

HardGen: Generating Hard Samples for Tool-use Agents

Bingguang Hao^{1,2*}, Zengzhuang Xu^{2*}, Yuntao Wen^{2*}, Xinyi Xu^{2*}, Quanen Sun^{2*},
Yang Liu², Zihao Huang³, Tong Zhao³, Long Chen², Dong Wang², Yicheng Chen²,
Maolin Wang¹, Cunyin Peng^{2†}, Xiangyu Zhao¹, Chenyi Zhuang^{2‡}, Ji Zhang^{4‡}

¹City University of Hong Kong ²Inclusion AI, Ant Group

³Zhejiang University ⁴Southwest Jiaotong University

{bingguanghao7, jizhang.jim}@gmail.com chenyi.zcy@antgroup.com

Abstract

The advancement of LLM agents with tool-use capabilities requires diverse and complex training corpora. Existing data generation methods, which predominantly follow the paradigm of random sampling and shallow generation, often yield simple and homogeneous trajectories that fail to capture complex, implicit logical dependencies. To bridge this gap, we introduce **HardGen**, an automatic agentic pipeline designed to generate hard tool-use training samples. *First*, HardGen establishes a dynamic API Graph built upon agent failure cases, from which it samples hard tool-execution sequences. *Second*, these execution sequences are used to reversely build advanced tools that combine multi-step operations into one, which then help generate hard queries requiring unstated intermediate steps. *Finally*, advanced tools and hard queries drive the generation of Chain-of-Thought (CoT), which is iteratively refined through closed-loop verification feedback. Extensive evaluations demonstrate that a 4B parameter model trained with our curated dataset achieves superior performance compared to several leading open-source and closed-source competitors (e.g., GPT-5.2, Gemini-3-Pro and Claude-Opus-4.5). Our code, models and the curated dataset with 27,000 samples will be open-sourced to facilitate future research.

1 Introduction

Equipping Large Language Models (LLMs) with the capability to execute external tools¹, primarily through Function Calling (FC), has significantly expanded the boundaries of artificial intelligence (Wang et al., 2026; Shen, 2024; Wang et al., 2025). As agents are deployed in increasingly sophisticated scenarios—from enterprise workflow automation to intricate data analysis—the demand

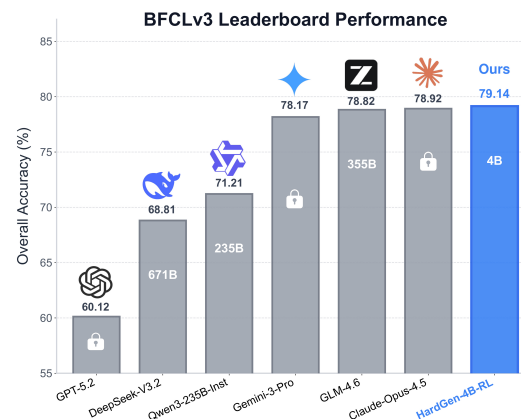


Figure 1: **Results on the BFCLv3 Leaderboard** (Patil et al., 2025). A HardGen trained 4B model (**HardGen-4B-RL**) consistently surpasses leading frontier models.

for high-quality, diverse, and complex training corpora has become the primary bottleneck limiting their advancement (Liu et al., 2024; Zeng et al., 2026). Although recent work has made strides in synthesizing diverse datasets, existing data generation pipelines adhere predominantly to the paradigm of random sampling and shallow generation—randomly selecting API combinations from static tool pools and generation of trajectories through basic user-assistant simulation (Huang et al., 2025; Prabhakar et al., 2025). Trajectories generated by those pipelines tend to be homogeneous and follow a “happy path”, failing to capture the *unstated intermediate steps* and *multi-turn reasoning* required in hard real-world tasks. For instance, real-world problems often necessitate bridging logical gaps, where the output of one tool serves as a latent precondition for another, or implicitly parameterizes the subsequent tool call. Current pipelines often miss these deep structural complexities, resulting in agents that are robust on simple queries but fragile when facing hard, logically-intertwined instructions (Zhang et al., 2026a).

