

Procedure-Aware Reinforcement Learning for Tool-Augmented Large Language Models

Qinglong Zheng* Jiajun Fan* Chaoran Cheng Ge Liu

University of Illinois Urbana-Champaign
{qzheng19, jiajunf3, chaoran7, geliu}@illinois.edu

Abstract

Training tool-augmented large language models has evolved from supervised fine-tuning (SFT) to reinforcement learning with verifiable rewards (RLVR). Yet existing methods rarely examine whether intermediate execution logic—specifically, the procedural fidelity of multi-turn tool invocations—requires explicit supervision. We hypothesize that enforcing sequential validity at the turn level is a prerequisite for robust multi-turn task completion. To this end, we introduce a Procedure-Aware Reinforcement Learning Framework that explicitly evaluates procedural validity through both invocation chain quality and final system state correctness. Using Group Relative Policy Optimization (GRPO) with KL regularization to fine-tune Qwen2.5-3B-Instruct, we demonstrate on API-Bank, Bamboogle, and BFCL-V3 that our approach consistently outperforms correctness-only baselines. Despite operating at a substantially smaller scale, our 3B model achieves average accuracy competitive with proprietary models such as GPT-5-mini, GPT-4.1-mini, Claude-Haiku-4.5, and Gemini-2.5-Flash-Lite, whose scores are obtained under BFCL’s official per-model evaluation protocols. Qualitative analysis reveals that optimizing tool sequencing yields emergent benefits, simultaneously enhancing multi-turn performance and error recovery from failed tool calls.

1 Introduction

Pioneering works such as ReAct (Yao et al., 2023) and DeepSeek-R1 (Guo et al., 2025) have laid the groundwork for deploying large language models (LLMs) as autonomous agents, while AutoTIR (Wei et al., 2025) formalized Tool-Integrated Reasoning (TIR) in mathematical reasoning tasks, enabling agents to engage in interactive reasoning and tool invocation within a single generation process. Subsequent research has primarily applied reinforcement learning to mathematical reasoning, coding tasks, and web search (Zhang et al., 2025; Zhou et al., 2024; Jin et al., 2025; Liu et al., 2023), where agents typically interact with a single tool such as a code sandbox, unit test, or search API. Yet even in these simple settings, multi-turn tool invocation poses persistent challenges: disordered execution sequences, parameter mismatches causing API failures, and reasoning-execution inconsistencies. These issues become substantially more severe in general tool-calling tasks, where agents must coordinate multiple domain-specific APIs within a unified environment to address real-world problems. The dominant training paradigm has evolved from supervised fine-tuning (SFT) on large-scale synthetic data (Liu et al., 2025; Chen et al., 2025; Lin et al., 2024; Yang et al., 2025) to Reinforcement Learning with Verifiable Reward (RLVR) (Qian et al., 2025; Singh et al., 2025; Zeng et al., 2025). However, current RLVR approaches lack a principled understanding of the challenges inherent to general tool-calling, leading to reward formulations that neglect procedural validity.

ToolRL (Qian et al., 2025) demonstrated that RLVR can surpass SFT by introducing fine-grained reward signals decomposed into tool name, parameter name, and parameter value

*Equal contribution.

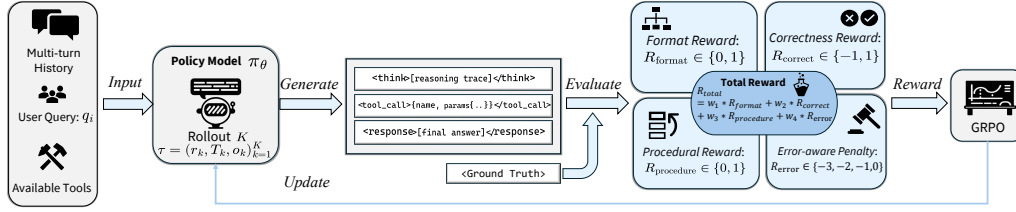


Figure 1: Overview of the Procedure-Aware Agentic RL Framework. In the figure, w_1-w_4 correspond to (w_f, w_c, w_p, w_e) in Eq. 4, and the reward ranges are abbreviated: R_{correct} and R_{proc} take continuous values in $[-1, 1]$ and $[0, 1]$, respectively (Sec. 3.2).

matching while ignoring tool **execution order**. Such designs fail to capture the procedural validity critical to state-aware environments, where the sequence of tool invocations determines the correctness of the resulting system status. We argue that mastering general tool-calling requires a fundamental shift from declarative learning, which fits a model to the distribution of “what” correct tool calls look like, to procedural learning that emphasizes “how” to achieve the goal in the correct sequence. Procedural learning cultivates capability through fine-grained and multi-dimensional reward signals that evaluate execution quality and step-wise correctness, as opposed to merely fitting the distribution of static expert trajectories. This intuition is supported by prior work in mathematical reasoning, where process supervision has been shown to significantly outperform outcome supervision (Lightman et al., 2023; Setlur et al., 2024). In realistic tool-calling scenarios, validity is far more complex. It hinges on discrete decision points: the decision to invoke a tool, the selection of a contextually reasonable tool, the efficient sequencing of tool invocations, the precise extraction of typed parameters and values from context, and the rigorous suppression of hallucinations. To our knowledge, no existing agentic RL method has systematically explored these specific dimensions.

In this work, we address these gaps by introducing a fundamental paradigm shift from outcome verification to rewarding the conscious procedure. Our framework leverages Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and systematically incentivizes four key pillars: (1) Decision Accuracy, ensuring the model correctly identifies when tool use is necessary; (2) Selection Logic, rewarding the choice of the most reasonable tool given the context; (3) Sequential Validity, strictly enforcing the correct ordering of tool invocations within a single turn; (4) Component Precision, validating the extraction of correct values and types. By jointly optimizing these complementary dimensions of tool-calling competence, we demonstrate that agents can achieve robust multi-turn task completion. In summary, our approach makes several key contributions:

1. We propose a Procedure-Aware Reward Framework that extends standard outcome-based reinforcement learning for multi-turn tool learning. By integrating diverse reward signals—Format Compliance, Correctness, Procedural Validity, and Error-Aware signal—into a Group Relative Policy Optimization (GRPO), we explicitly cultivate tool calling as a trainable, procedural skill rather than a latent emergent property.
2. We adopt Normalized Discounted Cumulative Gain as a fine-grained reward signal for evaluating the granularity of tool invocation sequence. Combined with state-aware verification, this metric allows us to reward the quality of partial ordering, enabling the model to distinguish between optimal, suboptimal, and invalid execution paths beyond simple binary correctness.
3. We demonstrate that enforcing procedural validity yields disproportionate performance gains. Our 3B models achieve average accuracy competitive with leading closed-source models, including GPT-5-mini, Claude-Haiku-4.5 and Gemini-2.5-Flash-Lite, under BFCL’s official per-model evaluation protocols.
4. We provide comprehensive quantitative and qualitative analysis through extensive ablation and case studies, confirming that each component of our compositional

reward structure contributes critically to final performance: all ablation deltas are statistically significant under paired bootstrap tests, with the procedural reward driving the largest multi-turn gains.

2 Related Work

Tool-Augmented LLMs Early approaches to tool-augmented Large Language Models (LLMs) primarily relied on SFT to align models with API specifications (Qu et al., 2025). However, SFT-based approaches (Patil et al., 2023; Schick et al., 2023; Qin et al., 2023; Liu et al., 2025; Yang et al., 2025) on large-scale trajectories establish a strong baseline but suffer from significant data labeling costs and often fail to generalize to unseen tools and environments. Recent work like ToolRL (Qian et al., 2025) has shifted toward RLVR, demonstrating that SFT imposes an upper bound on model performance and that RLVR can surpass SFT by optimizing directly for task success. However, our work identifies a critical limitation in their methods: their correctness evaluation employs greedy matching that finds optimal pairings between predicted and ground-truth tool calls while ignoring their execution order. Such designs treat tool usage as a “bag-of-actions”, rewarding the mere presence of correct tools while neglecting the intra-turn procedural fidelity—the strict sequential logic required to execute complex, interdependent tool calls.

Rule-based Outcome and Process Rewards in GRPO. Under the GRPO framework, rule-based rewards can be categorized into two levels based on evaluation granularity. **Outcome rewards** evaluate only the final result—whether the answer matches ground truth or code passes unit tests—providing a binary signal per episode. While simple to implement, this coarse feedback offers limited guidance for credit assignment across multi-step trajectories. In contrast, **process rewards** evaluate intermediate steps, such as the validity of each reasoning step or tool invocation. Traditional process reward models (PRMs) require expensive human annotations for step-level labels (Lightman et al., 2023; Setlur et al., 2024). Our work bridges this gap by designing rule-based process rewards that capture procedural validity without manual annotation, enabling scalable supervision for tool-calling tasks.

Procedural Learning in Sequential Decision Making. In broader reinforcement learning contexts, such as autonomous driving and robotics, rewarding the process—including safety constraints, smooth trajectories, and comfort metrics—is as critical as reaching the goal (Gu et al., 2024). Safe RL methods formalize this through Constrained MDPs, where agents must maximize cumulative rewards while satisfying auxiliary cost constraints throughout the learning and deployment processes (Wachi et al., 2024). Recent frameworks like OmniSafe (Ji et al., 2023) and Safety Gymnasium systematically evaluate how well algorithms balance reward optimization with constraint satisfaction across diverse robotic control tasks. Inspired by these principles, our work formalizes *sequential validity* as a measurable process metric for tool-calling tasks, ensuring that the agent internalizes the underlying logic of tool sequencing rather than merely achieving correct final outputs.

3 Methodology

3.1 Problem Formulation

TIR Trajectory. Following the formalization of Tool-Integrated Reasoning Wei et al. (2025); Qian et al. (2025), we define the tool-augmented language model task as a multi-step sequential decision making process. Figure 1 illustrates our overall framework. Given a user query q and a set of available tools $\mathcal{T} = \{t_1, t_2, \dots, t_N\}$, the model generates a trajectory τ consisting of interleaved reasoning steps and tool invocations:

$$\tau = ((r_1, \mathcal{T}_1, o_1), (r_2, \mathcal{T}_2, o_2), \dots, (r_K, \mathcal{T}_K, o_K)) \quad (1)$$

where at each step k : r_k denotes the natural language reasoning content, $\mathcal{T}_k \subseteq \mathcal{T}$ represents the set of tool calls invoked, and o_k captures the observation returned from tool execution.

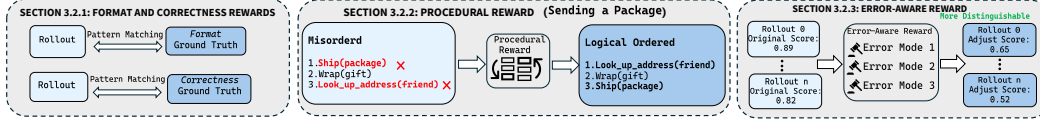


Figure 2: Reward Mechanism Overview. Each reward component targets a distinct dimension of tool-calling competence.

The model’s policy π maps from the current state s_k (comprising the query q , available tools $\mathcal{T}$, and trajectory history up to step k) to the next action $(r_{k+1}, \mathcal{T}_{k+1})$:

$$\pi : s_k \rightarrow (r_{k+1}, \mathcal{T}_{k+1}) \quad (2)$$

The optimization objective seeks to maximize the expected cumulative reward:

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{k=1}^K R(s_k, \mathcal{T}_{k+1}, o_{k+1}) \right] \quad (3)$$

Tool Calling Interaction. Modern LLMs support two paradigms for tool invocation: *native function calling* (FC) mode, where the model uses a built-in tool-calling interface managed by the serving framework, and *prompt-based* mode, where tool calls are emitted as structured text (e.g., XML or JSON) within the generation output. We adopt the prompt-based approach because XML-based templates allow the introduction of additional fields such as `<think>`, enabling explicit reasoning traces that are central to our focus on agentic behavior. In contrast, native FC mode offers limited customization over the generation format, which becomes a constraint when applying RL to improve reasoning depth or integrate fine-grained supervision signals. This prompt-based setting is also adopted by recent RL-based tool-learning works (Wang et al., 2025; Qian et al., 2025; Singh et al., 2025).

Structured Output Format. We enforce a structured XML-based output format to facilitate format reward computation. See App A.10 for the prompt template used.

3.2 Procedure-Aware Reward Decomposition

We decompose the total reward into four complementary components:

$$R_{\text{total}} = w_f \cdot R_{\text{format}} + w_c \cdot R_{\text{correct}} + w_p \cdot R_{\text{proc}} + w_e \cdot R_{\text{error}} \quad (4)$$

where the weights are empirically set to $(w_f, w_c, w_p, w_e) = (1, 3, 2, 1)$ and $R_{\text{total}} \in [-6, 6]$.

3.2.1 Verifiable Format and Correctness

Format Reward. The format reward $\mathcal{R}_{\text{format}} \in \{0, 1\}$ enforces adherence to the expected XML tag structure via pattern matching. Given ground truth g^* , we identify the expected pattern (*response-only*, *tool-only*, or *tool-then-response*) by tag presence:

$$\mathcal{R}_{\text{format}} = \begin{cases} 1, & \text{if all required fields appear in correct order} \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

We additionally enforce strict tag boundary checking to prevent reward hacking via content leakage between tags.

