answers directly from the benchmark as the *chosen* responses and extract incorrect responses from the failed trajectories, forming *chosen–rejected* pairs for each task.

For multi-turn tasks (the split introduced in BFCL v3), evaluation instead relies on state-based and response-based checks, which differ from the rule-based matching used to check tool calls in

[2]Trajectories from `https://github.com/HuanzhiMao/BFCL-Result`

Table 4: Detailed policy model accuracy (%) on BFCL-v3 before and after RL training with ToolRM-Gen.

Model	Non-Live AST				Live AST				Multi-Turn			
	Non-Live Simple	Non-Live Multiple	Non-Live Parallel	Non-Live Parallel Multiple	Live Simple	Live Multiple	Live Parallel	Live Parallel Multiple	Multi Turn Base	Multi Turn Miss Func	Multi Turn Miss Param	Multi Turn Long Context
Qwen3-4B-Instruct-2507	73.83	94.50	91.00	88.50	79.84	76.26	56.25	75.00	27.00	10.50	18.00	24.00
- RL w/ ToolRM-Gen	71.75	94.50	90.50	87.50	81.78	78.25	68.75	62.50	41.00	9.50	20.50	31.00



(a) Evaluation results on $\text{TRBench}_{\text{BFCL}}$.

(b) Statistics across different data scales.

Figure 5: Statistics and the impact of data scale on model training.

building $\mathcal{D}_{\text{pref}}$[3]. In these complex scenarios, while pinpointing the single failing tool call is difficult, one can easily identify the entire incorrect turn by comparing the generated trajectory to the ground truth. We leverage this to create evaluation pairs: the incorrect output is the concatenation of all tool calls the model generated in that turn, and the correct output is the concatenation of all tool calls from the corresponding ground-truth solution. We show statistics of the enhanced reward model benchmark $\text{TRBench}_{\text{BFCL}}$ in Figure 4.

To ensure fair evaluation across different types of baseline models, we first apply the same *think-mode/no-think-mode* template used in our model evaluations. If the test model is unable to follow the specific instruction, we instead evaluate it using its official prompt. To fully harness the potential of the test models, the official default sampling parameters are used for inference, except that the maximum output length is limited to 8,192 tokens to prevent excessively long and repetitive chain-of-thought content.

D.3 Policy Model Training

In the policy-model RL training phase using ToolRM as the reward model, we begin by collecting 10,000 multi-turn agent tool-use trajectories spanning 2,000 unique tool schemas. For each trajectory, we randomly select one tool-call assistant response and construct a training query from all preceding messages, discarding any subsequent messages (including the selected assistant response).

[3] https://gorilla.cs.berkeley.edu/blogs/13_bfcl_v3_multi_turn.html

Table 5: Ablated evaluation results on $\text{TRBench}_{\text{BFCL}}$.

Model	W-Avg. Acc
Full ToolRM-Gen	71.87
- w/o Full BMDS	67.24 (-4.63)
- w/o DDS	68.64 (-3.23)
- w/o CPI	70.29 (-1.58)
- w/o CT	68.89 (-2.98)
- w/o EC	68.69 (-3.18)

This yields 11,500 multi-turn queries and 3,500 single-turn queries. We then run two epochs of GRPO training with a learning rate of 2e-6 and a clip ratio of $\epsilon = 0.2$. At each training step, we sample a batch of 512 queries and generate 8 trajectories per query. During policy training, we cap the maximum prompt length at 16,384 tokens and the maximum response length at 2,048 tokens. Table 4 reports detailed BFCL-v3 policy model accuracy before and after RL training with ToolRM-Gen.

E Impact of Data Scaling on ToolRM

We investigate the influence of data scaling on model performance. Figure 5a shows the results for Qwen3-4B-Thinking-2507 on $\text{TRBench}_{\text{BFCL}}$, trained with data samples ranging from 10K to 40K. Notably, the model achieves its highest performance with 30K training samples. Performance does not increase monotonically with data size because our sampling strategy prioritizes more complex tasks. As the dataset grows, the average task complexity declines, leading to less effective training signals. Figure 5b illustrates this trend: while the number of unique tasks rises with larger datasets, their average complexity decreases. These results demonstrate that our proposed strategy successfully balances task diversity and complexity when exploring the candidate data pool.

F Ablation Studies on Preference Data Construction

To assess the contribution of our two key data construction components, we conduct an ablation study with two sets of variants. In the first set, we replace balanced multi-dimensional sampling with random sampling (*w/o BMDS*) and perform fine-

Table 6: Evaluation results of different variants on $\text{TRBench}_{\text{BFCL}}$.

Model	W-Avg. Acc
Qwen3-4B-Instruct-2507	59.67
- GenRM on *NormalPref*	63.82 (+4.15)
- GenRM on *ToolPref*	**66.85 (+7.18)**
- DiscRM on *NormalPref*	67.88 (+8.21)
- DiscRM on *ToolPref*	**77.61 (+17.94)**
Qwen3-4B-Thinking-2507	57.59
- GenRM on *NormalPref*	63.19 (+5.60)
- GenRM on *ToolPref*	**71.87 (+14.28)**
- DiscRM on *NormalPref*	69.69 (+12.10)
- DiscRM on *ToolPref*	**76.80 (+19.21)**

grained ablations along three critical dimensions: diversity of data sources (*w/o DDS*), coverage of preference intensity (*w/o CPI*), and complexity of tasks (*w/o CT*). In the second set, we remove the unified evaluation criteria during training (*w/o EC*). Models are trained using GRPO on Qwen3-4B-Thinking-2507 with 30K pairwise preferences, keeping all other settings fixed. As shown in Table 5, removing either component significantly degrades performance. Each BMDS dimension contributes to performance; diversity of data sources and task complexity have larger effects than preference intensity, underscoring the importance of both diversity and contextual complexity for reward-model training. Moreover, output length of models decreases sharply without the evaluation criteria (1,204→694), suggesting these criteria promote more comprehensive reasoning during training.