To bridge this gap, we introduce **HardGen**, an

*Equal contribution. † Project lead.

‡Corresponding author.

¹We use tools and APIs interchangeably in this paper.

automated agentic pipeline designed to synthesize challenging training samples for overcoming the performance bottleneck of tool-use agents. Unlike previous pipelines that focus primarily on broadening API coverage or enforcing syntactic correctness, HardGen prioritizes the logical complexity and difficulty of the generated trajectories. The core philosophy of HardGen is to automatically generate challenging training samples that capture complex, real-world tool-use scenarios. Specifically, instead of sampling from a static tool pool, HardGen establishes a dynamic API Graph derived explicitly from agent failure cases. This allows the pipeline to sample *hard tool-execution sequences* that target the specific weaknesses of models. Next, we utilize these sequences as conditional priors to guide the generation of advanced tools that combine multi-step operations into one. These tools are then leveraged as stepping stones to formulate hard queries that necessitate multi-step reasoning and implicit dependency, moving beyond simple API pattern matching. At the end, the synergy of advanced tools and hard queries facilitates the generation of Chain-of-Thoughts (CoTs), with a closed-loop evaluation feedback mechanism steering the continuous refinement of the generation process, where reasoning correctness is further verified through a function call checking module. This robust, repeatable pipeline enables HardGen to construct trajectories of high diversity and complexity for tool-use agents.

Generated Dataset. Using HardGen, we build a large-scale dataset of 27,000 trajectories covering 2,095 APIs from real environments (see [Section 3.6](#)). The dataset features diverse and challenging interaction scenarios, especially hard queries requiring implicit dependency inference and multi-step reasoning without explicit guidance. This high-fidelity corpus provides a rigorous foundation for training tool-use agents with deep reasoning and dependency modeling capabilities.

Remarkable Results. We evaluate HardGen by applying SFT and RL training to Qwen3-4B using the curated HardGen dataset, producing **HardGen-4B-SFT** and **HardGen-4B-RL**. On challenging benchmarks including BFCLv3, HardGen-4B-RL achieves an overall accuracy of **79.14**, establishing a new state of the art at the 4B scale (see [Table 3](#)). Despite its compact size, it surpasses strong proprietary models, outperforming OpenAI GPT-5.2 by **19.02**, DeepSeek-V3.2 by **10.33** (see [Figure 1](#)). These gains further generalize to additional tool-

use benchmarks, including **ACEBench**, **APIBank**, and **BFCLv4 Agentic**, demonstrating robust agentic reasoning rather than template memorization.

Our contributions are summarized as follows:

- We propose HardGen, a novel pipeline designed to generate hard samples for tool-use agents.
- HardGen is compatible with a wide spectrum of APIs and models, facilitating the synthesis of tool-use datasets characterized by both high fidelity and complexity.
- Extensive experiments demonstrate that models trained on our generated data achieve superior performance, allowing a 4B parameter model to surpass powerful competitors.

2 Related Work

Tool-use Ability of LLMs. Empowering LLMs to interact with external APIs is pivotal for the realization of autonomous agentic systems ([Wang et al., 2026](#); [Hao et al., 2026](#)). By bridging the gap between static knowledge and dynamic environment, this paradigm enables models to interrogate databases, synthesize code, and manipulate digital interfaces ([Jimenez et al., 2024](#); [Mohammadjafari et al., 2024](#); [Gao et al., 2024](#); [Zhang et al., 2026c](#)). However, the transition from isolated function invocations to coherent, multi-turn tool orchestration presents a substantial barrier ([Guo et al., 2025](#); [Chen et al., 2025](#)). In these scenarios, agents are required to maintain state and resolve dependencies over extended interaction horizons. The core bottleneck is the lack of large-scale, verifiable training corpora that faithfully encode realistic multi-turn interactions, including implicit preconditions, parameter couplings, and cross-turn logical dependencies ([Prabhakar et al., 2025](#); [Xu et al., 2025](#)).