Correctness Reward: Hierarchical Semantic Evaluation. The correctness reward $\mathcal{R}_{\text{correct}} \in [-1, 1]$ provides a dense signal evaluating whether the model invokes the correct tools with correct parameters and values. Following prior work (Qian et al., 2025), we adopt a three-level hierarchical decomposition: tool name matching, parameter key matching, and parameter value matching. A frequency-aware matching function handles cases where tools are invoked multiple times, and a greedy matching strategy ensures each predicted tool matches at most one ground truth tool. See App. A.11 for the complete formulation.

3.2.2 Procedural Reward: Step-wise Execution Validity

The correctness reward evaluates *what* tools are called but ignores the *order* in which they are invoked (Figure 2). Reinforcing correct tool sets in invalid sequences leads to fragile agents that fail in state-sensitive environments. We denote the ground truth tool sequence as $G = [t_1^g, t_2^g, \dots, t_m^g]$ and the predicted sequence as $P = [t_1^p, t_2^p, \dots, t_n^p]$, excluding parameters to disentangle sequential validity from parameter correctness. The procedural reward $\mathcal{R}_{\text{proc}} \in [0, 1]$ decomposes into two components:

$$\mathcal{R}_{\text{proc}} = w_{\text{trace}} \cdot \mathcal{R}_{\text{trace}} + w_{\text{state}} \cdot \mathcal{R}_{\text{state}} \quad (6)$$

where $w_{\text{trace}} = w_{\text{state}} = 0.5$ (Singh et al., 2025). The trace reward measures positional ordering quality, while the state reward verifies final system state correctness.

Trace Reward via NDCG. The trace reward $\mathcal{R}_{\text{trace}} \in [0, 1]$ uses Normalized Discounted Cumulative Gain at position $p = |G|$. For each predicted tool at position i , we identify its closest match j^* in the ground truth and compute relevance as:

$$\text{rel}(i) = \frac{1}{1 + |i - j^*|} \quad (7)$$

The NDCG is then:

$$\mathcal{R}_{\text{trace}} = \frac{\text{DCG}@p}{\text{IDCG}@p}, \quad \text{DCG}@p = \sum_{i=1}^p \frac{2^{\text{rel}(i)} - 1}{\log_2(i+1)}, \quad \text{IDCG}@p = \sum_{i=1}^p \frac{1}{\log_2(i+1)} \quad (8)$$

State Reward via Suffix Matching. In multi-turn tasks, models may take equally valid but divergent trajectories—e.g., listing files before creating a directory—that deviate from the ground truth without being incorrect. They may also invoke redundant yet reasonable actions, or encounter execution errors that require recovery steps. Strict trajectory matching penalizes all such behaviors indiscriminately. The state reward addresses this by only checking whether the model ultimately reaches the correct final state, regardless of the path taken. Formally, $\mathcal{R}_{\text{state}} \in \{0, 1\}$ verifies that G appears as a suffix of P without earlier occurrence:

$$\mathcal{R}_{\text{state}} = \mathbb{I}[\text{suffix}(G, P)] \cdot \mathbb{I}[\text{unique}(G, P)] \quad (9)$$

This allows credit for exploratory paths that eventually converge to the correct execution sequence (Patil et al., 2025).

3.2.3 Error-aware Penalty

The error-aware signal penalizes structural invocation errors through three binary components ($\{-1, 0\}$ each), targeting distinct levels of the tool call hierarchy:

$$\mathcal{P}_{\text{func}} = \begin{cases} -1 & \mathcal{N}^p \neq \mathcal{N}^g \\ 0 & \text{otherwise} \end{cases}, \quad \mathcal{P}_{\text{param}} = \begin{cases} -1 & \mathcal{K}^p \neq \mathcal{K}^g \\ 0 & \text{otherwise} \end{cases}, \quad \mathcal{P}_{\text{type}} = \begin{cases} -1 & \text{type mismatch} \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

where $\mathcal{N}$, $\mathcal{K}$ denote tool name and parameter key sets (superscripts g/p for ground truth/prediction). The function-level penalty detects hallucinated or missing tools, the parameter-level penalty catches schema violations, and the type penalty identifies type mismatches (e.g., string vs. integer) and normalization inconsistencies. The total penalty aggregates all components:

$$\mathcal{R}_{\text{error}} = \mathcal{P}_{\text{func}} + \mathcal{P}_{\text{param}} + \mathcal{P}_{\text{type}} \quad (11)$$

See App. A.12 and App. A.13 for implementation details.

3.3 Optimization via Group Relative Policy Optimization

We employ Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to integrate our procedure-aware reward into a principled policy gradient framework. For each query Q ,

GRPO samples a group of n responses $\{r_1, \dots, r_n\}$ from the current policy and computes their rewards $\{R_1, \dots, R_n\}$ using Eq. 4. The advantage for each response is estimated via group-level normalization:

$$A_i = \frac{R_i - \mu_Q}{\sigma_Q + \epsilon}, \quad \mu_Q = \frac{1}{n} \sum_{j=1}^n R_j, \quad \sigma_Q = \sqrt{\frac{1}{n} \sum_{j=1}^n (R_j - \mu_Q)^2} \quad (12)$$

where ϵ is a small constant for numerical stability. The final GRPO objective combines the clipped surrogate loss with KL regularization:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{Q, \{r_i\}} \left[\frac{1}{n} \sum_{i=1}^n \left(\mathcal{L}_{\text{clip}}^{(i)}(\theta) - \beta \cdot D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) \right) \right] \quad (13)$$

where the clipped surrogate loss for each response is:

$$\mathcal{L}_{\text{clip}}^{(i)}(\theta) = \min \left(\rho_i(\theta) A_i, \text{clip} \left(\rho_i(\theta), 1 - \epsilon_{\text{low}}, 1 + \epsilon_{\text{high}} \right) A_i \right) \quad (14)$$

with importance ratio $\rho_i(\theta) = \frac{\pi_\theta(r_i|Q)}{\pi_{\text{old}}(r_i|Q)}$, and the KL divergence is computed as:

$$D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}}) = \mathbb{E}_{r \sim \pi_\theta} \left[\log \frac{\pi_\theta(r|Q)}{\pi_{\text{ref}}(r|Q)} \right] \quad (15)$$

We set $\beta = 0.001$ to ensure stable training dynamics throughout the optimization process.

4 Experiments

We evaluate our method across static single-turn, interactive multi-turn, and out-of-domain benchmarks to assess the effectiveness and generalizability of procedure-aware reward supervision.

Table 1: BFCL V3 Benchmark Results. **Blue** indicates best performance, **green** indicates second-best.

Model	Non-Live Acc	Live Acc	Multi-Turn Acc	Relevance Det.	Irrelevance Det.	Avg.Acc	Δ
Our Proposed Method							
Qwen2.5-3B-Instruct (Ours)	83.29 $\pm$ 0.22%	69.13 $\pm$ 0.34%	7.25$\pm$0.75%	88.24%	80.45%	65.67$\pm$0.17%	—
Qwen2.5-1.5B-Instruct (Ours)	76.48 $\pm$ 0.29%	63.27 $\pm$ 0.41%	3.50 $\pm$ 1.04%	83.33%	65.82%	58.48 $\pm$ 0.23%	—
RL Baseline Methods							
Qwen2.5-3B-Instruct (ToolRL)	79.52 $\pm$ 0.24%	67.88 $\pm$ 0.83%	3.88 $\pm$ 0.88%	88.24%	75.81%	63.07 $\pm$ 0.25%	+2.61 $\pm$ 0.30%
Qwen2.5-1.5B-Instruct (ToolRL)	78.40 $\pm$ 0.31%	67.06 $\pm$ 0.96%	1.25 $\pm$ 1.02%	100.00%	38.68%	57.08 $\pm$ 0.29%	+8.59 $\pm$ 0.33%
Base Models							
Qwen2.5-3B-Instruct (Raw)	75.52 $\pm$ 0.14%	62.18 $\pm$ 0.18%	1.00 $\pm$ 0.12%	94.12%	53.74%	57.31 $\pm$ 0.05%	+8.36 $\pm$ 0.18%
Qwen2.5-1.5B-Instruct (Raw)	70.43 $\pm$ 0.17%	55.61 $\pm$ 0.15%	0.50 $\pm$ 0.11%	82.35%	60.87%	53.95 $\pm$ 0.05%	+11.72 $\pm$ 0.18%
Larger-Scale Baselines							
GPT-4.1-mini (OpenAI)	84.60%	74.76%	2.50%	87.50%	73.88%	64.65%	+1.02 $\pm$ 0.17%
Gemma-3-12b-it (Google)	79.44%	74.24%	5.75%	93.75%	70.29%	64.69%	+0.98 $\pm$ 0.17%
Gemini-2.5-Flash-Lite (Google)	83.90%	54.85%	7.63%	50.00%	93.33%	57.94%	+7.73 $\pm$ 0.17%
GPT-5-mini (OpenAI)	68.04%	62.55%	5.50%	93.75%	55.71%	57.11%	+8.56 $\pm$ 0.17%
Claude-Haiku-4.5 (Anthropic)	55.42%	52.48%	1.75%	31.35%	95.29%	47.26%	+18.41 $\pm$ 0.17%

4.1 Experimental Setup

We adopt the 4k training data recipe¹, comprising multi-turn tool-calling samples where each sample contains system instructions specifying available tools, user queries, and ground-truth tool invocations. We experiment with two model scales from the Qwen2.5-Instruct-1.5B and Qwen2.5-Instruct-3B (Qwen Team, 2025), training 15 epochs with GRPO using 8 rollouts per prompt, batch size of 256. During training, rollout sampling uses temperature $\tau = 1.0$ to encourage exploration; at inference time, we use vLLM with temperature

¹<https://huggingface.co/datasets/emreacanacikgoz/ToolRL>

Table 2: API-Bank Benchmark Results. **Blue** indicates best, **green** indicates second-best. Accuracy (%) across three difficulty levels. Δ : Ours-3B Avg.Acc minus each model.

Model	Avg.Acc	Level 1	Level 2	Level 3	Δ
Our Proposed Method					
Qwen2.5-3B (Ours)	63.66%	74.19%	67.16%	49.62%	—
Qwen2.5-1.5B (Ours)	61.23%	73.68%	62.69%	47.33%	—
RL Baseline Methods					
Qwen2.5-3B (ToolRL)	57.92%	69.17%	65.67%	38.93%	+5.74%
Qwen2.5-1.5B (ToolRL)	55.18%	67.67%	59.70%	38.17%	+8.48%
Base Models					
Qwen2.5-3B (Raw)	27.28%	41.10%	20.90%	19.85%	+36.38%
Qwen2.5-1.5B (Raw)	32.73%	29.57%	37.31%	31.30%	+30.93%

Table 3: Bamboogle Benchmark Results. **Blue** indicates best, **green** indicates second-best. Multi-hop QA with compositional tool use. Std reflects search API variability across 3 runs.

Model	Accuracy	Avg Tool Calls	Δ
Our Proposed Method			
Qwen2.5-3B (Ours)	56.80$\pm$2.19%	1.38	—
Qwen2.5-1.5B (Ours)	41.60 $\pm$ 2.83%	1.21	—
RL Baseline Methods			
Qwen2.5-3B (ToolRL)	48.40 $\pm$ 2.45%	1.30	+8.40%
Qwen2.5-1.5B (ToolRL)	39.20 $\pm$ 2.67%	1.14	+17.60%
Base Models			
Qwen2.5-3B (Raw)	48.80$\pm$2.52%	1.71	+8.00%
Qwen2.5-1.5B (Raw)	21.60 $\pm$ 1.94%	0.58	+35.20%

$\tau = 0.01$ (near-greedy decoding) for deterministic evaluation. Training is conducted on $2 \times$ NVIDIA A6000 GPUs. We compare against two baseline categories: (1) Raw Qwen2.5-Instruct models without task-specific fine-tuning, and (2) ToolRL models using their released open-source weights^{2,3}, which were trained with format and correctness rewards. We fix the Qwen2.5 series across all systems to match ToolRL’s released checkpoints; holding the base model constant isolates the contribution of reward design from confounds introduced by newer base models. An additional BFCL-V3 comparison against a matched reproduction of ToolZero (Zeng et al., 2025) is provided in App. A.6. To contextualize our results against larger scale models, we additionally benchmark against GPT-5-mini, GPT-4.1-mini, Claude-Haiku-4.5, Gemini-2.5-Flash-Lite, and Gemma-3-12b-it. Evaluation spans three complementary benchmarks: BFCL V3 (Patil et al., 2025) for comprehensive function-calling assessment across synthetic, real-world, multi-turn scenarios; API-Bank for hierarchical difficulty evaluation; Bamboogle for multi-hop reasoning requiring search tool.

4.2 Main Results

4.2.1 BFCL V3 Benchmark

Table 1 presents results on BFCL V3, evaluating tool-calling across synthetic (Non-Live), real-world (Live), and multi-turn sequential interactions. Per-category scores are **AST summary** unweighted scores following BFCL’s official formula; Avg.Acc is the arithmetic mean of all five sub-category scores, and Δ denotes the Avg.Acc improvement of Ours-3B over each row. Scores for our models and ToolRL are averaged over multiple runs, while proprietary scores are taken from the official BFCL leaderboard. Our 3B model achieves the highest five-category average accuracy among all entries and clearly outperforms the correctness-only ToolRL baseline (+2.61); it also surpasses a reproduction of ToolZero trained under matched conditions (App. A.6). Against the larger-scale baselines—four proprietary models and the open-weight Gemma-3-12b-it, each evaluated by the BFCL maintainers under its own official protocol—our 3B model is competitive: the ranking is robust under the three-category aggregate that excludes Relevance/Irrelevance Detection, where it remains essentially tied with GPT-4.1-mini and ahead of every other entry in this group (App. A.5). The procedural reward drives the most significant gains in Multi-Turn accuracy, while the error-aware penalty improves parameter-level precision in Non-Live and Live categories.