G Ablation Studies on Model Training

Data Domain. We investigate the influence of in-domain preference data on reward model performance by conduct the following experiments: (i) we randomly sample 30,000 instances from *Skywork-Reward-Preference-80K-v0.2* (Liu et al., 2024a), a high-quality general preference dataset; (ii) we make minimal modifications to ToolRM prompt template (removing the original evaluation criteria) and use it to perform RL training on the baseline models in the same way as for previous ToolRM; (iii) the trained models are then evaluated on $\text{TRBench}_{\text{BFCL}}$ where evaluation results are labeled with *NormalPref* in Table 6. According to the results, models trained on high-quality normal preference data do improve their judging performance on pairwise classification tasks in the tool-use domain. However, the in-domain preference dataset delivers substantially larger gains over base

Table 7: Evaluation results of different variants on ACEBench.

Model	Acc
Qwen3-4B-Instruct-2507	63.4
- BoN-16 w/ ToolRM-Gen	66.6 (+3.2)
- BoN-16 w/ ToolRM-Disc	67.2 (+3.8)
- SC w/ ToolRM-Gen	**74.8 (+11.4)**

models, particularly when training from a think-version base model.

Training Objective. Following Liu et al. (2024a), we further investigate the impact of ToolPref-Pairwise-30K on training discriminative reward models (DiscRM) using the Bradley–Terry (BT) objective. As shown in Table 6, the constructed dataset remains effective under BT objective and can further improve RM performance compared with the RL objective in pairwise preference classification tasks. This is consistent with our previous findings on the Skywork-Critic/Reward model series: when using the same base model and training data, DiscRM trained with a BT objective naturally produces more accurate relative scores than GenRM. We also observe that instruct-tuned base models, which produce more concise outputs, are better suited to train DiscRM with a BT objective for generating precise scores, whereas think-version models, which produce longer initial chain of thoughts and exhibit stronger exploration capability, are better suited for RL training to obtain GenRM with stronger analytical ability. As shown in Table 7, ToolRM-Disc yields larger gains when used to judge best-of-N sampling, whereas ToolRM-Gen is substantially more effective when used to provide self-correction feedback. In practice, each training objective has distinct strengths and should be chosen according to the application scenario: use GenRM when critique-style feedback and interpretability are required (e.g., self-correction), and use DiscRM when only accurate reward scoring is needed (e.g., RL training or BoN sampling).

H The Balanced Multi-Dimensional Sampling Algorithm

In this section, we detail the implementation of the BMDS strategy for efficient sampling. To discretize the distribution of preference intensities $I_{\text{preference}}$ among data samples, we initialize a set of bins $B = \{b_0, b_1, \ldots, b_m\}$ with fixed intervals. In our experiments, we set: $B = \{(0, 0.1], (0.1, 0.2], \ldots, (0.9, 1]\}$. Each sample in the candidate pairwise data pool $\mathcal{D}_{\text{pair-cand}}$ is assigned to the corresponding bin, indexed from 0 to m, according to its preference intensity. We then group the samples by a composite key (`source`, `bin_index`) to ensure representation across different data sources and varying preference intensities. Within each group, samples are sorted in descending order of task complexity $S_{\text{complexity}}$. Sampling proceeds greedily: we first exhaustively select all samples from the group with the fewest entries, and then allocate the remaining quota as evenly as possible across the other bins. This yields a diverse, well-balanced, and sufficiently challenging subset of data. We present pseudocode of this strategy in Algorithm 1.

I Example of Tool-Use Task Trajectory

During conversation order validation, we retain only trajectories that satisfy the following message-role transition rules: [`system`→`user`, `user`→`assistant`, `assistant`→`user`/`tool`, `tool`→`assistant`]. In this work, tool responses are set into `user` messages for compatibility. Figure 6 shows a format-aligned example from BUTTON illustrating a tool-use task trajectory.

J Prompting Templates

We present the evaluator prompt templates for the pairwise critique task used in both training and inference. The *think-mode* and *no-think-mode* templates are shown in Figures 7 and 8, respectively. Figure 9 shows the prompt template used by the judge for the BoN sampling task, in which the N sampled responses are inserted and labeled from 1 to N. Figures 10 and 11 show the prompt templates used by critic and editor for the self-correction task. Figure 12 presents the template of the system prompt in each tool-use trajectory.

K Case Studies

Valid Cases Through representative valid cases, we compare critiques from Claude 4 Sonnet and ToolRM-Gen-Qwen3-4B-Thinking-2507 on TRBench$_{\text{BFCL}}$ test samples. In the case shown in Figure 13, ToolRM accurately distinguishes correct from incorrect tool-call parameters without inducing "overthinking" hallucinations when the user query plausibly maps to multiple candidate parameters. Another case in Figure 14 further demonstrates its tendency to ground analysis in contextual rationale rather than engage in speculative, divergent reasoning. Moreover, as shown in Figure 15, ToolRM adheres more closely to the evaluation criteria, preferring tool calls without redundant parameters. Taken together, these behaviors enable ToolRM to deliver reliable critiques in tool-use scenarios.

Error Cases We also present representative error cases of ToolRM-Gen on TRBench$_{\text{BFCL}}$, as discussed in Section 4.5. Figure 16 illustrates an error of type (i), where the model fails to reason correctly given an underspecified tool parameter description, while Figure 17 illustrates an error of type (ii), where the model fails to distinguish primary errors from secondary ones.