Data Synthesis for Tool-use Training. Current paradigms in tool-use data generation focus predominantly on expanding breadth and ensuring executability ([Yin et al., 2025](#); [Acikgoz et al., 2025](#)), typically via synthesis from fixed tool sets or multi-agent simulations ([Prabhakar et al., 2025](#); [Yin et al., 2025](#)). Although these methods scale dataset size ([Xu et al., 2025](#)), they suffer from a critical limitation: the difficulty of the generated trajectories is often artificial—derived from explicit structural constraints—rather than reflecting the intrinsic reasoning hurdles and implicit dependencies characteristic of authentic agentic workflows ([Zeng et al.,](#)

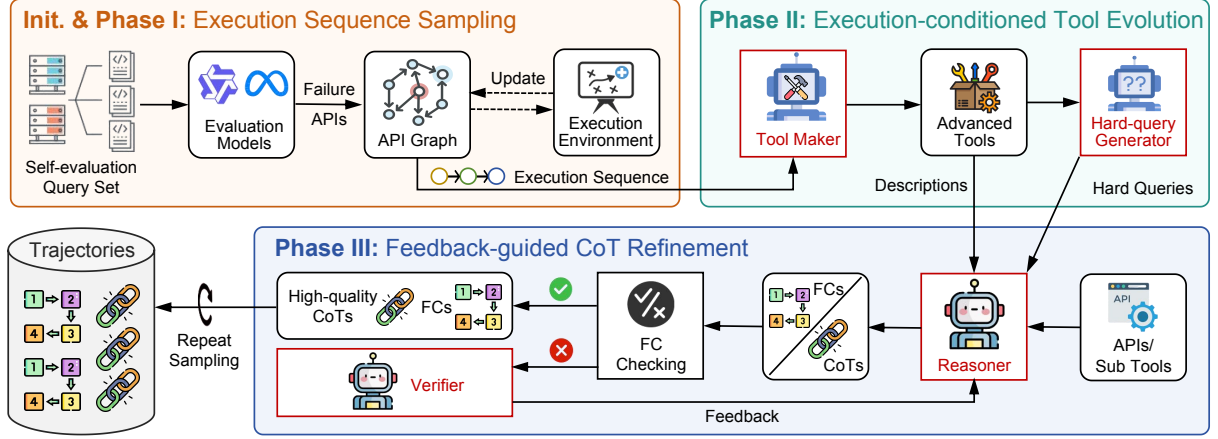


Figure 2: **Overview of HardGen.** The framework consists of three phases: **I) Execution Sequence Sampling** for constructing error-prone traces; **II) Execution-conditioned Tool Evolution** for synthesizing advanced tools and hard queries; and **III) Feedback-guided CoT Refinement** for closed-loop reasoning correction.

Symbol	Concept
$\mathcal{S}$	Environment Simulation Space
$\mathcal{V}$	Tool Node Set (all 2,095 tools)
$\mathcal{T} \subseteq \mathcal{V}$	Failure API Set (1,204 tools)
$\mathcal{G} = (\mathcal{V}, \mathcal{D}, \mathcal{P})$	API Graph (Tool Nodes $\mathcal{V}$, Edge set $\mathcal{D}$, Parameter Constraints $\mathcal{P}$)
$T_a \in \mathcal{T}$	Target / Anchor Tool of a trajectory
$T_c \in \mathcal{V}$	Generic candidate tool considered for sampling
$\text{Pre}(T_c) \subseteq \mathcal{V}$	Prerequisite tools of T_c (in-neighbours in $\mathcal{G}$)
$\mathcal{H} \subseteq \mathcal{V}$	History of executed tools (replaces $\mathcal{T}_{\text{called}}$)
$\mathcal{T}_{\text{legal}}(\mathcal{H})$	Legal candidate set at the current step
A_T, A_Q, A_R, A_V	Agents: Tool Maker, Hard-query Generator, Reasoner, Verifier
Q_{hard}	Hard Query
T_{adv}	Advanced Tool
K_i^*	Iteration index at which step i first succeeds
M	Number of Tool Executions Per Turn
N	Number of Turns Per Trajectory

Table 1: **Core notations.** Summary of symbols and definitions used in HardGen.