4.2.2 API-Bank Benchmark

Table 2 presents results on API-Bank across three difficulty levels. Our method achieves the best overall accuracy, with particularly substantial gains on Level-3 (sequential tool coordination), validating that procedural reward translates into superior multi-step performance.

²ToolRL-Qwen2.5-1.5B: <https://huggingface.co/chengq9/ToolRL-Qwen2.5-1.5B>

³ToolRL-Qwen2.5-3B: <https://huggingface.co/chengq9/ToolRL-Qwen2.5-3B>

4.2.3 Bamboogle Benchmark

Table 3 presents results on Bamboogle, a multi-hop question answering benchmark requiring compositional reasoning across multiple tool invocations. Our 3B model achieves the highest accuracy with significant gains over the ToolRL baseline, notably on a benchmark outside our training distribution—demonstrating that procedural supervision cultivates generalizable reasoning rather than benchmark-specific optimizations.

4.3 Case Study: Qualitative Analysis

To verify improved procedural validity beyond accuracy metrics, we present a qualitative comparison between Ours-3B and ToolRL-3B on a representative BFCL-V3 multi-turn file system task (Table 4). Our model correctly identifies that navigation (`cd`) must precede file creation, while ToolRL attempts to create an already-existing directory, causing cascading state errors. This demonstrates that procedure-aware training cultivates robust **sequential validity** and **state tracking**, enabling the model to internalize API preconditions for robust error recovery. See App. A.14, App. A.15, and App. A.16 for additional case studies.

Table 4: Qualitative comparison on BFCL-V3 multi-turn file system task. Our method correctly identifies that navigation (`cd`) must precede file creation, while ToolRL incorrectly attempts to create a directory that already exists, causing cascading state errors.

Task: Navigate to Documents → Create `summary.txt` → Write “quantum computing” → Count words

Tools: `cd`, `mkdir`, `touch`, `echo`, `wc` **Challenge:** Correct sequencing of directory navigation before file creation

Ours-3B (SUCCESS)

Turn 0: Navigate & create file

```
<think> We need to first navigate to the 'Documents' directory and then create the file. </think>
<tool_call>
  cd(folder="Documents")
  touch(file_name="summary.txt")
</tool_call>
[OK] cwd: "Documents"
```

Turn 1: Write content

```
<tool_call>
  echo(content="quantum computing",
    file_name="summary.txt")
</tool_call>
[OK] Written successfully
```

Turn 2: Count words

```
<tool_call>
  wc(file_name="summary.txt", mode="w")
</tool_call>
[OK] {count: 2, type: "words"}
✓ Navigated first, then created file correctly
```

ToolRL-3B (FAILED)

Turn 0 - Step 0: Wrong assumption

```
<think> There is no 'Documents' dir, so we need to create one first. </think>
<tool_call>
  mkdir(dir_name="Documents")
  touch(file_name="summary.txt")
</tool_call>
[ERR] "Directory already exists"
→ File created in root, not Documents!
```

Turn 0 - Step 1: Failed recovery

```
<tool_call>
  touch(file_name="summary.txt")
</tool_call>
[ERR] "File already exists"
```

Turn 0 - Step 2: Gives up

```
<response> An error occurred because 'summary.txt' already exists. </response>
(Remaining turns fail due to state mismatch)
✗ Skipped cd, cascading failures
```

4.4 Ablation Study

Table 5 and Figure 3 present ablation results using the fine-tuned Qwen2.5-3B-Instruct model, evaluated at the same checkpoint. The full model achieves best performance across all categories: removing either component lowers Avg.Acc by 2.3–3.2 points, and every delta is 5–7× its propagated standard deviation and statistically significant under a paired bootstrap test on per-sample outcomes ($p < 10^{-7}$ throughout; Table 5). Removing the

Table 5: Ablation study on BFCL V3 benchmark. **Blue** indicates best performance, **green** indicates second-best. We evaluate the contribution of each proposed component by progressively removing it from the full framework. Per-category scores are BFCL AST summary scores as in Table 1; Avg.Acc is the unweighted mean of the three sub-category scores, i.e., (Non-live + Live + Multi-turn)/3. Δ : change in Avg.Acc relative to the Full Model. p : paired-bootstrap p-value on per-sample BFCL outcomes for the Full Model versus each variant; all deltas are statistically significant.

Method	Components		Performance (%)				Δ	p
	Procedural Reward	Error-aware Penalty	Non-live Acc	Live Acc	Multi-turn Acc	Avg.Acc		
Full Model	✓	✓	83.88 $\pm$ 0.22	70.62 $\pm$ 0.34	7.25 $\pm$ 0.75	53.92	—	—
w/o Penalty	✓	—	81.55 $\pm$ 0.23	67.69 $\pm$ 0.38	5.50 $\pm$ 0.46	51.58	-2.34	1.05×10^{-8}
w/o Procedural Reward	—	✓	82.27 $\pm$ 0.21	68.18 $\pm$ 0.35	4.50 $\pm$ 0.91	51.65	-2.27	2.46×10^{-8}
ToolRL-3B	—	—	80.45 $\pm$ 0.24	67.87 $\pm$ 0.83	3.88 $\pm$ 0.88	50.73	-3.19	1.32×10^{-10}

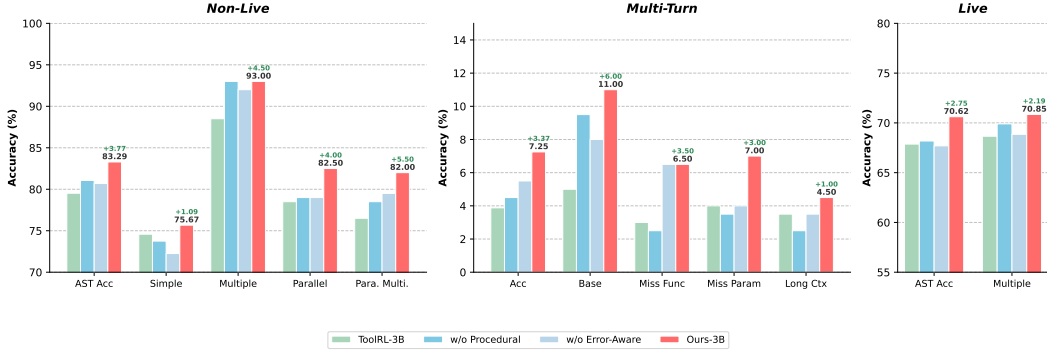


Figure 3: Task-wise ablation results on BFCL V3 benchmark. The full model (blue) consistently outperforms ablated variants across all sub-categories.

error-aware penalty degrades Non-Live and Live accuracy (more invocation errors), while removing the procedural reward causes the largest Multi-Turn drop—validating that explicit sequential supervision is critical. The Multi-Turn sub-category captures the procedural reward’s effect most directly: it supervises *intra-turn ordering* of tool invocations, and ablating it drops Multi-Turn by 2.75 absolute points (a 38% relative drop from the Full Model), whereas Non-Live and Live decline by only 1.9% and 3.5% relative. The aggregate Avg.Acc therefore understates the procedural reward’s contribution to its target capability. Neither component alone matches the full model: the procedural reward teaches *when* and *in what order* to invoke tools, while the error-aware signal teaches *how* to invoke them correctly—the two signals address complementary failure modes. We further validate the choice of NDCG as the trace metric through a controlled comparison against two procedural-reward variants built on Kendall’s τ and normalized edit distance (App. A.7): all three metrics successfully drive learning, but NDCG—the only position-sensitive one—is both the most principled choice and the empirical winner (+1.92 and +1.37 Avg.Acc over the two alternatives).

Figure 4 further illustrates these dynamics over the first 100 GRPO steps. Format compliance rises rapidly and saturates near 1.0 for all configurations. The procedural reward converges steadily for all configurations with procedural supervision, with 3B reaching ~ 0.88 and 1.5B reaching ~ 0.83 . The 3B Full Model achieves the highest correctness score, while the 1.5B model starts from a lower baseline with delayed onset (~ 20 steps). The error-aware penalty initially deepens during early exploration, then recovers as models learn to avoid parameter errors—with 3B recovering faster.

4.5 Error Analysis

To provide actionable insights for the community, we conduct a systematic error taxonomy over the failed multi-turn cases on BFCL V3, classifying 851 error instances in total (a single failed case may exhibit multiple error types). We emphasize that the absolute Multi-Turn

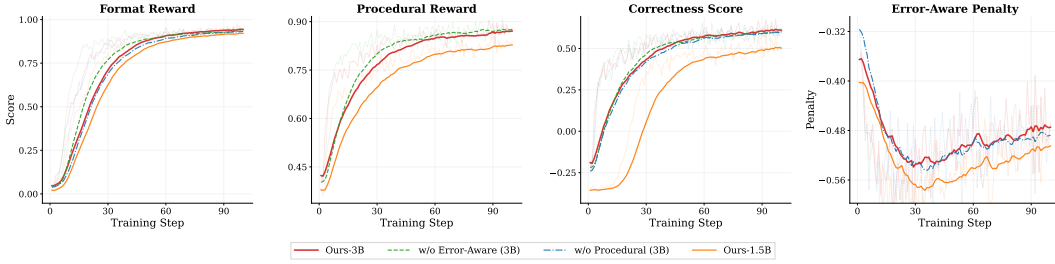


Figure 4: Training dynamics for ablation configurations (3B) and Ours-1.5B & 3B. From left to right: format reward, procedural reward, correctness score, and error-aware penalty (closer to zero is better).

accuracy remains low: this reflects a *systemic* challenge of BFCL-V3 multi-turn evaluation rather than a failure mode specific to our method—proprietary models also score low in this regime (GPT-4.1-mini: 2.50%, GPT-5-mini: 5.50%)—and within this difficult regime our procedural reward still delivers an 87% relative gain over ToolRL (3.88 $\rightarrow$ 7.25). A second notable pattern is the Base-vs.-Long-Context asymmetry: generation collapse under long context accounts for 26.4% of failures, with 29.5% of Long-Context cases force-terminated versus only 3.0% of Base cases. Because BFCL packs the full prior dialogue and all tool schemas into the context window, attention dilution is especially severe for a 3B model, making long-context *generation stability* a distinct bottleneck from procedural competence. These findings suggest that scaling multi-turn performance requires advances on multiple fronts, complementary to the reward-level improvements demonstrated in this work. See App. A.2 for the full analysis.

5 Conclusion

We introduced a Procedure-Aware RL Framework that shifts tool-calling supervision from outcome verification to explicit procedural validity. By decomposing rewards into format, correctness, procedure, and error-aware signals, our method transforms tool invocation from an emergent capability into a trainable skill. For sequential decision-making, the *how* of execution is as critical as the *what*. See App. A for extended discussions.

Ethics Statement

This paper presents work whose goal is to advance the field of Machine Learning. An LLM was used to polish the writing for ease of reading. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

- Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Yuefeng Huang, Wulong Liu, Xinzhi Wang, Defu Lian, Baoqun Yin, Yasheng Wang, and Wu Liu. Acebench: Who wins the match point in tool learning? *CoRR*, abs/2501.12851, 2025. doi: 10.48550/ARXIV.2501.12851. URL <https://doi.org/10.48550/arXiv.2501.12851>.
- Jiajun Fan, Roger Ren, Jingyuan Li, Rahul Pandey, Prashanth Gurunath Shivakumar, Ivan Bulyko, Ankur Gandhe, Ge Liu, and Yile Gu. Incentivizing consistent, effective and scalable reasoning capability in audio llms via reasoning process rewards, 2025.
- Shangding Gu, Long Yang, Yali Du, Guang Chen, Florian Walter, Jun Wang, and Alois Knoll. A review of safe reinforcement learning: Methods, theory and applications, 2024. URL <https://arxiv.org/abs/2205.10330>.

- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jia Shi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645 (8081):633–638, September 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL <http://dx.doi.org/10.1038/s41586-025-09422-z>.
- Jiaming Ji, Jiayi Zhou, Borong Zhang, Juntao Dai, Xuehai Pan, Ruiyang Sun, Weidong Huang, Yiran Geng, Mickel Liu, and Yaodong Yang. Omnisafe: An infrastructure for accelerating safe reinforcement learning research, 2023. URL <https://arxiv.org/abs/2305.09304>.
- Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025. doi: 10.48550/ARXIV.2503.09516. URL <https://doi.org/10.48550/arXiv.2503.09516>.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step, 2023. URL <https://arxiv.org/abs/2305.20050>.
- Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, Jun Wang, and Weinan Zhang. Hammer: Robust function-calling for on-device language models via function masking. *CoRR*, abs/2410.04587, 2024. doi: 10.48550/ARXIV.2410.04587. URL <https://doi.org/10.48550/arXiv.2410.04587>.
- Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. Rlrf: Reinforcement learning from unit test feedback, 2023. URL <https://arxiv.org/abs/2307.04349>.
- Weiwen Liu, Xu Huang, Xingshan Zeng, Xinlong Hao, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong Wang, Yuxian Wang, Wu Ning, Yutai Hou, Bin Wang, Chuhan Wu, Xinzhi Wang, Yong Liu, Yasheng Wang, Duyu Tang, Dandan Tu, Lifeng Shang, Xin Jiang, Ruiming Tang, Defu Lian, Qun Liu, and Enhong Chen.