Algorithm 1 Balanced Multi-Dimensional Sampling Strategy

Iutput: Data pool $\mathcal{D}_{\text{pair-cand}}$, bin edges B, target sample size N
Output: A subset $\mathcal{D}_{\text{pair-sampled}}$ of diverse, balanced, and challenging samples

 # Step 0: Check data sufficiency
if $|\mathcal{D}_{\text{pair-cand}}| < N$ **then**
  raise InsufficientDataError
end if
 # Step 1: Assign samples to bins
for each $d_i \in \mathcal{D}_{\text{pair-cand}}$ **do**
  $d_i.\texttt{bin\_idx} \leftarrow \text{assign}(d_i.I_{\text{preference}}, B)$
end for
 # Step 2: Group by composite key
Initialize group dictionary $\mathcal{G} \leftarrow \emptyset$
for each $d_i \in \mathcal{D}_{\text{pair-cand}}$ **do**
  $key \leftarrow (d_i.\texttt{source}, d_i.\texttt{bin\_idx})$
  $\mathcal{G}[key] \leftarrow \mathcal{G}[key] \cup \{d_i\}$
end for
 # Step 3: Sort within each group by task complexity (descending)
for each group $G \in \mathcal{G}$ **do**
  $G \leftarrow \text{sort}(G, \text{key} = S_{\text{complexity}}, \text{order=descending})$
end for
 # Step 4: Sort groups by size (ascending)
$\mathcal{G}_{\text{sorted}} \leftarrow \text{sort}(\mathcal{G}.\text{values}(), \text{key} = |G|, \text{order=ascending})$
 # Step 5: Greedy allocation
Initialize sampling quotas: $Q \leftarrow [0] \times |\mathcal{G}_{\text{sorted}}|$
$N_{\text{remaining}} \leftarrow N$, $k \leftarrow 0$
while $k < |\mathcal{G}_{\text{sorted}}|$ **and** $N_{\text{remaining}} > 0$ **do**
  $m \leftarrow |\mathcal{G}_{\text{sorted}}| - k$
  $n_{\text{avg}} \leftarrow \lceil N_{\text{remaining}}/m \rceil$
  if $|\mathcal{G}_{\text{sorted}}[k]| \leq n_{\text{avg}}$ **then**
    $Q[k] \leftarrow |\mathcal{G}_{\text{sorted}}[k]|$
    $N_{\text{remaining}} \leftarrow N_{\text{remaining}} - |\mathcal{G}_{\text{sorted}}[k]|$
    $k \leftarrow k + 1$
  else
    # Distribute remaining quota evenly
    $q \leftarrow \lfloor N_{\text{remaining}}/m \rfloor$
    $r \leftarrow N_{\text{remaining}} \mod m$
    for $i = k$ **to** $|\mathcal{G}_{\text{sorted}}| - 1$ **do**
      $Q[i] \leftarrow q$
    end for
    for $i = 0$ **to** $r - 1$ **do**
      $Q[|\mathcal{G}_{\text{sorted}}| - 1 - i] \leftarrow Q[|\mathcal{G}_{\text{sorted}}| - 1 - i] + 1$
    end for
    break
  end if
end while
 # Step 6: Sample data based on quotas
$\mathcal{D}_{\text{pair-sampled}} \leftarrow \emptyset$
for $i = 0$ **to** $|\mathcal{G}_{\text{sorted}}| - 1$ **do**
  $\mathcal{D}_{\text{pair-sampled}} \leftarrow \mathcal{D}_{\text{pair-sampled}} \cup \mathcal{G}_{\text{sorted}}[i][: Q[i]]$
end for
return $\mathcal{D}_{\text{pair-sampled}}$

A Tool-Call Trajectory from BUTTON

```
{
    "role": "system",
    "content": "# Tools

     You may call one or more functions to assist with the user query.
     You are provided with function signatures within <tools></tools> XML tags:
     <tools>
     {"type": "function", "function": {"name": "get_items_by_color", "description": "Retrieve all items in the inventory filtered by a specified color.", "parameters": {"required": ["color", "date"], "type": "object", "properties": {"color": {"type": "string", "description": "The color of the items to be retrieved."}, "date": {"type": "string", "description": "The date for which the inventory is being checked, in YYYY-MM-DD format."}}}}}
     {"type": "function", "function": {"name": "count_items", "description": "Count the number of items in a given list.", "parameters": {"required": ["items"], "type": "object", "properties": {"items": {"type": "array", "description": "The list of items to be counted."}}}}}
     </tools>

     For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
     <tool_call>
     {"name": <function-name>, "arguments": <args-json-object>}
     </tool_call>"
},
{
    "role": "user",
    "content": "Hi, I need to count the red and blue items in the warehouse inventory today and compare their quantities. The warehouse inventory data should have information about item colors and quantities. Can you help me with this?"
},
{
    "role": "assistant",
    "content": "<think>
     The user wants to count the red and blue items in the warehouse inventory today and compare their quantities. To accomplish this task, we need to:
     1. Retrieve the total number of red items in the warehouse inventory today.
     2. Retrieve the total number of blue items in the warehouse inventory today.
     3. Compare the counts of red and blue items.
     4. Provide the final answer based on the comparison.
     We will start by retrieving and counting the red items first.
     </think>
     <tool_call>
     {"name": "get_items_by_color", "arguments": {"color": "red", "date": "2023-10-05"}}
     </tool_call>"
},
{
    "role": "user",
    "content": "<tool_response>
     {"name": "get_items_by_color", "arguments": {"color": "red", "date": "2023-10-05"}, "results": {"items": [{"id": "001", "name": "Red T-shirt", "color": "red", "quantity": 50, "date_added": "2023-09-20"}, {"id": "002", "name": "Red Mug", "color": "red", "quantity": 120, "date_added": "2023-10-01"}, {"id": "003", "name": "Red Hat", "color": "red", "quantity": 30, "date_added": "2023-09-29"}]}}
     </tool_response>"
},
{
    "role": "assistant",
    "content": "<think>
     The function `get_items_by_color` returned a list of red items along with their quantities for the date 2023-10-05. The items are:
     - Red T-shirt: Quantity 50
     - Red Mug: Quantity 120
     - Red Hat: Quantity 30 We need to count the total quantity of these red items. We'll sum up the quantities of all the red items returned.
     </think>
     <tool_call>
     {"name": "count_items", "arguments": {"items": [{"id": "001", "name": "Red T-shirt", "color": "red", "quantity": 50, "date_added": "2023-09-20"}, {"id": "002", "name": "Red Mug", "color": "red", "quantity": 120, "date_added": "2023-10-01"}, {"id": "003", "name": "Red Hat", "color": "red", "quantity": 30, "date_added": "2023-09-29"}]}}
     </tool_call>"
},
{
    "role": "user",
    "content": "<tool_response>
     {"name": "count_items", "arguments": {"items": [{"id": "001", "name": "Red T-shirt", "color": "red", "quantity": 50, "date_added": "2023-09-20"}, {"id": "002", "name": "Red Mug", "color": "red", "quantity": 120, "date_added": "2023-10-01"}, {"id": "003", "name": "Red Hat", "color": "red", "quantity": 30, "date_added": "2023-09-29"}]}, "results": {"count": 3}}
     </tool_response>"
  },
{
   "role": "assistant",
   "content": "<think>
     The `count_items` function returned a count of 3 for the red items. However, the task requires summing the quantities of the items, not just counting the number of different items. We need to manually sum the quantities of the red items, as the `count_items` function didn't provide the total quantity but rather the count of item types. The quantities are 50, 120, and 30. The total quantity of red items is (50 + 120 + 30 = 200).
     Next, we should retrieve and count the blue items in the warehouse inventory for today.
     </think>
     <tool_call>
     {"name": "get_items_by_color", "arguments": {"color": "blue", "date": "2023-10-05"}}
     </tool_call>"
}
```