2026; Lam et al., 2024; Xu et al., 2025). In contrast, our HardGen aims to generate complex, verifiable tool-use trajectories that deliver grounded reasoning for SFT and precise rewards for RL (Qian et al., 2025; Hao et al., 2025), thereby driving superior performance in agentic tasks.

3 The Proposed HardGen Pipeline

In this section, we present HardGen, an automatic pipeline for generating challenging tool-use training samples (Figure 2); the corresponding notation is summarized in Table 1.

3.1 Initialization

Instead of uniformly sampling tool combinations, HardGen uses failure-driven self-evaluation to identify tools and dependencies that challenge the current model. The resulting API Graph captures failure-prone tools and their interaction patterns.

Failure-driven Self-evaluation and Graph Construction. We deploy a large-scale API environment comprising 2,095 tools, which together form the node set $\mathcal{V}$ of the API Graph, to perform model self-evaluation. For each tool, HardGen generates a tool-execution sequence and constructs a corresponding hard query, forming a Self-evaluation Query Set used to evaluate Qwen3-4B (Yang et al., 2025) and Llama-3.2-3B-Instruct (Grattafiori et al., 2024); the whole construction procedure is detailed in Appendix B. A tool is designated as challenging if both models produce incorrect execution results during inference. Through this process, we identify 1,204 challenging tools, which constitute the Failure API Set $\mathcal{T} \subseteq \mathcal{V}$ and serve as the admissible *target* tools when sampling on the API Graph $\mathcal{G}$, enabling systematic capture of challenging tools and their dependencies. The graph construction and its update are shown in Appendix C. By performing self-evaluation and graph construction on Qwen and Llama models, our approach demonstrates robust performance across heterogeneous architectures as shown in Table 3, Table 5 and Table 16, validating the strong generalization capability of the HardGen synthesized data.

Dependency Filtering. The initial API Graph $\mathcal{G} = (\mathcal{V}, \mathcal{D}, \mathcal{P})$ (see Appendix C) contains spurious yet type-compatible dependencies. We therefore

apply two-stage filtering. First, **Semantic Compatibility Filtering** checks whether the output domain of T_i aligns with the semantic constraints of T_j 's input using type signatures, value ranges, and API descriptions. Second, **Execution Validation** executes each surviving edge in the sandbox: if T_j fails or produces invalid results using T_i 's output, the edge is removed. After filtering, 5,783 of 8,462 candidate edges (68.3%) remain over the full node set $\mathcal{V}$ of 2,095 tools, among which $\mathcal{T}$ contains 1,204 failure tools. Full graph construction and dynamic updates are provided in **Appendix C**.

3.2 Phase I: Execution Sequence Sampling

Sampling tool-execution sequences over the API Graph $\mathcal{G}$ must avoid two pitfalls: producing operationally invalid calls that violate tool dependencies, and drifting away from the failure-prone targets in $\mathcal{T}$. Phase I therefore (i) restricts sampling to dependency-legal tools at every step, and (ii) biases the trajectory toward a chosen target $T_a \in \mathcal{T}$, yielding executable yet challenging tool-execution sequences Γ_i as input to Phase II.