- Toolace: Winning the points of LLM function calling. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=8EB8k6DdCU>.
- Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis, 2023. URL <https://arxiv.org/abs/2305.15334>.
- Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. The berkeley function calling leaderboard (BFCL): from tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*. OpenReview.net, 2025. URL <https://openreview.net/forum?id=2GmDdhBdDk>.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *CoRR*, abs/2504.13958, 2025. doi: 10.48550/ARXIV.2504.13958. URL <https://doi.org/10.48550/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. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL <https://arxiv.org/abs/2307.16789>.
- Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: a survey. *Frontiers Comput. Sci.*, 19(8):198343, 2025. doi: 10.1007/S11704-024-40678-2. URL <https://doi.org/10.1007/s11704-024-40678-2>.
- Qwen Team. Qwen2.5 technical report, 2025. URL <https://arxiv.org/abs/2412.15115>.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.
- Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for llm reasoning, 2024. URL <https://arxiv.org/abs/2410.08146>.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300, 2024. doi: 10.48550/ARXIV.2402.03300. URL <https://doi.org/10.48550/arXiv.2402.03300>.
- Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. *CoRR*, abs/2505.01441, 2025. doi: 10.48550/ARXIV.2505.01441. URL <https://doi.org/10.48550/arXiv.2505.01441>.
- Akifumi Wachi, Xun Shen, and Yanan Sui. A survey of constraint formulations in safe reinforcement learning, 2024. URL <https://arxiv.org/abs/2402.02025>.
- Hongru Wang, Cheng Qian, Wanjuan Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Acting less is reasoning more! teaching model to act efficiently, 2025. URL <https://arxiv.org/abs/2504.14870>.
- Yifan Wei, Xiaoyan Yu, Yixuan Weng, Tengfei Pan, Angsheng Li, and Li Du. Autotir: Autonomous tools integrated reasoning via reinforcement learning. *CoRR*, abs/2507.21836, 2025. doi: 10.48550/ARXIV.2507.21836. URL <https://doi.org/10.48550/arXiv.2507.21836>.

- Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. Toolmind technical report: A large-scale, reasoning-enhanced tool-use dataset. *CoRR*, abs/2511.15718, 2025. doi: 10.48550/ARXIV.2511.15718. URL <https://doi.org/10.48550/arXiv.2511.15718>.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.
- Yirong Zeng, Xiao Ding, Yutai Hou, Yuxian Wang, Li Du, Juyi Dai, Qiuyang Ding, Duyu Tang, Dandan Tu, Weiwen Liu, Bing Qin, and Ting Liu. Tool zero: Training tool-augmented llms via pure RL from scratch. *CoRR*, abs/2511.01934, 2025. doi: 10.48550/ARXIV.2511.01934. URL <https://doi.org/10.48550/arXiv.2511.01934>.
- Hanchen Zhang, Xiao Liu, Bowen Lv, Xueqiao Sun, Bohao Jing, Iat Long Long, Zhenyu Hou, Zehan Qi, Hanyu Lai, Yifan Xu, Rui Lu, Hongning Wang, Jie Tang, and Yuxiao Dong. Agentrl: Scaling agentic reinforcement learning with a multi-turn, multi-task framework. *CoRR*, abs/2510.04206, 2025. doi: 10.48550/ARXIV.2510.04206. URL <https://doi.org/10.48550/arXiv.2510.04206>.
- Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL <https://openreview.net/forum?id=b6rA0kAHT1>.

A Appendix

Table of Contents

- A.1 [Discussion](#) — Summary of key findings, emergent capabilities, limitations, and future work
- A.2 [Error Analysis: Multi-Turn Failure Modes](#)
- A.3 [Scaling to 7B](#)
- A.4 [SFT vs. RL on Identical Data](#)
- A.5 [Three-Category Aggregate Comparison with Larger-Scale Baselines](#)
- A.6 [ToolZero Reproduction Details](#)
- A.7 [Procedural Trace-Metric Variants](#)
- A.8 [Reward-Weight Rationale and Sensitivity](#)
- A.9 [Training-Compute Overhead of Procedure-Aware Rewards](#)
- A.10 [Output Format and Prompt Template](#)
- A.11 [Correctness Reward Details](#)
- A.12 [Why Correctness Alone Is Insufficient](#)
- A.13 [Error-Aware Penalty Details](#)
- A.14 [API-Bank Case Studies](#)
- A.15 [Bamboogle Case Studies](#)
- A.16 [BFCL-V3 Multi-Turn Case Studies with Tool Feedback](#)

A.1 Discussion

A.1.1 Summary of Key Findings

Our experimental results provide compelling evidence for the efficacy of procedure-aware reinforcement learning in tool-augmented language models. The proposed framework consistently outperforms matched RL baselines across three diverse benchmarks—BFCL-V3, API-Bank, and Bamboogle—while remaining competitive with substantially larger proprietary models. Several key findings merit detailed discussion.

Procedural Supervision Yields Disproportionate Multi-Turn Gains. The most striking result is the substantial improvement in multi-turn task completion. On BFCL-V3, our 3B model achieves 7.25% multi-turn accuracy compared to 3.88% for the ToolRL baseline—an 87% relative improvement. Similarly, on API-Bank Level-3 (the category directly testing sequential tool coordination), our method achieves 49.62% versus 38.93% for ToolRL, representing a 27% relative gain. These results validate our core hypothesis: explicitly rewarding procedural validity translates into superior multi-step performance that cannot be achieved through outcome-only supervision.

Complementary Interaction Between Reward Components. A critical finding from our ablation analysis is that neither the procedural reward nor the error-aware penalty alone matches the full model’s performance, indicating that these signals address complementary failure modes: the procedural reward teaches *when* and *in what order* to invoke tools, while the error-aware signal teaches *how* to invoke them correctly. The combined gain over ToolRL (+3.19) is smaller than the sum of the individual component contributions (+2.34 and +2.27), i.e., the two components are sub-additive and partially overlap in the failures they prevent; nonetheless, every ablation delta is large relative to run-to-run variability and statistically significant under paired bootstrap tests (Table 5).

Generalization Beyond Training Distribution. Our method demonstrates strong out-of-distribution generalization on Bamboogle, a multi-hop question answering benchmark outside our training distribution. The 8.4-percentage-point improvement over ToolRL (56.80% vs. 48.40%) suggests that procedural supervision cultivates transferable reasoning capabilities rather than benchmark-specific optimizations.

A.1.2 Emergent Capabilities from Procedural Learning

The qualitative case studies reveal emergent capabilities that naturally arise from procedural training:

Robust Error Recovery. In the BFCL-V3 error recovery case (App. A.16.1), our model demonstrates sophisticated diagnostic reasoning when encountering API failures. Upon receiving “distance not found in database,” our model reasons: “This could be due to the zipcodes not being recognized. I will need to find the zipcodes for both locations to proceed.” This ability to diagnose root causes and identify alternative solution paths is a direct consequence of training with sequential validity signals, which penalize premature task abandonment.

State Tracking Across Turns. The API-Bank continuation awareness case (App. A.14.4) demonstrates that our model maintains awareness of completed operations across dialogue turns. The thinking trace explicitly references prior results (“already retrieved likes for posts 13 and 14”), enabling incremental action generation rather than redundant computation. ToolRL, by contrast, treats each turn as a fresh request, resulting in duplicate API calls.

Grounded Tool Use Without Hallucination. In the BFCL-V3 API selection case (App. A.16.2), ToolRL generates fabricated tool responses (hallucinating “.hidden.file” entries that were never returned), while our model maintains strict fidelity to actual tool feedback. This grounding is reinforced by our error-aware penalties, which create negative gradients for responses inconsistent with expected API schemas.

A.1.3 Limitations

Despite the demonstrated effectiveness, our approach has several limitations that warrant acknowledgment:

Reliance on Ground Truth Tool Sequences. Our procedural reward computation requires access to ground truth tool invocation sequences during training. This limits applicability to scenarios where such annotations are available or can be synthesized. Extending the framework to settings with only outcome labels (e.g., task success/failure) would require developing techniques for inferring procedural validity from sparse feedback.

Limited Exploration of Model Scale. Our main experiments focus on 1.5B and 3B parameter models, and App. A.3 extends the comparison to 7B, where the procedural reward continues to improve Multi-Turn accuracy over ToolRL. Nevertheless, the interaction between model capacity and procedural learning at larger scales (13B and beyond) remains underexplored: larger models may exhibit different learning dynamics or require modified reward structures.

Evaluation Scope. Although we evaluate on three complementary benchmarks, the tool-calling domain encompasses diverse scenarios including code execution, database queries, and physical robot control. The transferability of our findings to these domains requires further investigation.

A.1.4 Future Work

Several promising directions emerge from our findings:

Automated Procedural Reward Discovery. Rather than manually designing reward components, future work could explore learning procedural validity signals from data. Techniques such as inverse reinforcement learning or reward model distillation could potentially extract implicit procedural knowledge from expert trajectories.

Hierarchical Procedural Learning. Complex agentic tasks often involve nested sub-goals requiring hierarchical planning. Extending our framework to model procedural validity at multiple abstraction levels—from individual API calls to high-level task decomposition—could improve performance on more sophisticated benchmarks.

Online Procedural Adaptation. Our current approach trains on a fixed dataset. Incorporating online learning mechanisms that continuously refine procedural understanding based on deployment feedback could enable adaptation to evolving API environments.

Cross-Domain Procedural Transfer. Investigating whether procedural skills learned in one tool domain (e.g., file system operations) transfer to others (e.g., web APIs) would illuminate the generality of procedural representations.

Integration with Process Reward Models. While our approach uses rule-based procedural rewards, combining these with learned PRMs could provide more nuanced feedback for ambiguous scenarios where multiple valid procedures exist.

A.2 Error Analysis: Multi-Turn Failure Modes

Despite achieving an 87% relative improvement in multi-turn accuracy over ToolRL (7.25% vs. 3.88%), the absolute score indicates that multi-turn tool-calling remains fundamentally challenging. We analyze all failed multi-turn cases on BFCL V3 across the four sub-categories (200 tasks each), classifying 851 error instances in total; a single failed case may exhibit multiple error types, so instance counts sum above the number of failed cases. Table 6 summarizes the error distribution.

Table 6: Multi-turn error distribution across BFCL V3 sub-categories. Errors are classified by the BFCL evaluation framework into five types. Counts are error *instances*: a failed case may contribute to multiple error types, so row sums can exceed the number of failed cases in a sub-category.

Sub-category	State Mismatch	Exec. Mismatch	Empty Response	Force Term.	Infer. Error	Acc.
Base	87	57	28	6	5	11.0%
Miss Parameter	87	48	37	14	10	7.0%
Miss Function	75	40	35	37	35	6.5%
Long Context	75	42	15	59	59	4.5%
Total	324	187	115	116	109	—

A.2.1 Training Data Distribution Gap (38.1%)

The largest class of errors (324 instances) arises from a distributional mismatch between training data and evaluation tasks. As shown in Table 7, our training set is heavily dominated by single-turn interactions (94.2%), with only 5.8% multi-turn samples—none of which involve the interactive file system operations that constitute the majority of BFCL multi-turn evaluations.

Table 7: Training data composition by dialogue turn count (total: 3,920 samples). BFCL multi-turn tasks require 3–5 interactive turns.

Turns	Count	Percentage
1 turn	3,691	94.2%
2 turns	93	2.4%
3 turns	110	2.8%
4–5 turns	26	0.6%

Notably, our model *does* succeed on multi-turn tasks when the evaluation domain aligns with the training distribution. For example, in `multi_turn_base_107`, the model correctly executes a 5-turn stock trading workflow—chaining `get_symbol_by_name` → `get_stock_info` → `place_order` → `get_order_details` → `cancel_order`—with proper error recovery when an initial API call fails. This contrast between success on in-distribution domains and failure on out-of-distribution file system tasks directly supports the training distribution gap hypothesis: the procedural reward effectively improves intra-turn execution ordering, but the base model lacks sufficient exposure to interactive file system trajectories for cross-turn generalization.

A.2.2 Incomplete Task Execution (22.0%)

In 187 cases, the model completes early turns correctly but fails to produce all required execution results in later turns. A representative example is `multi_turn_base_1`, where the model successfully navigates to the workspace directory and moves a file, but then inserts a redundant `pwd()` call. When subsequently attempting to read the moved file with `cat('log.txt')`, the file is no longer at the expected location, causing a “file does not exist”

error that prevents completion of later turns. These errors typically stem from redundant actions that consume interaction turns without advancing the task, causing the model to lose alignment with the expected execution trajectory.

A.2.3 Tool Schema Misperception (13.5%)

In 115 cases, the model generates zero function calls for an entire turn, despite having the required tools available. In `multi_turn_base_2`, the model’s reasoning trace states: *“there is no tool provided for file operations in the list,”* even though file system tools (`touch`, `cat`, `cd`, etc.) are present in the prompt. This misperception is especially prevalent in the Miss Function sub-category (35 out of 200 cases), where some tools are intentionally removed from the schema. The model appears to over-generalize from the absence of certain tools to conclude that *all* relevant tools are unavailable, leading to complete task abandonment.