Figure 6: A format-aligned tool-use trajectory from BUTTON.

Serve as an Evaluator for Pairwise Critique Task (Think Mode)

<task>
You are an expert evaluator of AI assistant performance. Given a complete user-assistant conversation history and two generated assistant responses, you are to conduct a thorough, fact-based, and comprehensive comparison. Based on specific evidence from your evaluation, make a clear choice of which response is superior. There may be a list of tools available to the assisatnt. The assistant starts with one or more cycles of (thinking about which tool to use -> performing tool call -> waiting for tool response), and ends with (thinking about the answer -> answer of the question). The thinking processes, tool calls, tool responses, and answer are enclosed within their tags. There could be multiple thinking processes, tool calls, tool call parameters and tool response parameters.
</task>

<evaluation_criteria>
- Available tools must be fully and appropriately leveraged to meet the requirements.
- Tool call names must be valid, correct, and complete.
- Tool call arguments must be valid, correct, and complete.
- Fabrication, including the creation of information or knowledge not provided by the user, conflicting with user input, or not derived from the tools, must be penalized.
- Repetitive or unnecessary tool calls must be penalized.
- Excessive or unnecessary requests for user clarification beyond what is essential must be penalized.
</evaluation_criteria>

<conversation_history>
{chat_history}
</conversation_history>

<current_response_1>
{assistant_response_1}
</current_response_1>

<current_response_2>
{assistant_response_2}
</current_response_2>

Output your choice (either '1' or '2') within <choice></choice> XML tags. No explanations should precede or follow the choice. Answer in the following format.
<choice>
{{your_choice}}
</choice>

Figure 7: Evaluator prompt template of the pairwise critique task for reasoning LLMs.

Serve as an Evaluator for Pairwise Critique Task (No Think Mode)

<task>
You are an expert evaluator of AI assistant performance. Given a complete user-assistant conversation history and two generated assistant responses, you are to conduct a thorough, fact-based, and comprehensive comparison. Based on specific evidence from your evaluation, make a clear choice of which response is superior. There may be a list of tools available to the assisatnt. The assistant starts with one or more cycles of (thinking about which tool to use -> performing tool call -> waiting for tool response), and ends with (thinking about the answer -> answer of the question). The thinking processes, tool calls, tool responses, and answer are enclosed within their tags. There could be multiple thinking processes, tool calls, tool call parameters and tool response parameters.
</task>

<evaluation_criteria>
- Available tools must be fully and appropriately leveraged to meet the requirements.
- Tool call names must be valid, correct, and complete.
- Tool call arguments must be valid, correct, and complete.
- Fabrication, including the creation of information or knowledge not provided by the user, conflicting with user input, or not derived from the tools, must be penalized.
- Repetitive or unnecessary tool calls must be penalized.
- Excessive or unnecessary requests for user clarification beyond what is essential must be penalized.
</evaluation_criteria>

<conversation_history>
{chat_history}
</conversation_history>

<current_response_1>
{assistant_response_1}
</current_response_1>

<current_response_2>
{assistant_response_2}
</current_response_2>

Output your evaluation within <evaluation></evaluation> XML tags, and then enclose your choice (either '1' or '2') within <choice></choice> XML tags. Answer in the following format.
<evaluation>
{{your_evaluation}}
</evaluation>
<choice>
{{your_choice}}
</choice>

Figure 8: Evaluator prompt template of the pairwise critique task for non-reasoning LLMs.