Legality-constrained Sampling. To guarantee execution validity, tool selection is subject to a strict dependency constraint. For any candidate tool $T_c \in \mathcal{V}$, let $\text{Pre}(T_c) \subseteq \mathcal{V}$ denote the set of its prerequisite tools (i.e., the in-neighbours of T_c in $\mathcal{G}$, induced by the edge set $\mathcal{D}$); T_c is callable if and only if every tool in $\text{Pre}(T_c)$ has already been executed. We denote the history of executed tools at the current step by $\mathcal{H} \subseteq \mathcal{V}$ (used in place of $\mathcal{T}_{\text{called}}$ to avoid confusion with the Failure API Set $\mathcal{T}$). This legality condition is formalized as

$$I(T_c, \mathcal{H}) = \mathbf{1}_{\{\text{Pre}(T_c) \subseteq \mathcal{H}\}}, \quad (1)$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function. We further define the set of legal tools at the current step as

$$\mathcal{T}_{\text{legal}}(\mathcal{H}) = \{T_c \in \mathcal{V} \setminus \mathcal{H} : I(T_c, \mathcal{H}) = 1\}. \quad (2)$$

Tool-execution Sequence Sampling. To reach the selected target tool T_a while satisfying its prerequisite constraints, we design a *Sampler* that explicitly biases execution sequence construction toward T_a . Sampling is run for a fixed budget of M steps: at each step, the Sampler employs a greedy heuristic over the legal candidate set $\mathcal{T}_{\text{legal}}(\mathcal{H})$, prioritizing tools that minimize the graph distance $\text{dist}(\cdot, \cdot)$ to T_a , where $\text{dist}(T_k, T_a)$ denotes the length of the shortest path from T_k to T_a in $\mathcal{G}$. Once the target T_a has been collected into $\mathcal{H}$, the remaining steps

until the budget M is reached are drawn uniformly at random from $\mathcal{T}_{\text{legal}}(\mathcal{H})$ in order to enrich the surrounding workflow. The resulting sampling policy is formally defined as:

$$\begin{aligned} T_s &= \text{Sampler}(T_a, \mathcal{H}) \\ &= \begin{cases} \text{rand}(\mathcal{T}_{\text{legal}}(\mathcal{H})), & T_a \in \mathcal{H}, \\ T_a, & I(T_a, \mathcal{H}) = 1 \wedge T_a \notin \mathcal{H}, \\ \arg \min_{T_k \in \mathcal{T}_{\text{legal}}(\mathcal{H})} \text{dist}(T_k, T_a), & \text{otherwise.} \end{cases} \end{aligned} \quad (3)$$

The sampled tool T_s is executed with parameters P_s as a call $C_s = (T_s, P_s)$, producing environment feedback E_s and updating the system state $\mathcal{S}$, the history of executed tools $\mathcal{H}$, and the dynamic API graph $\mathcal{G}$. Sampling terminates once the prescribed budget of M steps has been collected, and the resulting execution trace $\Gamma_i = (C_1, E_1, \dots, C_M, E_M)$ serves as a failure-aware prior for subsequent phases.

3.3 Phase II: Execution-conditioned Tool Evolution

The hard execution sequence Γ_i from Phase I guarantees correct execution and captures the underlying workflow. However, robust tool-use agents must also learn *implicit logical bridging*—inferring missing intermediate steps from underspecified queries. Directly generating queries from explicit traces weakens this ability, as models may rely on pattern matching rather than genuine reasoning. To address this, Phase II reformulates the execution sequence through a two-stage process: first abstracting the multi-step workflow into a high-level operation, and then constructing a hard query that implicitly requires this abstraction.

Advanced Tool Construction. Given Γ_i , the *Tool Maker* (A_T) synthesizes a high-level abstraction, denoted as the advanced tool T_{adv} :

$$T_{\text{adv}} = A_T(\Gamma_i), \quad (4)$$

where A_T abstracts the multi-step tool-execution sequence into a high-level operation that encapsulates the collective functionality and interdependencies of all tool calls within Γ_i .

Hard Query Generation. Conditioned on the advanced tool T_{adv} , the *Hard-query Generator* (A_Q) generates a challenging query Q_{hard} whose solution requires realizing the functionality abstracted by T_{adv} using primitive tools. Formally, the process is expressed as:

$$Q_{\text{hard}} = A_Q(T_{\text{adv}}). \quad (5)$$

3.4 Phase III: Feedback-guided CoT Refinement

While the hard query Q_{hard} from Phase II introduces implicit logical dependencies, it also increases the difficulty of generating correct and coherent CoT reasoning. To address this, Phase III adopts an iterative refinement mechanism that progressively corrects flawed reasoning through feedback-driven prompting, steering the CoT toward valid solutions.