A.2.4 Generation Collapse Under Long Context (26.4%)

A total of 225 cases exhibit catastrophic generation failure: 116 were force-terminated after producing degenerate output, and 109 resulted in unparseable responses. In `multi_turn_base_37`, the model generates single characters (`“E”`, `“r”`, `“r”`, `“o”`...) in an infinite loop, requiring force termination after 324 turns. This failure mode is strongly correlated with context length: in the Long Context sub-category, 59 out of 200 cases (29.5%) are force-terminated, compared to only 6 out of 200 (3%) in the Base sub-category. This indicates that the 3B model’s generation stability degrades substantially under extended multi-turn context pressure.

A.2.5 Impact of Task Complexity

Accuracy decreases monotonically with task complexity: Base (11.0%) > Miss Parameter (7.0%) > Miss Function (6.5%) > Long Context (4.5%). Each sub-category introduces an additional challenge beyond standard multi-turn interaction: Miss Parameter demands inference of missing argument values from context, Miss Function requires adaptation when expected tools are unavailable, and Long Context combines extended context pressure with all prior challenges. The Long Context sub-category exhibits disproportionate generation collapse (29.5% force-terminated), suggesting that context length is the primary bottleneck for small-scale models.

A.2.6 Implications

Most proprietary models also exhibit low multi-turn accuracy on BFCL V3 (GPT-4.1-mini: 2.50%, GPT-5-mini: 5.50%), confirming that multi-turn tool coordination is a systemic challenge. Our analysis identifies three actionable directions: (1) enriching training data with interactive multi-turn trajectories across diverse tool domains, particularly file system operations; (2) developing trajectory-level planning rewards that extend procedural supervision beyond the intra-turn level; and (3) incorporating context compression or retrieval-augmented techniques to maintain generation stability across long interactions.

A.3 Scaling to 7B

To examine whether the benefits of procedural supervision persist at larger model scale, we train Qwen2.5-7B-Instruct under our procedure-aware RL framework with the same training recipe as the main 3B run, and compare against the released ToolRL-Qwen2.5-7B checkpoint. Both models are evaluated on BFCL-V3 under an identical configuration (`max_model_len` = 16,384, `temperature` = 0.001); for each model, we run three independent BFCL generations and report the mean $\pm$ std.

Table 8: 7B-scale comparison on BFCL-V3. Scores are mean $\pm$ std over three independent generation runs. Δ : Ours-7B minus ToolRL-7B on the run means.

Sub-category	Ours-7B	ToolRL-7B	Δ
Multi-Turn	17.29$\pm$0.14	15.13 $\pm$ 0.57	+2.17
Non-Live AST	86.68 $\pm$ 0.24	86.71 $\pm$ 0.08	-0.03
Live AST	75.91$\pm$0.32	74.96 $\pm$ 0.40	+0.95
Relevance Detection	82.35	82.35	0
Irrelevance Detection	79.29$\pm$0.30	76.29 $\pm$ 0.17	+3.00
Avg.Acc	68.30$\pm$0.07	67.09 $\pm$ 0.18	+1.21

Across three runs, our 7B model attains a mean Multi-Turn accuracy of 17.29 ± 0.14 , compared with 15.13 ± 0.57 for ToolRL-7B—a +2.17 absolute ($\approx 14\%$ relative) improvement—alongside gains on Live AST (+0.95) and Irrelevance Detection (+3.00), yielding a +1.21 improvement in five-category Avg.Acc. These results indicate that the procedural reward is not merely compensating for weak base capability at small scale: explicit sequential supervision continues to improve multi-turn coordination as model capacity grows.

A.4 SFT vs. RL on Identical Data

The procedural reward requires gold tool sequences, which raises the question of whether our advantage over SFT stems from the RL formulation itself or merely from richer utilization of the same ground-truth annotations. To address this, we conduct a head-to-head SFT-vs.-RL comparison on the same 4k annotated data. We independently reproduce SFT on Qwen2.5-3B-Instruct using a learning-rate sweep over $\{5 \times 10^{-6}, 2 \times 10^{-5}, 5 \times 10^{-5}\}$ with full BFCL evaluation (mean $\pm$ std over 3 rounds); we report the best run, obtained at $\text{LR} = 5 \times 10^{-6}$.

Table 9: SFT vs. correctness-based RL vs. procedure-aware RL on identical 4k training data (Qwen2.5-3B-Instruct). Scores are mean $\pm$ std over three BFCL evaluation rounds.

Setup	Non-Live AST	Live	Multi-Turn	Relevance	Irrelevance	Avg.Acc
SFT	49.09 $\pm$ 0.19	61.97 $\pm$ 0.11	1.92 $\pm$ 0.07	82.35	81.23	55.31 $\pm$ 0.06
Correctness-based RL — ToolRL	79.52 $\pm$ 0.24	67.88 $\pm$ 0.83	3.88 $\pm$ 0.88	88.24	75.81	63.07 $\pm$ 0.25
Procedure-aware RL — Ours	83.29$\pm$0.22	69.13$\pm$0.34	7.25$\pm$0.75	88.24	80.45	65.67$\pm$0.17

The results show two cumulative effects. First, RL substantially improves over SFT on the same 4k data: on Multi-Turn, reproduced SFT achieves 1.92 whereas the ToolRL baseline achieves 3.88, a +1.96 absolute gain. Second, procedure-aware RL further raises Multi-Turn from 3.88 to 7.25 over correctness-based RL. Cumulatively, our procedure-aware RL substantially exceeds SFT’s Multi-Turn performance on the same data (1.92 $\rightarrow$ 7.25). This reflects the transition from **declarative learning to procedural learning** advocated in the introduction: SFT primarily fits “what” correct tool calls look like through imitation on gold sequences, whereas procedure-aware RL extracts “how” to achieve the goal in the correct sequence through trace and state rewards. Both our method and ToolRL access the identical annotations; the gain therefore isolates the effect of reward design while holding annotation access fixed. Viewed as supervision-extraction efficiency, procedure-aware RL converts each unit of multi-turn supervision into $\approx 1.87\times$ the Multi-Turn capability extracted by ToolRL and $\approx 3.77\times$ that of SFT; complementarily, the training-data distribution gap of App. A.2 characterizes the ceiling—failures whose test patterns are genuinely absent from the supervision distribution cannot be fabricated by any reward design.

A.5 Three-Category Aggregate Comparison with Larger-Scale Baselines

The five-category Avg.Acc in Table 1 includes Relevance and Irrelevance Detection, where some proprietary baselines score abnormally low under their official protocols. To verify that our ranking does not depend on absorbing these low detection sub-scores into the mean, we recompute the aggregate using only the three AST sub-categories (Non-Live, Live, Multi-Turn), dropping Relevance and Irrelevance entirely.

Table 10: Three-category aggregate (no Relevance/Irrelevance) on BFCL-V3. Baseline scores are from the official BFCL-V3 leaderboard; the Ours-3B per-category values follow the same-checkpoint protocol of Table 5. Δ : each model’s three-category aggregate minus Ours-3B.

Model	Non-Live	Live	Multi-Turn	3-cat (no R/IR)	Δ vs Ours-3B
Ours-3B	83.88	70.62	7.25	53.92	—
GPT-4.1-mini	84.60	74.76	2.50	53.95	+0.03
Gemma-3-12b-it	79.44	74.24	5.75	53.14	−0.78
Gemini-2.5-Flash-Lite	83.90	54.85	7.63	48.79	−5.13
GPT-5-mini	68.04	62.55	5.50	45.36	−8.56
Claude-Haiku-4.5	55.42	52.48	1.75	36.55	−17.37

Excluding Relevance/Irrelevance, Ours-3B is essentially tied with GPT-4.1-mini ($\Delta = +0.03$) and remains ahead of every other entry in this group by 0.78 to 17.37 points. Concretely, dropping R/IR changes the conclusion in only one of the five comparisons (GPT-4.1-mini: from a small lead to a statistical tie) and strengthens or leaves unchanged the conclusion in the other four. If the ranking in Table 1 were an artifact of R/IR absorption, removing R/IR would collapse or reverse our lead across most baselines; the actual numbers show the opposite. This robustness check motivates the “competitive with proprietary models” framing used throughout the paper.

A.6 ToolZero Reproduction Details

ToolZero (Zeng et al., 2025) has not released its code, so we re-implement its GG-GRPO progressive reward within our framework:

$$R_{\text{final}} = R_{\text{format}} + \sigma(t, m) \cdot r_{\text{strict}} + (1 - \sigma(t, m)) \cdot r_{\text{general}} \quad (16)$$

where $\sigma(t, m)$ is the progressive schedule with steepness $\kappa = 0.1$ and midpoint $m = 25$; r_{general} is a token-overlap soft reward in $[-0.5, +0.5]$; and $r_{\text{strict}} = r_{\text{ast}} + r_{\text{multi-tool}} + r_{\text{value-error}}$, following their App. A.1. We implement this reward within our framework and retrain Qwen2.5-3B-Instruct on the identical 4k data using the same hyperparameters as our main 3B run. The selected checkpoint is evaluated over three independent BFCL runs and aggregated following the Table 1 protocol.

Table 11: ToolZero reproduction vs. Ours-3B on BFCL-V3. Both models are trained on the identical 4k data with the same hyperparameters; scores are averaged over three independent BFCL runs following the Table 1 protocol.

Model	Non-Live	Live	Multi-Turn	Relevance	Irrelevance	Avg.Acc
Ours-3B	83.29±0.22	69.13±0.34	7.25±0.75	88.24	80.45	65.67±0.17
ToolZero-3B	60.54±0.42	65.51±0.16	1.96±0.39	76.47	77.93	56.48±0.12
Δ (Ours – ToolZero)	+22.75	+3.62	+5.29	+11.77	+2.52	+9.19

As shown in Table 11, the best ToolZero-3B reproduction reaches 56.48 ± 0.12 Avg.Acc, compared with 65.67 ± 0.17 for Ours-3B ($\Delta = +9.19$), with the gap concentrating on the call-coordination sub-categories (Non-Live +22.75, Multi-Turn +5.29). This confirms that the procedural-reward contribution holds against a second contemporary RL-based tool-learning method, not just ToolRL.

A.7 Procedural Trace-Metric Variants

The NDCG-based trace reward (Eq. 8) is one of several plausible ordering metrics. To validate this design choice, we train two procedural-reward variants that replace NDCG with alternative trace metrics, keeping the training recipe otherwise identical: (i) **Kendall’s τ (subset $\times$ coverage)**, which greedily matches predicted tools to ground-truth tools by ID, computes τ_b over the matched pairs, rescales $[-1, 1]$ to $[0, 1]$, and multiplies by a coverage term $\text{matched} / \max(|P|, |G|)$ —position-agnostic among matched pairs; and (ii) **normalized edit distance**, $1 - \text{Lev}(P, G) / \max(|P|, |G|)$ over tool-ID sequences, which assigns a uniform cost to any out-of-order placement. NDCG is the only one of the three that weights early-position errors more heavily, which is precisely the property our procedural-validity hypothesis requires.

Table 12: Procedural trace-metric variants on BFCL V3. All rows use the identical training recipe and evaluation protocol as the Full Model; the only difference is the trace metric inside the procedural reward. Scores are 3-round mean±std. Δ : Avg.Acc relative to NDCG.

Trace metric	Non-Live Acc	Live Acc	Multi-Turn Acc	Relevance Det.	Irrelevance Det.	Avg.Acc	Δ
NDCG (Full Model)	83.29±0.22	69.13±0.34	7.25±0.75	88.24	80.45	65.67±0.17	—
Kendall’s τ	81.47±0.13	68.89±0.16	4.71±0.40	88.24	75.45	63.75±0.23	−1.92
Normalized edit distance	81.24±0.18	68.42±0.48	5.75±0.45	88.24	77.86	64.30±0.37	−1.37

Table 12 shows that NDCG leads on Avg.Acc with margins that are large relative to run-to-run variability: Kendall’s τ is lower by 1.92 ± 0.21 and normalized edit distance by 1.37 ± 0.35 (independent-error propagation). The gap is dominated by Irrelevance Detection and Multi-Turn rather than by Live or Non-Live AST, indicating that position-sensitive credit for early ordering errors matters most exactly where sequential coordination is stressed. Notably, all three metrics successfully drive learning (63.8–64.3 Avg.Acc, within approximately two points of NDCG), consistent with our framing that *procedural validity as a reward signal* is the main contribution, while the specific scoring function is an important design choice;

NDCG is both the most principled—position-weighted, graded, and normalized—and the empirical winner in this controlled comparison.

A.8 Reward-Weight Rationale and Sensitivity

The weights $(w_f, w_c, w_p, w_e) = (1, 3, 2, 1)$ in Eq. 4 are fixed by two principles in sequence.

$(w_f, w_c) = (1, 3)$ **is inherited unchanged from ToolRL.** This is intentional: keeping the format-and-correctness components identical to our primary baseline ToolRL (Qian et al., 2025) ensures that the observed gain is attributable to the *added* procedural and error-aware rewards, rather than to re-tuning of the shared reward components. Giving correctness the largest weight is also corroborated by concurrent reward-design work beyond tool use: CESAR (Fan et al., 2025) reports that, after an empirical sweep, weighting the correctness/accuracy reward higher than process-oriented rewards ($\alpha_{\text{correctness}} = 5.0$ vs. $\alpha_{\text{process}} = 1.0$ in their notation) “yielded the best results,” suggesting that correctness-dominant weighting is a principle shared by independently developed RL reward frameworks.