Serve as A Judge for Best-of-N Sampling (Think Mode)

<task>
You are an expert evaluator of AI assistant performance. Given a complete user-assistant conversation history and {N} generated assistant responses, you are to conduct a thorough, fact-based, and comprehensive comparison. Based on specific evidence from your evaluation, make a clear choice of which response is superior. If multiple responses are identical and equally the best, select the one with the smallest number.
</task>

<evaluation_criteria>
- Available tools must be fully and appropriately leveraged to meet the requirements.
- Tool call names must be valid, correct, and complete.
- Tool call arguments must be valid, correct, and complete.
- Fabrication, including the creation of information or knowledge not provided by the user, conflicting with user input, or not derived from the tools, must be penalized.
- Repetitive or unnecessary tool calls must be penalized.
- Excessive or unnecessary requests for user clarification beyond what is essential must be penalized.
</evaluation_criteria>

<conversation_history>
{chat_history}
</conversation_history>

<current_response_1>
{assistant_response_1}
</current_response_1>

<current_response_2>
{assistant_response_2}
</current_response_2>
…
<current_response_{N}>
{assistant_response_N}
</current_response_{N}>

Output your choice (a number between 1 and {N}) within <choice></choice> XML tags. No explanations should precede or follow the choice. Answer in the following format.
<choice>
{{your_choice}}
</choice>

Figure 9: Judge prompt template of the Best-of-N sampling task for reasoning LLMs.

Serve as a Critic for Self-Correction (Think Mode)

<task>
You are an expert evaluator of AI assistant performance. Given a complete user-assistant conversation history and a generated assistant response, you are to conduct a thorough, fact-based, and comprehensive evaluation. Based on specific evidence from your evaluation, provide a concise critique on how the current assistant response should be revised. If the response is entirely correct and requires no changes, output '[correct]' as your critique.
</task>

<evaluation_criteria>
- Available tools must be fully and appropriately leveraged to meet the requirements.
- Tool call names must be valid, correct, and complete.
- Tool call arguments must be valid, correct, and complete.
- Fabrication, including the creation of information or knowledge not provided by the user, conflicting with user input, or not derived from the tools, must be penalized.
- Repetitive or unnecessary tool calls must be penalized.
- Excessive or unnecessary requests for user clarification beyond what is essential must be penalized.
</evaluation_criteria>

<conversation_history>
{chat_history}
</conversation_history>

<current_response>
{assistant_response}
</current_response>

Output your final critique within <critique></critique> XML tags. No explanations should precede or follow the critique. Answer in the following format.
<critique>
{{your_critique}}
</critique>

Figure 10: Critic prompt template of the self-correction task for reasoning LLMs.

Serve as an Editor for Self-Correction (No Think Mode)

<task>
You are an expert editor of AI assistant response. Given a complete user-assistant conversation history, a generated assistant response, and a critique about how to improve it, your task is to produce the revised response.
</task>

<conversation_history>
{chat_history}
</conversation_history>

<current_response>
{assistant_response}
</current_response>

<critique>
{critique}
</critique>

Output the revised response within <revised_response></revised_response> XML tags. No explanations should precede or follow the response. Answer in the following format.
<revised_response>
{{revised_response}}
</revised_response>

Figure 11: Editor prompt template of the self-correction task for non-reasoning LLMs.

System Prompt in Tool-Use Trajectory

Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{tool_descs}
</tools>

For each function call, return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{{"name": <function-name>, "arguments": <args-json-object>}}
</tool_call>

Agent Policy
{agent_policy}

Figure 12: System prompt template of the tool-use trajectory.

A Case from TRBench$_{\text{BFCL}}$

Prompt of the Pairwise Critique Task

…
<conversation_history>
[system]: # Tools …
<tools>
{"type": "function", "function": {"name": "add_mtnards_server", "description": "Registers a new MTNA Rich Data Services (RDS) server with the specified nickname and host within the environment, using the provided API key for authentication.", …}}}}

{"type": "function", "function": {"name": "list_servers", "description": "Retrieve a list of all the servers within the specified environment, optionally filtering by server type.", "parameters": {"type": "dict", "properties": {"server_type": {"type": "string", "description": "The type of servers to be listed. If not specified, all server types are included.", "enum": ["all", "graphql", "mtna", "openapi", "postgres", "rds", "sql"], "default": "all"}}, "required": []}}}
…
</tools>
…
[user]: Could you retrieve a list of all the MTNA Rich Data Services servers for me?
</conversation_history>

<current_response_1>
<tool_call>
{"name": "list_servers", "arguments": {"server_type": **"mtna"**}}
</tool_call>
</current_response_1>

<current_response_2>
<tool_call>
{"name": "list_servers", "arguments": {"server_type": **"rds"**}}
</tool_call>
</current_response_2>

Response from ToolRM

<think>
Okay, let's tackle this evaluation. So, the user is asking for a list of all MTNA Rich Data Services servers.
First, I need to check the available tools. The tools list has a function called list_servers which can take a server_type parameter. The possible server types include "mtna" as one of the options.
Looking at Response 1: They used list_servers with server_type "mtna". That seems correct because the user is asking for MTNA Rich Data Services servers. MTNA probably stands for MTNA Rich Data Services, so the server_type should be "mtna".
Response 2: They used list_servers with server_type "rds". Wait, rds is another server type. The user is asking for MTNA servers, not RDS servers. So Response 2 is incorrect here. The tool parameter should be "mtna" to get MTNA servers, not "rds".
Also, checking the tools documentation: The list_servers function has a server_type enum that includes "mtna" and "rds" among others. So the correct server_type for MTNA Rich Data Services is "mtna".
Therefore, Response 1 is correct because it uses server_type "mtna", which matches the user's request. Response 2 uses "rds" which is wrong because the user is asking for MTNA servers, not RDS servers. So the superior response is Response 1.
</think>
<choice>
1
</choice>