Reasoning with Hint. Given the hard query Q_{hard} , the *Reasoner* A_R attempts to generate the correct function call with the hint from the description of the advanced tool T_{adv} . The initial prompt, denoted as $\text{Prompt}_i^{(1)}$, is constructed from the hard query, the primitive tool set used to define the advanced tool, and the advanced tool description.

Error Diagnosis. If the attempt fails ($\text{FC}_i^{(k)} \neq C_i$), the *Verifier* A_V compares it with the ground truth, identifies the error and provides corrective feedback $\text{Error}_i^{(k)}$ without revealing the answer.

CoT Refinement. At attempt k , the corrective hint $\text{Error}_i^{(k)}$ is appended to the current prompt:

$$\text{Prompt}_i^{(k+1)} = \text{Concat}(\text{Prompt}_i^{(k)}, \text{Error}_i^{(k)}). \quad (6)$$

Using the updated prompt, A_R generates a refined reasoning process $\text{CoT}_i^{(k+1)}$ and function call $\text{FC}_i^{(k+1)}$. Refinement continues until the function call is correct or the maximum attempt limit K_{max} is reached.

Let $K_i^* = \min\{k \leq K_{\text{max}} : \text{FC}_i^{(k)} = C_i\}$ denote the iteration at which step i first succeeds. The successful prompt of step i together with the executed call and feedback is then used to initialize step $i+1$:

$$\text{Prompt}_{i+1}^{(1)} = \text{Concat}(\text{Prompt}_i^{(K_i^*)}, C_i, E_i). \quad (7)$$

This conditions subsequent reasoning on the full execution history. A trajectory is retained only if all M function calls are successfully generated within K_{max} attempts each.

Repeating the three-phase pipeline produces reliable yet challenging multi-turn trajectories, forming a high-quality corpus for tool-use training.

Multi-turn Dialogue Construction. A trajectory contains N user turns, each with at most M tool executions. The first turn uses the procedure above. For every subsequent turn, A_Q is conditioned on the accumulated dialog, verified calls and feedback,

Metric	Min	Max	Mean
Dependency Depth (d)	0	6	2.14
Latent Variable Count (ℓ)	0	5	2.37
Branching Factor (b)	3.2	47.8	18.6

Table 2: **Hardness metric statistics over the HardGen dataset.** The dataset spans a broad difficulty spectrum, from single-step to deeply chained multi-tool reasoning.

and the current state of the sandbox, and generates a follow-up goal that depends on the information or the state established earlier. Phase III verifies and refines each turn, and the complete trajectory is retained only if every turn executes successfully. Thus, *multi-turn* denotes context-dependent user-agent dialogue, rather than multiple calls within one user turn.

3.5 Definition of Hardness

To move beyond intuitive notions of difficulty, HardGen also defines three quantitative hardness metrics grounded in the structure of the API Graph $\mathcal{G}$ and the generated queries.

Dependency Depth (d). The length of the shortest directed path in $\mathcal{G}$ from any root node to the target tool T_a , where $d = 0$ indicates that T_a is itself a root node and requires no prerequisite. A tool sequence with $d = 1$ requires only a single prerequisite, while $d \geq 2$ implies a chain of latent preconditions.

Latent Variable Count (ℓ). The number of intermediate tool calls in Γ_i whose outputs serve as required inputs to downstream calls but whose execution is not explicitly referenced in the hard query Q_{hard} . Formally, $\ell = |\{C_j \in \Gamma_i : C_j \text{ provides a parameter to some } C_k \text{ with } k > j, \text{ and } C_j \text{ is not referenced in } Q_{\text{hard}}\}|$.