$(w_p, w_e) = (2, 1)$ **is then fixed by a reward-symmetry requirement.** With per-component bounds $\text{format} \in [0, 1]$, $\text{correctness} \in [-1, 1]$, $\text{procedural} \in [0, 1]$, and $\text{error-aware} \in [-3, 0]$, the maximum positive contribution is $1 \cdot 1 + 3 \cdot 1 + 2 \cdot 1 + 1 \cdot 0 = +6$, while the minimum negative contribution is $1 \cdot 0 + 3 \cdot (-1) + 2 \cdot 0 + 1 \cdot (-3) = -6$. The aggregated weighted reward therefore lies in the symmetric interval $[-6, +6]$, avoiding reward-scale drift across rollouts. Within the procedural reward, $w_{\text{trace}} = w_{\text{state}} = 0.5$ splits the $w_p = 2$ budget equally between stateless trace ordering and stateful execution progress.

Sensitivity. The strongest perturbation of a reward weight is to set it to zero, which is precisely what the ablation study tests: the “w/o Procedural Reward” variant zeroes the procedural term, while the “w/o Penalty” variant zeroes the error-aware term. The training dynamics in Figure 4 and the ablation results in Table 5 show that both ablated variants train stably to convergence; the difference lies in downstream BFCL-V3 accuracy, not optimization stability. Since re-tuning weights within their positive range is a weaker perturbation than completely removing a component, the ablation results provide a conservative robustness check for the weighting design.

A.9 Training-Compute Overhead of Procedure-Aware Rewards

The procedural and error-aware rewards are rule-based (NDCG@p over short tool lists, suffix matching for terminal-state verification, and type/set comparisons for the structural penalty) and are dominated entirely by GRPO’s rollout and FSDP backward cost. We quantify the overhead with a microbenchmark over 50 real training samples on a single CPU thread.

Table 13: Wall-clock cost of reward computation. Per-GRPO-step cost assumes batch 256×8 rollouts = 2048 reward calls.

Configuration	Per-sample reward wall-clock	Per-GRPO-step
Procedure-aware (Ours): Format + Correct + Procedure + Error-aware	0.218±0.082 ms	0.45 s
ToolRL-equivalent stack: Format + Correct only	0.093±0.029 ms	0.19 s
Overhead	+0.124 ms	+0.25 s

The procedural + error-aware reward computation contributes 0.25 s out of a 356 s training step, i.e., $\approx 0.07\%$ of training wall-clock—three orders of magnitude below the dominant training phases. Adopting a procedure-aware reward over a ToolRL-style reward therefore costs essentially nothing in compute.

A.10 Output-format and Prompt

System Prompt for Training

You are a helpful dialogue assistant capable of leveraging tool calls to solve user tasks and provide structured chat responses.

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

Steps for Each Turn

1. **Think:** Recall relevant context and analyze the current user goal.
2. **Decide on Tool Usage:** If a tool is needed, specify the tool and its parameters.
3. **Respond Appropriately:** If a response is needed, generate one while maintaining consistency across user queries.

Output Format

```

<think> Your thoughts and reasoning </think>
<tool.call>
{"name": "Tool name", "parameters": {"Parameter name": "Parameter content", "... ...": "... ..."}}
{"name": "... ...", "parameters": {"... ...": "... ...", "... ...": "... ..."}}
...
</tool.call>
<response> AI's final response </response>

```

Important Notes

1. You must always include the `<think>` field to outline your reasoning. Provide at least one of `<tool.call>` or `<response>`. Decide whether to use `<tool.call>` (possibly multiple times), `<response>`, or both.
2. You can invoke multiple tool calls simultaneously in the `<tool.call>` fields. Each tool call should be a JSON object with a "name" field and a "parameters" field containing a dictionary of parameters. If no parameters are needed, leave the "parameters" field an empty dictionary.
3. Refer to the previous dialogue records in the history, including the user's queries, previous `<tool.call>`, `<response>`, and any tool feedback noted as `<obs>` (if exists).

Figure 5: System prompt template used during training.

A.11 Correctness reward details

We adopt the correctness reward from ToolRL; see the original paper (Qian et al., 2025) for further details. We denote a tool call as $t = (n, \theta)$, where n is the tool name and $\theta : \mathcal{K} \rightarrow \mathcal{V}$ is a parameter dictionary mapping parameter names (keys) to their values. Given ground truth tool calls $\mathcal{G} = \{g_1, g_2, \dots, g_m\}$ and predicted tool calls $\mathcal{P} = \{p_1, p_2, \dots, p_n\}$, where each $g_i = (n_i^g, \theta_i^g)$ and $p_j = (n_j^p, \theta_j^p)$, we compute the correctness reward as follows.

We first define a frequency-aware matching function. For two lists L_1 and L_2 , the match score is computed as:

$$\text{match}(L_1, L_2) = \frac{\sum_{k \in L_1 \cap L_2} \min(\text{count}(k, L_1), \text{count}(k, L_2))}{|L_1| + |L_2| - |L_1 \cap L_2|} \quad (17)$$

where $\text{count}(k, L)$ denotes the frequency of element k in list L . This score leverages both ground truth and prediction in the denominator, providing a symmetric similarity measure that accounts for element frequencies.

Three-Level Hierarchical Decomposition. The correctness score is decomposed into three granular levels:

Level 1: Tool Name Matching. We first evaluate whether the model selects the correct tools by name. Let $\mathcal{N}^g = [n_1^g, n_2^g, \dots, n_m^g]$ denote the list of ground truth tool names and $\mathcal{N}^p = [n_1^p, n_2^p, \dots, n_n^p]$ denote the predicted tool names. The tool name matching score is:

$$S_{\text{name}} = \text{match}(\mathcal{N}^g, \mathcal{N}^p) \quad (18)$$

Level 2: Parameter Key Matching. For each ground truth tool g_i that has a matching predicted tool p_j (where $n_i^g = n_j^p$), we evaluate whether the model specifies the correct parameter schema. Let $\text{keys}(\theta)$ denote the set of parameter names in dictionary θ . The parameter key matching score for tool pair (g_i, p_j) is:

$$S_{\text{keys}}^{(i,j)} = \text{match}(\text{keys}(\theta_i^g), \text{keys}(\theta_j^p)) \quad (19)$$

Level 3: Parameter Value Matching. Finally, we evaluate whether the parameter values are exactly correct. For each parameter key k in the ground truth parameters θ_i^g , we use exact equality with no normalization. The parameter value matching score for tool pair (g_i, p_j) is:

$$S_{\text{values}}^{(i,j)} = \sum_{k \in \text{keys}(\theta_i^g)} \mathbb{I}[\theta_j^p[k] = \theta_i^g[k]] \quad (20)$$

where $\mathbb{I}[\cdot]$ is the indicator function.

Final Correctness Reward. The total correctness score aggregates all three levels:

$$S = S_{\text{name}} + \sum_{i=1}^m \left(S_{\text{keys}}^{(i)} + S_{\text{values}}^{(i)} \right) \quad (21)$$

where $S_{\text{keys}}^{(i)}$ and $S_{\text{values}}^{(i)}$ denote the parameter key and value scores for the best matching predicted tool for ground truth tool g_i .

The maximum achievable score is:

$$S_{\text{max}} = 1 + \sum_{i=1}^m \left(1 + |\text{keys}(\theta_i^g)| \right) \quad (22)$$

where the initial 1 represents perfect tool name matching, and for each ground truth tool, 1 represents perfect parameter key matching and $|\text{keys}(\theta_i^g)|$ represents all parameter values being correct.

The correctness reward is then normalized to the range $[-1, 1]$ via linear scaling:

$$\mathcal{R}_{\text{correct}} = 2 \cdot \frac{S}{S_{\text{max}}} - 1 \quad (23)$$

A.12 Why Correctness Alone Is Insufficient.

The correctness reward $\mathcal{R}_{\text{correct}} \in [-1, 1]$, while comprehensive in evaluating semantic accuracy through its three-level decomposition, exhibits two fundamental limitations when addressing structural violations. The match score mechanism provides diluted gradient signals in scenarios with multiple tool invocations or tools with numerous parameters. Consider a single-turn response invoking three tools, where one tool contains a typo in a parameter name (e.g., `queryry` instead of `query`). The correctness reward computes an aggregate score across all three tools and all their parameters, effectively averaging this critical structural error with potentially many correct components. The resulting gradient signal becomes diffuse, making it difficult for the model to identify and correct the specific violation that caused API failure. In production environments where a single malformed parameter can abort an entire transaction chain, this dilution of error signals is particularly problematic.

A.13 Error-Aware Penalty Details

This section provides the complete specification of the error-sensitive signal’s three penalty components and the normalization function.

Notation. We first establish the notation used throughout this section. Let superscripts g and p denote ground truth and prediction, respectively. We use $\mathcal{N}$ for sets of tool names and θ for parameter mappings from keys to values, with $\mathcal{K} = \text{keys}(\theta)$ denoting the corresponding key sets. For a parameter key k , we write $\theta[k]$ to access its value, and $|\theta[k]|$ to denote its cardinality (i.e., the number of elements when the value is a list). The function $\text{type}(\cdot)$ returns the data type of a value, and $\text{islist}(\cdot)$ is a predicate indicating whether a value is a list. We denote by $\mathcal{S}$ the set of all strings, $\mathcal{S}_{\text{json}} \subset \mathcal{S}$ the subset of JSON-parsable strings (those beginning with `[` or `{`), $\mathcal{L}$ the set of lists, and $\mathcal{D}$ the set of dictionaries. The auxiliary functions include $\text{lower}(\cdot)$ for lowercase conversion, $\text{rmws}(\cdot)$ for whitespace removal, and $\text{parse}(\cdot)$ for JSON parsing. We employ boolean predicates C with descriptive subscripts to indicate specific error conditions: C_{extra} for **extraneous** parameters, C_{miss} for **missing** parameters, C_{card} for **cardinality** (list length) mismatches, C_{type} for **type** mismatches, and C_{norm} for **normalization**-detectable inconsistencies.

Function-Level Penalty. The function-level penalty detects mismatches between the sets of invoked tools. Let $\mathcal{N}^g$ and $\mathcal{N}^p$ denote the tool name sets in ground truth and prediction, respectively. The penalty is defined as:

$$\mathcal{P}_{\text{func}} = \begin{cases} -1 & \text{if } (\mathcal{N}^p \setminus \mathcal{N}^g \neq \emptyset) \vee (\mathcal{N}^g \setminus \mathcal{N}^p \neq \emptyset) \\ 0 & \text{otherwise} \end{cases} \quad (24)$$

This penalty activates under two conditions: (1) $\mathcal{N}^p \setminus \mathcal{N}^g \neq \emptyset$, indicating the model invoked extraneous tools absent from the ground truth; or (2) $\mathcal{N}^g \setminus \mathcal{N}^p \neq \emptyset$, indicating the model omitted required tools.

Parameter-Level Penalty. For each matched tool pair (g, p) where tool names coincide, let θ^g and θ^p denote the respective parameter mappings, with $\mathcal{K}^g = \text{keys}(\theta^g)$ and $\mathcal{K}^p = \text{keys}(\theta^p)$. The penalty is:

$$\mathcal{P}_{\text{param}} = \begin{cases} -1 & \text{if } \exists (g, p) : C_{\text{extra}} \vee C_{\text{miss}} \vee C_{\text{card}} \\ 0 & \text{otherwise} \end{cases} \quad (25)$$

where the error conditions are defined as:

$$C_{\text{extra}} \triangleq \mathcal{K}^p \setminus \mathcal{K}^g \neq \emptyset \quad (26)$$

$$C_{\text{miss}} \triangleq \mathcal{K}^g \setminus \mathcal{K}^p \neq \emptyset \quad (27)$$

$$C_{\text{card}} \triangleq \exists k \in \mathcal{K}^g \cap \mathcal{K}^p : \text{islist}(\theta^g[k]) \wedge \text{islist}(\theta^p[k]) \wedge |\theta^g[k]| \neq |\theta^p[k]| \quad (28)$$

These conditions capture three distinct error types: (1) extraneous parameters not specified in the schema, (2) missing required parameters, and (3) cardinality mismatches for list-typed parameters.

Type Error Penalty. For parameters with matching keys $k \in \mathcal{K}^g \cap \mathcal{K}^p$, the type error penalty is:

$$\mathcal{P}_{\text{type}} = \begin{cases} -1 & \text{if } \exists k : C_{\text{type}}(k) \vee C_{\text{norm}}(k) \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

where:

$$C_{\text{type}}(k) \triangleq \text{type}(\theta^p[k]) \neq \text{type}(\theta^g[k]) \quad (30)$$

$$C_{\text{norm}}(k) \triangleq \text{type}(\theta^p[k]) = \text{type}(\theta^g[k]) \wedge \theta^p[k] \neq \theta^g[k] \wedge \text{norm}(\theta^p[k]) = \text{norm}(\theta^g[k]) \quad (31)$$

The predicate C_{type} captures explicit type mismatches (e.g., string versus integer). The predicate C_{norm} identifies values that differ in surface form but are semantically equivalent after normalization. Notably, type checking takes precedence: C_{norm} is evaluated only when types match.