Response from Claude-4-Sonnet

<think>
The user is asking for a list of all the MTNA Rich Data Services servers.
Looking at the tools available, there's a `list_servers` function that can "Retrieve a list of all the servers within the specified environment, optionally filtering by server type."
The parameters show that `server_type` is an optional parameter with possible values: ["all", "graphql", "mtna", "openapi", "postgres", "rds", "sql"].
The user specifically asked for "MTNA Rich Data Services servers".
Let me analyze this:
1. MTNA Rich Data Services - this seems to be a specific type of service
2. Looking at the `add_mtnards_server` function, it mentions "Registers a new MTNA Rich Data Services (RDS) server"
So MTNA Rich Data Services = RDS server type according to the function description.
Now let me look at the two responses:
Response 1: `{"name": "list_servers", "arguments": {"server_type": "mtna"}}`
Response 2: `{"name": "list_servers", "arguments": {"server_type": "rds"}}`
From the `add_mtnards_server` function description: "Registers a new MTNA Rich Data Services (RDS) server"
This clearly indicates that MTNA Rich Data Services servers are referred to as "rds" type servers in the system, not "mtna" type servers.
So Response 2 is correct - it uses "rds" as the server_type, which matches what the system calls MTNA Rich Data Services servers.
Response 1 uses "mtna" which doesn't appear to be the correct server type designation for MTNA Rich Data Services servers based on the function descriptions.
</think>
<choice>
2
</choice>

Figure 13: Valid *case 1* from TRBench$_{\text{BFCL}}$, featuring responses from ToolRM-Gen and Claude 4 Sonnet.

A Case from TRBench$_{\text{BFCL}}$

Prompt of the Pairwise Critique Task

…
<conversation_history>
[system]: # Tools …
<tools>
{"type": "function", "function": {"name": "CustomDashboardsApi.get_custom_dashboards", "description": "Retrieves a list of custom dashboards that the user has access to, based on the provided query parameters.", "parameters": {"type": "dict", "required": ["query"], "properties": {"query": {"type": "string", "description": "A search query string to filter accessible dashboards. Supports logical operators and field-specific filtering."}, "pageSize": {"type": "integer", "description": "The number of dashboard entries to return per page.", "default": 10}, "page": {"type": "integer", "description": "The page number to retrieve, used for paginating through large sets of dashboard entries.", "default": 1}}}}}

{"type": "function", "function": {"name": "CustomDashboardsApi.get_custom_dashboard", "description": "Retrieve the details of a custom dashboard by its unique identifier.", "parameters": {"type": "dict", "required": ["customDashboardId"], "properties": {"customDashboardId": {"type": "string", "description": "The unique identifier of the custom dashboard to retrieve."}}}}}
…
</tools>
…
[user]: Get the dashboard for "dashing".
</conversation_history>

<current_response_1>
<tool_call>
{"name": "**CustomDashboardsApi.get_custom_dashboards**", "arguments": {"**query**": "dashing"}}
</tool_call>
</current_response_1>

<current_response_2>
<tool_call>
{"name": "**CustomDashboardsApi.get_custom_dashboard**", "arguments": {"**customDashboardId**": "dashing"}}
</tool_call>
</current_response_2>

Response from ToolRM

<think>
Okay, let's tackle this evaluation. So, the user wants to get the dashboard for "dashing". The available tools are three functions: add_custom_dashboard, get_custom_dashboards, and get_custom_dashboard.
First, looking at the conversation history, the user's query is "Get the dashboard for 'dashing'". The assistant needs to use the tools correctly.
Looking at Response 1: They used get_custom_dashboards with query "dashing". That seems right because the get_custom_dashboards function takes a query parameter to find dashboards. The parameters are correct here.
Response 2: They used get_custom_dashboard with customDashboardId "dashing". Wait, the get_custom_dashboard function requires a customDashboardId, which is a unique identifier. But the user's query is "Get the dashboard for 'dashing'", which might mean they want a dashboard named "dashing" or with a title containing "dashing". However, the tool get_custom_dashboard needs a specific ID, not a name. The problem is that the user didn't provide an ID, so the assistant can't know the ID beforehand. Therefore, using get_custom_dashboard with ID "dashing" might be incorrect because the ID is a unique identifier, not the dashboard name. The correct approach would be to first use get_custom_dashboards to find the dashboard by name (query "dashing") and then get its ID to use in get_custom_dashboard. But in this case, the user's request is to get the dashboard for "dashing", so maybe they mean a dashboard named "dashing".
Wait, the tools description says for get_custom_dashboards, the query parameter is a search string. So Response 1 is correct because they're searching for dashboards with query "dashing".
Response 2 is trying to get a dashboard by ID "dashing", but the ID is a unique identifier that isn't necessarily the same as the name.
Since the user said "dashboard for 'dashing'", it's more likely they want a dashboard whose name is "dashing" or contains "dashing".
So the correct tool here is get_custom_dashboards with query "dashing" (Response 1). Response 2 is using get_custom_dashboard with an ID that's probably not valid because the ID is a unique identifier, not a name. The tool get_custom_dashboard requires a specific ID, which the user didn't provide. So Response 2 is incorrect because they're assuming "dashing" is a valid ID, but the tool's parameters need a real ID. Therefore, Response 1 is superior.
</think>
<choice>
1
</choice>

Response from Claude-4-Sonnet

<think>
Let me analyze both responses based on the evaluation criteria.
The user is asking: "Get the dashboard for 'dashing'."
I need to evaluate which tool call is more appropriate.
Response 1 uses: `CustomDashboardsApi.get_custom_dashboards` with arguments `{"query": "dashing"}`
Response 2 uses: `CustomDashboardsApi.get_custom_dashboard` with arguments `{"customDashboardId": "dashing"}`
Let me look at the function signatures:
1. `CustomDashboardsApi.get_custom_dashboards`:
 - Description: "Retrieves a list of custom dashboards that the user has access to, based on the provided query parameters."
 - Required parameters: ["query"]
 - This function searches for dashboards using a query string
2. `CustomDashboardsApi.get_custom_dashboard`:
 - Description: "Retrieve the details of a custom dashboard by its unique identifier."
 - Required parameters: ["customDashboardId"]
 - This function retrieves a specific dashboard by its ID
Now, the user said "Get the dashboard for 'dashing'". The key question is: is "dashing" a dashboard ID or a search term? From the context, it's ambiguous. "Dashing" could be:
1. The unique identifier of a specific dashboard
2. A name or part of a name to search for
However, looking at the function descriptions:
- `get_custom_dashboard` is for retrieving a specific dashboard by its unique identifier
- `get_custom_dashboards` is for searching/filtering dashboards
The user said "Get THE dashboard for 'dashing'" - the use of "THE" suggests they're looking for a specific dashboard, not searching for multiple dashboards.
If "dashing" is indeed the dashboard ID, then Response 2 would be correct.
If "dashing" is a search term or dashboard name (not ID), then Response 1 would be correct.
The singular phrasing "the dashboard" implies a specific, unique dashboard. This strongly points to using the `get_custom_dashboard` method with "dashing" as the identifier.
I'll prioritize the most precise interpretation based on the user's language, which suggests a direct retrieval of a specific dashboard. The approach balances efficiency and user intent, targeting a direct dashboard fetch while maintaining flexibility if the identifier proves invalid.
The definite article implies a precise dashboard selection, making the direct retrieval method most aligned with the user's expectation. Searching becomes a secondary strategy if direct identification fails.
</think>
<choice>
2
</choice>

Figure 14: Valid *case 2* from TRBench$_{\text{BFCL}}$, featuring responses from ToolRM-Gen and Claude 4 Sonnet.

A Case from TRBench$_{BFCL}$

Prompt of the Pairwise Critique Task

…
<conversation_history>
[system]: # Tools …
<tools>
{"type": "function", "function": {"name": "get_shortest_driving_distance", "description": "Calculate the shortest driving distance between two locations.", "parameters": {"type": "dict", "properties": {"origin": {"type": "string", "description": "Starting point of the journey."}, "destination": {"type": "string", "description": "End point of the journey."}, "unit": {"type": "string", "description": "Preferred unit of distance (optional, default is kilometers)."}}, "required": ["origin", "destination"]}}}
…
</tools>
…
[user]: Find the shortest driving distance between New York City and Washington D.C.
</conversation_history>

<current_response_1>
<tool_call>
{"name": "get_shortest_driving_distance", "arguments": {"origin": "New York City", "destination": "Washington D.C."}}
</tool_call>
</current_response_1>

<current_response_2>
<tool_call>
{"name": "get_shortest_driving_distance", "arguments": {"origin": "New York City", "destination": "Washington D.C.", **"unit": "miles"**}}
</tool_call>
</current_response_2>

Response from ToolRM

<think>
Okay, let's tackle this evaluation. So, the user wants the shortest driving distance between New York City and Washington D.C. The available tools include get_shortest_driving_distance, which requires origin and destination. The other parameters have a unit option, defaulting to kilometers.

Looking at Response 1: They called the function with origin and destination, no unit specified. That's correct because the tool's parameters say unit is optional with default kilometers. So that's valid.

Response 2: They added "unit": "miles". The tool's description says the unit is optional with default kilometers. So specifying miles is okay, but does the tool accept that? The tool's parameters say unit is optional, so it's valid. However, the user's query doesn't specify a unit, so the assistant should probably use the default.
The tool's description says the default is kilometers, so maybe the assistant shouldn't change it unless needed. Since the user didn't ask for miles, Response 2 is adding an unnecessary parameter. That would be a redundancy. The evaluation criteria say to avoid unnecessary tool calls or parameters. So Response 2 has an extra parameter that's not needed, which is a drawback. Therefore, Response 1 is better because it doesn't add unnecessary info. The tool's parameters say unit is optional, but since the user didn't specify, the default is fine. So the superior response is Response 1.
</think>
<choice>
1
</choice>

Response from Claude-4-Sonnet

<think>
Let me analyze these two responses based on the evaluation criteria:
1. **Available tools must be fully and appropriately leveraged to meet the requirements.**
Both responses use the correct tool `get_shortest_driving_distance` which is appropriate for finding the shortest driving distance between two locations.
2. **Tool call names must be valid, correct, and complete.**
Both responses use the correct function name "get_shortest_driving_distance".
3. **Tool call arguments must be valid, correct, and complete.**
Let me check the function signature:
```
"parameters": {"type": "dict", "properties": {"origin": {"type": "string", "description": "Starting point of the journey."}, "destination": {"type": "string", "description": "End point of the journey."}, "unit": {"type": "string", "description": "Preferred unit of distance (optional, default is kilometers)."}}, "required": ["origin", "destination"]}
```

Response 1 arguments:
- "origin": "New York City" ✓
- "destination": "Washington D.C." ✓
- Missing "unit" parameter (but it's optional)

Response 2 arguments:
- "origin": "New York City" ✓
- "destination": "Washington D.C." ✓
- "unit": "miles" ✓
Both responses provide the required parameters. Response 2 also includes the optional "unit" parameter.
4. **Fabrication, including the creation of information or knowledge not provided by the user, conflicting with user input, or not derived from the tools, must be penalized.**
Neither response fabricates information. Both are making appropriate tool calls.
5. **Repetitive or unnecessary tool calls must be penalized.**
Neither response makes repetitive or unnecessary calls.
6. **Excessive or unnecessary requests for user clarification beyond what is essential must be penalized.**
Neither response requests clarification.
The key difference is that Response 2 includes the "unit" parameter set to "miles". Since the user asked for the distance between two US cities (New York City and Washington D.C.), it would be more natural and user-friendly to provide the result in miles rather than the default kilometers. While the "unit" parameter is optional, specifying "miles" would likely give a more useful result for someone asking about distances between US cities.
Response 2 demonstrates enhanced user-centricity by selecting miles as the distance unit, which aligns more naturally with US geographic conventions and user expectations.
</think>
<choice>
2
</choice>

Figure 15: Valid *case 3* from TRBench$_{BFCL}$, featuring responses from ToolRM-Gen and Claude 4 Sonnet.