Normalization Function. The function $\text{norm}(\cdot)$ recursively processes values to enable semantic comparison:

$$\text{norm}(v) = \begin{cases} \text{norm}(\text{parse}(v)) & \text{if } v \in \mathcal{S}_{\text{json}} \\ \text{lower}(\text{rmws}(v)) & \text{if } v \in \mathcal{S} \setminus \mathcal{S}_{\text{json}} \\ [\text{norm}(x)]_{x \in v} & \text{if } v \in \mathcal{L} \\ \{\text{lower}(k) : \text{norm}(u)\}_{(k,u) \in v} & \text{if } v \in \mathcal{D} \\ \text{str}(v) & \text{if } v \in \mathbb{Z} \cup \mathbb{R} \end{cases} \quad (32)$$

This normalization accommodates cosmetic variations such as: "New York" versus "new york" (case and whitespace), "[1, 2]" versus [1, 2] (stringified versus parsed list), and {"Name": 1} versus {"name": 1} (dictionary key case).

Edge Cases When the ground truth contains no tool calls, the task does not require tool invocation and no penalty is applied. When the model's response cannot be parsed due to malformed output, no penalty is applied as such errors are captured by separate format validation rewards.

A.14 API-Bank Case Studies

The following cases from API-Bank demonstrate common failure modes in tool-calling scenarios, including parameter type compliance, complex formatting, missing parameters, and dialogue context awareness.

A.14.1 Parameter Type Compliance (String vs Float)

Task Description

Query: Get weather for coordinates (latitude=37.7749, longitude=122.4194)

Available Tools: GetWeatherForCoordinates

Ground Truth: GetWeatherForCoordinates(latitude="37.7749", longitude="122.4194")

Key Challenge: Parameter type must be string, not float

Ours-3B (SUCCESS)

Turn 0: Fetch weather data

```
<think> First, I need to fetch the
current weather information for the
given coordinates. </think>
<tool_call>
  GetWeatherForCoordinates(
    latitude="37.7749",
    longitude="122.4194")
</tool_call>
```

[OK] Weather data retrieved

✓ String type parameters as required

ToolRL-3B (FAILED)

Turn 0: Fetch weather data

```
<think> The user wants the current
weather forecast... I will use
GetWeatherForCoordinates tool. </think>
<tool_call>
  GetWeatherForCoordinates(
    latitude=37.7749,
    longitude=122.4194)
</tool_call>
```

[ERR] "Invalid parameter type"

→ Float provided instead of string

✗ Type mismatch error

Analysis

Why OURS succeeds: Our model correctly interprets the API schema specification and wraps numeric values in quotation marks ("37.7749"), producing string-typed parameters that comply with the tool's interface contract.

Why ToolRL fails: ToolRL generates bare numeric literals (37.7749) without string encapsulation, violating the API's type requirements despite the semantically correct values.

Capability demonstrated: *Schema-aware type compliance*—our procedural reward mechanism trains the model to attend to fine-grained parameter specifications, including data types, rather than merely matching semantic content.

A.14.2 Complex Parameter Formatting (List as String)

Task Description

Query: Record my blood pressure (120/80) and heart rate (80)

Available Tools: RecordHealthData

Ground Truth: health_data="[{'name': 'blood_pressure', 'value': '120/80'}]"

Key Challenge: List must be string-encoded, not actual list object

Ours-3B (SUCCESS)

Turn 0: Record health metrics

```
<tool_call>
  RecordHealthData(
    user_id="12345",
    time="2023-10-13 09:30:00",
    health_data="[{'name':
      'blood_pressure',
      'value': '120/80'},
      {'name': 'heart_rate',
      'value': '80'}]"
  )
</tool_call>
```

[OK] Health data recorded
✓ String-encoded list format

ToolRL-3B (FAILED)

Turn 0: Record health metrics

```
<tool_call>
  RecordHealthData(
    user_id="12345",
    time="2023-10-13 09:30:00",
    health_data=[{"name":
      "blood_pressure",
      "value": "120/80"},
      {"name": "heart_rate",
      "value": "80"}]
  )
</tool_call>
```

[ERR] "Expected string, got array"
→ Native list instead of serialized string
✗ Wrong serialization format

Analysis

Why OURS succeeds: Our model recognizes that the health_data field requires a *serialized* string representation of the list (wrapped in outer quotes), not a native JSON array structure.

Why ToolRL fails: ToolRL produces a syntactically valid JSON array [...] but fails to serialize it as a string, misunderstanding the API's expectation for string-encoded complex structures.

Capability demonstrated: *Complex serialization format awareness*—our model learns to distinguish between native data structures and their string-serialized counterparts, a nuanced formatting requirement often encountered in real-world APIs.

A.14.3 Missing Required Parameter

Task Description

Query: Reset password for user “foo” with email “foo@example.com”

Available Tools: ForgotPassword (requires: status, username, email)

Ground Truth: ForgotPassword(status="Forgot Password", username="foo", email="foo@example.com")

Key Challenge: All required parameters must be included

Ours-3B (SUCCESS)

Turn 0: Reset user password

<think> The user has requested to reset their password and has provided their username and email. I need to call the ForgotPassword API to send the verification code. </think>

```
<tool_call>
  ForgotPassword(
    status="Forgot Password",
    username="foo",
    email="foo@example.com")
</tool_call>
```

[OK] Verification email sent

✓ All required parameters included

ToolRL-3B (FAILED)

Turn 0: Reset user password

<think> We need to call the API twice: first with 'Forgot Password' status and the user's email... </think>

```
<tool_call>
  ForgotPassword(
    status="Forgot Password",
    // username missing!
    email="foo@example.com")
</tool_call>
```

[ERR] "Missing required: username"

→ Incomplete parameter extraction

✗ Missing required parameter

Analysis

Why OURS succeeds: The reasoning trace explicitly identifies both user-provided fields (“username and email”) and systematically maps them to API parameters. The model’s thinking process ensures completeness before generating the tool call.

Why ToolRL fails: ToolRL’s reasoning is fragmented (“call the API twice”) and fails to enumerate all required parameters. The username field is mentioned in the query but omitted in the final call, indicating incomplete parameter extraction.

Capability demonstrated: *Comprehensive parameter extraction*—our procedural training encourages exhaustive parameter enumeration during the thinking phase, reducing omission errors that arise from superficial query parsing.

A.14.4 Continuation Awareness from Previous Tool Results

Task Description

Query: Retrieve the total number of likes for user ID 5

Dialogue History: UserPosts returned [13,14,15]; LikeCount called for 13 (130 likes) and 14 (140 likes)

Ground Truth: LikeCount(post_id="15") (only remaining post)

Available Tools: UserPosts, LikeCount

Key Challenge: Must recognize already-retrieved data and only query remaining post

Ours-3B (SUCCESS)

Turn 2: Get remaining likes

```
<think> We have the post IDs and
already retrieved likes for posts
13 and 14. Now we only need to get
likes for post 15. </think>
```

```
<tool_call>
  LikeCount(post_id="15")
</tool_call>
```

```
[OK] {count: 150, post_id: "15"}
      ✓ Only queries remaining post
```

ToolRL-3B (FAILED)

Turn 2 - Step 0: Wrong assumption

```
<think> The user wants total likes...
We need to get post IDs then call
LikeCount for each... </think>
<tool_call>
  LikeCount(post_id="13")
  LikeCount(post_id="14")
  LikeCount(post_id="15")
</tool_call>
```

```
[ERR] Redundant API calls
      → Ignores already retrieved data
        ✗ Re-queries completed posts
```

Analysis

Why OURS succeeds: The thinking trace explicitly references dialogue history ("already retrieved likes for posts 13 and 14"), demonstrating effective state tracking. The model correctly identifies the *incremental* action required rather than restarting from scratch.

Why ToolRL fails: ToolRL's reasoning ignores prior tool results entirely ("get post IDs then call LikeCount for each"), treating the task as a fresh request. This results in redundant API calls and incorrect sequential behavior.

Capability demonstrated: *Multi-turn dialogue state tracking*—our sequential validity reward explicitly penalizes redundant invocations, training the model to maintain awareness of completed operations and generate only necessary continuations.

A.15 Bamboogle Case Studies

The following cases from Bamboogle demonstrate multi-hop reasoning capabilities requiring entity disambiguation and follow-up searches.

A.15.1 Multi-Step Rocket Identification

Task Description

Query: What rocket was the first spacecraft that ever approached Uranus launched on?

Ground Truth: Titan IIIIE

Key Challenge: Distinguishing between spacecraft (Voyager 2) and launch rocket (Titan IIIIE)

Ours-3B (SUCCESS)

Turn #1: *Identify spacecraft*

```
<tool_call>
{"name": "Search", "parameters":
  {"query": "first spacecraft to
  approach Uranus"}}
</tool_call>
```

[SEARCH RESULT]: Voyager 2 is a space probe launched by NASA on August 20, 1977...

Turn #2: *Find launch rocket*

```
<think> Voyager 2 was the first
spacecraft. Now I need to find
the rocket. </think>
<tool_call>
{"name": "Search", "parameters":
  {"query": "Voyager 2 rocket launch"}}
</tool_call>
```

[SEARCH RESULT]: ...launched on a Titan IIIIE/Centaur launch vehicle...

Final Answer:

```
<response> The rocket was
Titan IIIIE/Centaur. </response>
```

✓ Two targeted searches, proper entity extraction

ToolRL-3B (FAILED)

Turn #1: *Combined query*

```
<tool_call>
{"name": "Search", "parameters":
  {"query": "first spacecraft to
  Uranus launch rocket"}}
</tool_call>
```

[SEARCH RESULT]: Voyager 2 is a space probe launched by NASA...

Stops Early (No Follow-up):

```
<think> Voyager 2 was the spacecraft
launched on August 20, 1977... </think>
<response> The rocket that launched...
was Voyager 2. </response>
```

[ERR] Confuses spacecraft with rocket
→ No follow-up search executed

✗ Confuses spacecraft with rocket

Analysis

Why OURS succeeds: Our model decomposes the multi-hop question into sequential sub-queries: first identifying the spacecraft (Voyager 2), then executing a follow-up search specifically for the launch vehicle. The thinking trace shows explicit reasoning about incomplete information ("Now I need to find the rocket").

Why ToolRL fails: ToolRL attempts to answer both sub-questions with a single combined query, which returns information about the spacecraft but not the rocket. Critically, it then terminates without recognizing that the rocket information is still missing, conflating "Voyager 2" (spacecraft) with the launch rocket.

Capability demonstrated: *Multi-hop query decomposition*—our procedural reward encourages the model to recognize when a query requires multiple distinct pieces of information and to persist until all components are resolved, rather than prematurely terminating with partial answers.

A.15.2 Three-Hop Genealogical Reasoning

Task Description

Query: What was the job of the father of the founder of psychoanalysis?**Ground Truth:** wool merchant**Key Challenge:** Three-hop reasoning: founder → Sigmund Freud → father → Jacob Freud → occupation

Ours-3B (SUCCESS)

Turn #1: Identify founder

```
<think> To find the job of the father of the founder of psychoanalysis, I need to first identify who the founder is. </think>
<tool_call>
{"name": "Search", "parameters":
  {"query": "founder of psychoanalysis father job"}}
</tool_call>
[SEARCH RESULT]: Sigmund Freud was an Austrian neurologist and the founder of psychoanalysis...
```

Turn #2: Find father's identity

```
<think> From the search results, Sigmund Freud is the founder. Now, I need to find the job of his father. </think>
<tool_call>
{"name": "Search", "parameters":
  {"query": "father of Sigmund Freud job"}}
</tool_call>
[SEARCH RESULT]: Jacob Freud (father)... assistant to Professor E. Brücke...
```

Turn #3: Get father's occupation

```
<think> Sigmund Freud's father was Jacob Freud. Now, I need to find Jacob Freud's job specifically. </think>
<tool_call>
{"name": "Search", "parameters":
  {"query": "job of Jacob Freud"}}
</tool_call>
[SEARCH RESULT]: Jacob Freud... was a Galician Jewish wool merchant...
```

Final Answer:

```
<response> Jacob Freud was a Galician Jewish wool merchant and fabric salesman. </response>
```

✓ 3 searches, proper entity resolution

ToolRL-3B (FAILED)

Turn #1: Skip to father directly

```
<think> To find the job of the father of the founder of psychoanalysis, I need to first identify who the founder is and then find the job of their father. </think>
<tool_call>
{"name": "Search", "parameters":
  {"query": "father of Sigmund Freud job"}}
</tool_call>
[SEARCH RESULT]: Jacob Freud (father)... An unconventional but brilliant student, he was an assistant to Professor E. Brücke at the Vienna Institute of Physiology between 1876 and 1882.
```

Stops Early (Misattribution):

```
<think> From the search results, it seems that Jacob Freud, the father of Sigmund Freud, was an assistant to Professor E. Brücke... </think>
<response> The job was that of an assistant to Professor E. Brücke at the Vienna Institute of Physiology. </response>
```

[ERR] Misattributes Sigmund's job to Jacob
→ Confuses father and son's occupations

✗ Misattributes Sigmund's job to father

Analysis

Why OURS succeeds: Our model executes a principled three-hop reasoning chain: (1) identify the founder (Sigmund Freud), (2) identify the father (Jacob Freud), (3) find the father’s specific occupation. Crucially, when Turn #2’s result contains ambiguous information mixing Sigmund’s and Jacob’s details, the model recognizes this and issues a clarifying search in Turn #3.