A Case from TRBench$_{\text{BFCL}}$

Prompt of the Pairwise Critique Task

…
<conversation_history>
[system]: # Tools …
<tools>
{"type": "function", "function": {"name": "SyntheticSettingsApi.get_synthetic_locations", "description": "Retrieve a list of all synthetic locations available for monitoring. The response can be sorted, filtered, and paginated to obtain the desired subset of locations.", "parameters": {"type": "dict", "required": ["limit"], "properties": {"sort": {"type": "string", "description": "...", "enum": ["asc", "desc"], "default": "asc"}, "offset": {"type": "integer", "description": "…", "default": 0}, "limit": {"type": "integer", "description": "The maximum number of results to return. Must be a positive integer to limit the response size."}, "filter": {"type": "string", "description": "A filter string to apply to the results, such as filtering by location name or type.", "default": ""}}}}}
…
</tools>
[user]: Can you retrieve a list of all synthetic locations available for monitoring and filter the results for locations named 'alpha', 'beta', and 'gamma', with a limit of 20 results?
</conversation_history>

<current_response_1>
<tool_call>
{"name": "SyntheticSettingsApi.get_synthetic_locations", "arguments": {"limit": 20, "filter": "**name IN ('alpha', 'beta', 'gamma')**"}}
</tool_call>
</current_response_1>

<current_response_2>
<tool_call>
{"name": "SyntheticSettingsApi.get_synthetic_locations", "arguments": {"sort": "asc", "offset": 0, "limit": 20, "filter": "**alpha,beta,gamma**"}}
</tool_call>
</current_response_2>

Response from ToolRM

<think>
Okay, let's tackle this evaluation. So, the user wants a list of all synthetic locations filtered for 'alpha', 'beta', 'gamma' with a limit of 20 …
Therefore, Response 2 is better because its filter is simpler and more likely to be correctly interpreted by the tool. Also, the tool's description says "filter string to apply", so a comma-separated list of names is a common way to filter.
Also, check the tool's parameters: the filter parameter is a string. The user's request is to filter for locations named 'alpha', 'beta', and 'gamma'. So the filter should be those names. Response 2's filter is "alpha,beta,gamma", which is correct. Response 1's filter is "name IN ('alpha', 'beta', 'gamma')", which might be invalid because the tool might not understand that syntax.
So the superior response is Response 2.
</think>
<choice>
2
</choice>

Figure 16: Error *case 1* from TRBench$_{\text{BFCL}}$, featuring responses from ToolRM-Gen.

A Case from TRBench$_{\text{BFCL}}$

Prompt of the Pairwise Critique Task

…
<conversation_history>
[system]: # Tools …
<tools>
{"type": "function", "function": {"name": "send_message", "description": "Sends a message to a human user, supporting a full range of Unicode characters, including emojis.", "parameters": {"type": "dict", "properties": {"message": {"type": "string", "description": "The content of the message to be sent. Supports Unicode characters such as text and emojis."}, "recipient": {"type": "string", "description": "The identifier of the recipient. …", "default": "default@example.com"}, …, "required": ["message"]}}}

{"type": "function", "function": {"name": "recall_memory_search", "description": "Searches the prior conversation history for occurrences of a specified string and returns paginated results.", "parameters": {"type": "dict", "required": ["query"], "properties": {"query": {"type": "string", "description": "The search string to locate within the conversation history."}, "page": {"type": "integer", "description": "Specifies the page number of results to retrieve, where 0 represents the first page.", "default": 0}, "request_heartbeat": {"type": "boolean", "description": "When set to true, triggers an immediate heartbeat post function execution to facilitate follow-up actions.", "default": false}}}}}
…
</tools>
[user]: Could you please find out when Shishir's birthday is from our previous conversations and then congratulate him by sending a 'Happy Birthday!' message regardless?
</conversation_history>

<current_response_1>
<tool_call>
{"name": "recall_memory_search", "arguments": {"query": "Shishir Birthday"}}
</tool_call>
<tool_call>
{"name": "send_message", "arguments": {"message": "Happy Birthday!", "recipient": "Shishir"}}
</tool_call>
</current_response_1>

<current_response_2>
<tool_call>
{"name": "recall_memory_search", "arguments": {"query": "Shishir's birthday", "page": 0, "request_heartbeat": false}}
</tool_call>
</current_response_2>

Response from ToolRM

<think>
Okay, let's tackle this evaluation …
Wait, the tool recall_memory_search will return the birthday date, but the assistant then sends the message regardless. So maybe the send_message is done without waiting for the tool response? But according to the process, the assistant does tool call -> wait for response -> then next step. So Response 1 has two tool calls: first recall_memory_search, then send_message. That's correct.
But the key is that the recall_memory_search must be done with the correct query. Since the user said "Shishir's birthday", Response 2's query is more precise. So Response 2 is superior …
Given that, I think Response 2 is better because the query is more precise. So the choice is 2.
</think>
<choice>
2
</choice>

Figure 17: Error *case 2* from TRBench$_{\text{BFCL}}$, featuring responses from ToolRM-Gen.