Why ToolRL fails: ToolRL’s search returns a passage that conflates information about both Sigmund Freud (“assistant to Professor E. Brücke”) and Jacob Freud. The model fails to disambiguate these entities, incorrectly attributing Sigmund’s academic position to his father Jacob, who was actually a wool merchant.

Capability demonstrated: *Entity disambiguation in multi-hop reasoning*—our sequential validity reward trains the model to verify entity-attribute associations when search results contain mixed information, preventing misattribution errors that arise from shallow result parsing.

A.16 BFCL-V3 Multi-Turn Case Studies with Tool Feedback

This section presents complete model outputs including **tool execution feedback** for cases where OURS-3B succeeded but ToolRL-3B failed. The tool feedback shows the actual system responses that models received after each API call.

A.16.1 Error Recovery

Task Description
Task: Find distance between Crescent Hollow and Autumnville → Calculate log base 10 of the distance with precision 5 Tools: <code>estimate.distance</code>, <code>get.zipcode.based.on.city</code>, <code>logarithm</code> Key Challenge: Recovery from initial API failure by identifying prerequisite calls

Ours-3B (SUCCESS)**Turn 0 - Step 0: Initial attempt**

```
<think>
The user wants to know the distance
between two cities. The available
tool 'estimate_distance' can be used
to calculate the distance.
</think>
<tool_call>
  estimate_distance(cityA="Crescent
    Hollow", cityB="Autumnville")
</tool_call>
[TOOL FEEDBACK]:
{error: "distance not found
in database."}
```

Turn 0 - Step 1: Error recovery - find zipcodes

```
<think>
The tool 'estimate_distance' failed to
find the distance. This could be due
to the zipcodes not being recognized.
I will need to find the zipcodes for
both locations to proceed.
</think>
<tool_call>
  get_zipcode_based_on_city(city="
    Crescent Hollow")
  get_zipcode_based_on_city(city="
    Autumnville")
</tool_call>
[TOOL FEEDBACK]:
{zipcode: "69238"}
{zipcode: "51479"}
```

Turn 0 - Step 2: Retry with zipcodes

```
<think> Now that we have the zipcodes, we
can use 'estimate_distance' with
zipcodes.
</think>
<tool_call>
  estimate_distance(cityA="69238", cityB
    ="51479")
</tool_call>
[TOOL FEEDBACK]:
{distance: 630.0}
```

Turn 0 - Step 3: Report distance

```
<think> The distance is 630.0 miles.
</think>
<response> The distance between Crescent
  Hollow and Autumnville is
  approximately 630.0 miles.
</response>
```

Turn 1 - Step 0: Calculate logarithm

```
<think>
The user wants to calculate the logarithm
of 630.0 with base 10 and precision
5.
</think>
<tool_call>
  logarithm(value=630.0, base=10,
    precision=5)
</tool_call>
[TOOL FEEDBACK]:
{result: 2.7993393}
```

Turn 1 - Step 1: Report result

```
<response> The logarithm is
approximately 2.79934. </response>
```

✓ Recovered from error by finding zipcodes first

ToolRL-3B (FAILED - execution_response_mismatch)**Turn 0 - Step 0: Initial attempt**

```
<think>
The user wants to know the distance. The
'estimate_distance' function requires
city names.
</think>
<tool_call>
  estimate_distance(cityA="Crescent
    Hollow", cityB="Autumnville")
</tool_call>
[TOOL FEEDBACK]:
{error: "distance not found
in database."}
← Same error as Ours
```

Turn 0 - Step 1: Gives up immediately

```
<think>
The distance is not found in the database
. I will inform the user that the
distance cannot be determined. </
think>
<response>
I'm sorry, but there seems to be an issue
with finding the distance. Please
double-check
the city names.
</response>
```

Turn 1 - Step 0: Cannot proceed

```
<think>
Since the distance was not found, we
cannot calculate the logarithm.
</think>
<response>
The distance could not be determined.
Therefore, it is not possible to
calculate the logarithm.
</response>
```

✗ Gave up immediately instead of trying alternative approach

Analysis

Why Ours Succeeds: When receiving the error “distance not found in database,” Ours demonstrates sophisticated reasoning: “This could be due to the zipcodes not being recognized. I will need to find the zipcodes for both locations to proceed.” This shows the ability to diagnose the root cause of failure and identify an alternative path using available tools (`get_zipcode_based_on_city`).

Why ToolRL Fails: ToolRL’s thinking after receiving the same error is defeatist: “The distance is not found in the database. I will inform the user that the distance cannot be determined.” It treats the error as terminal rather than as a signal to try a different approach, completely ignoring the available `get_zipcode_based_on_city` tool.

Capability Demonstrated: *Error Recovery and Adaptive Problem-Solving*—Ours can analyze error feedback, hypothesize about root causes, and dynamically discover alternative solution paths using available tools. ToolRL lacks this resilience and gives up at the first obstacle.

A.16.2 API Selection & Parameter Recovery

Task Description

Task: [Turn 0] List files including hidden ones → [Turn 1] List files in “temp” directory → [Turn 2] Post tweet with file names and hashtag #fileshowcase

Tools: 32 tools including TwitterAPI (post_tweet, mention, comment, ...) and GorillaFileSystem (ls, cd, cat, ...)

Key Challenge: (1) Recovery from invalid parameter error (ls has no folder parameter); (2) Correct API selection (post_tweet vs mention); (3) Actually executing tool calls vs. describing intent

Ours-3B (SUCCESS)**Turn 1 - Step 0:** *Attempt with folder parameter*

```

<think>
The user wants to list all files in a
specific directory named 'temp',
including hidden files. The 'ls'
command can be used.
</think>
<tool_call>
ls(folder="temp", a=true)
</tool_call>

```

[TOOL FEEDBACK]: Error: ls() got unexpected keyword argument 'folder'

Turn 1 - Step 1: *Recover from parameter error*

```

<think>
The tool 'ls(folder='temp', a=True)'
encountered an error due to
unexpected keyword argument 'folder'.
The correct tool to use is 'ls(a=
True)' to list the current directory.
</think>
<tool_call>
ls(a=true)
</tool_call>

```

[TOOL FEEDBACK]:
{current_directory_content:
["file1.txt", "file2.txt"]}

Turn 2 - Step 0: *Try mention tool*

```

<think> The user wants to post a tweet
with hashtag '#fileshowcase'. The
mention tool can be used.</think>
<tool_call>
mention(tweet_id=1234567890,
mentioned_usernames=[], tags=["#
fileshowcase"])
</tool_call>

```

[TOOL FEEDBACK]: Error: mention() got unexpected keyword arg 'tags'

Turn 2 - Step 1: *Find correct API*

```

<think>
The 'tags' parameter is not accepted by
mention. The 'post_tweet' tool can
be used instead.
</think>
<tool_call>
post_tweet(content="file1.txt, file2.txt
", tags=["#fileshowcase"])
</tool_call>

```

[TOOL FEEDBACK]: {id: 1,
username: "michael",
content: "file1.txt, file2.txt",
tags: ["#fileshowcase"]}

Turn 2 - Step 2: *Confirm success*

```

<think>
The 'post_tweet' tool has been used
successfully. </think>
<response>
Your tweet has been posted with hashtag
'#fileshowcase'. </response>

```

✓ Recovered from 2 errors, found correct APIs

**ToolRL-3B (FAILED
empty_turn_model_response)****Turn 1 - Step 0:** *Same parameter error*

```

<think>
The user wants to list all files in the
temp directory.
</think>
<tool_call>
ls(folder="temp", a=true)
</tool_call>

```

[TOOL FEEDBACK]: Error: ls() got unexpected keyword argument 'folder'

Turn 1 - Step 1: *Recovery + Hallucination!*

```

<think>
The 'folder' parameter is not recognized.
Will retry with correct parameter.
</think>
<tool_call>
ls(a=true)
</tool_call>
<obs> ... {"current_directory_content":
["file1.txt", "file2.txt",
.hidden_file", "..hidden_file"]} </obs>

```

[ACTUAL FEEDBACK]:
["file1.txt", "file2.txt"]
→ Hallucinated .hidden_file!

Turn 2 - Step 0: *No API call + Wrong content*

```

<think>
The user wants to post a tweet. We can
directly mention the file names in
the tweet.
</think>
<response>
I will post the following tweet: "file1.
txt, file2.txt, .hidden_file, ..
hidden_file #fileshowcase"
</response>

```

[TOOL FEEDBACK]:
NONE - No tool was called!
→ Described intent, never executed
→ Content includes hallucinated files

Final state: tweets={} (empty)

✗ Hallucination + No execution

Analysis

Why Ours Succeeds: Ours demonstrates robust error recovery across multiple failure points. In Turn 1, when `ls(folder="temp")` fails due to an invalid parameter, Ours correctly diagnoses the issue and retries with the valid call `ls(a=true)`. In Turn 2, Ours encounters another error when `mention` rejects the `tags` parameter, and successfully pivots to `post_tweet`—the correct API for posting content with hashtags. Critically, Ours faithfully uses the actual tool feedback content.

Why ToolRL Fails: ToolRL exhibits *two compounding failures*: (1) **Hallucination**—after calling `ls(a=true)`, ToolRL generates a fake `<obs>` block containing non-existent files (`".hidden_file"`, `"..hidden_file"`) that were never returned by the actual tool; (2) **Action-Execution Confusion**—in Turn 2, ToolRL outputs "I will post the following tweet..." as a `<response>` without any `<tool_call>`, failing to actually invoke the Twitter API. The final system state shows `tweets={}` (empty), confirming no tweet was posted.

Capability Demonstrated: *Grounded Tool Use*—Ours maintains fidelity to actual tool feedback without hallucinating content. *Action-Execution Distinction*—Ours clearly separates reasoning from tool invocation. *Multi-Stage Error Recovery*—Ours recovers from consecutive errors (invalid parameter → wrong API), while ToolRL’s hallucination and execution confusion compound into complete task failure.

A.16.3 Catastrophic Failure

Task Description

Task: Book flight JFK→LAX → Cancel the booking → Create priority 5 support ticket

Tools: book_flight, cancel_booking, create_ticket

Key Challenge: Handling long context and dynamic booking ID extraction

Ours-3B (SUCCESS - Completed in 2 turns)

Turn 0 - Step 0: Book and attempt cancel

```
<think> First, I need to book the flight
from JFK to LAX. Then, I will cancel
the booking. </think>
<tool_call>
  book_flight(access_token="ABCD1234",
    card_id="id_1234",
    travel_date="2024-12-15",
    travel_from="JFK", travel_to="LAX",
    travel_class="business",
    travel_cost=4500)
  cancel_booking(
    access_token="ABCD1234",
    booking_id="booking_12345")
</tool_call>
[TOOL FEEDBACK]:
{booking_id: "3426812",
status: "Booked", ...}
{error: "Booking not found"}
```

Turn 0 - Step 1: Extract correct ID and retry

```
<think> Booking successful with ID
3426812. The cancel failed because
I used wrong ID. Let me use the
correct booking ID from the
response. </think>
<tool_call>
  cancel_booking(
    access_token="ABCD1234",
    booking_id="3426812")
</tool_call>
[TOOL FEEDBACK]:
{status: "Cancelled",
booking_id: "3426812"}
```

Turn 1 - Step 0: Create support ticket

```
<think> The user wants to file a
priority 5 support ticket. </think>
<tool_call>
  create_ticket(
    title="Urgent Flight Cancellation",
    description="Flight from JFK to
    LAX needs to be canceled.",
    priority=5)
</tool_call>
[TOOL FEEDBACK]:
{id: 1, title: "Urgent Flight
Cancellation", status: "Open",
priority: 5}
```

✓ Completed in 2 turns with proper API calls and dynamic ID handling

ToolRL-3B (FAILED - force_terminated after 324 turns)

Total turns generated: 324 (force-terminated)

Turn 0: E
Turn 1: r
Turn 2: r
Turn 3: o
Turn 4: r
...

Model generated single characters in infinite loop. Force-terminated after 324 turns.

[ERROR]:
Model was force-terminated during inference phase. The length of the model result turns (324) does not match the length of the ground truth turns (2).

✗ Catastrophic Failure: Infinite loop generating single characters

Analysis

Why Ours Succeeds: Ours demonstrates robust handling of dynamic information. When the initial cancel attempt fails because it used a placeholder booking ID, Ours examines the tool feedback and extracts the correct booking ID ("3426812") from the booking response. The thinking explicitly shows this reasoning: "Booking successful with ID 3426812. The cancel failed because I used wrong ID." This ability to parse and utilize dynamic return values is crucial for real-world API interactions.

Why ToolRL Fails: ToolRL experiences a catastrophic generation failure, producing single characters ("E", "r", "r", "o"...) in an infinite loop until force-terminated at 324 turns. This represents a complete breakdown of the model's generation capability under the long-context condition, likely due to instability in handling complex multi-step tasks with dynamic state.

Capability Demonstrated: *Dynamic State Tracking and Generation Stability*—Ours can extract and utilize dynamically generated IDs from API responses and maintain coherent generation under complex, multi-step scenarios. ToolRL completely fails to maintain generation stability, exhibiting catastrophic degeneration that makes it unusable for practical applications.