Branching Factor (b). The average number of legal tools available at each sampling step: $b = \frac{1}{M} \sum_{i=1}^M |\mathcal{T}_{\text{legal}}(\mathcal{H}_i)|$, where $\mathcal{H}_i$ is the execution history just before step i and $\mathcal{T}_{\text{legal}}(\mathcal{H}_i)$ is the corresponding legal candidate set defined in Eq. 2.

3.6 Generated Dataset

Dataset Statistics. We deploy HardGen on a large-scale real-world API environment comprising 2,095 tools, which form the node set $\mathcal{V}$ of the API Graph $\mathcal{G}$. Among them, 1,204 tools are identified as failure tools through systematic self-evaluation and constitute the Failure API Set $\mathcal{T} \subseteq \mathcal{V}$ that supplies the admissible target tools, while the remaining tools serve as legitimate intermediate

Model	Parameter Counts	Single-Turn			Multi-Turn					Overall
		Non-Live	Live	Subset Overall	Base	Miss Func	Miss Param	Long Context	Subset Overall	
Closed-source										
Claude-Opus-4-5-20251101	-	88.58	79.79	84.19	81.00	64.00	58.00	70.50	68.38	78.92
Claude-Sonnet-4-5-20250929	-	88.65	81.13	84.89	69.00	65.00	52.50	59.00	61.38	77.05
Gemini-3-Pro-Preview	-	90.65	83.12	86.89	64.50	60.00	54.50	64.00	60.75	78.17
Grok-4-1-fast-reasoning	-	88.27	78.46	83.37	70.50	59.50	43.00	62.50	58.88	75.20
GPT-5.2-2025-12-11	-	81.85	70.39	76.12	36.50	18.00	27.50	30.50	28.13	60.12
GPT-4o-2024-11-20	-	83.88	70.54	77.21	55.50	34.50	29.00	51.00	42.50	65.64
Open-source										
Kimi-K2-Instruct	1043B	81.60	78.68	80.14	62.00	41.00	44.50	55.00	50.63	70.30
DeepSeek-V3.2-Exp	671B	85.52	76.02	80.77	55.00	49.00	27.00	48.50	44.88	68.81
Llama-4-Maverick	400B	88.65	73.65	81.15	27.00	22.00	14.00	18.00	20.25	60.85
GLM-4.6	355B	87.56	80.90	84.23	74.50	68.00	63.00	66.50	68.00	78.82
Qwen3-235B-A22B-Instruct	235B	90.33	78.68	84.51	54.00	42.50	31.50	50.50	44.63	71.21
Qwen3-32B	32B	88.77	82.01	85.39	56.00	52.50	40.00	43.00	47.88	72.88
Qwen3-30B-A3B-Thinking	30B	85.77	77.94	81.86	43.50	10.50	25.00	41.00	30.00	64.57
ToolACE-2-8B	8B	87.10	77.42	82.26	49.00	28.00	30.50	46.00	38.38	67.63
ToolACE-MT	8B	84.94	71.52	78.23	57.50	31.50	34.00	38.00	40.25	65.57
Nanbeige4-3B-Thinking-2511	3B	81.58	79.42	80.50	58.50	54.00	45.00	47.00	51.12	70.71
xLAM-2-3b-fc-r	3B	82.96	62.92	72.94	71.50	59.00	57.50	45.50	58.38	68.09
Qwen3-4B (Base Model)	4B	87.88	76.39	82.14	26.50	21.00	15.50	25.50	22.13	62.13
HardGen-4B-SFT	4B	89.03	82.42	85.73	55.20	49.10	41.40	52.90	49.65	73.70
Δ		+1.15	+6.03	+3.59	+28.70	+28.10	+25.90	+27.40	+27.52	+11.57
HardGen-4B-RL	4B	90.73	83.55	87.14	68.50	64.50	50.50	69.00	63.13	79.14
Δ		+2.85	+7.16	+5.00	+42.00	+43.50	+35.00	+43.50	+41.00	+17.01

Table 3: **Performance on BFCLv3** (last updated on 2025-12-16). All metrics are calculated using the official script and reported in terms of Accuracy (%), and Δ values are gains over the Qwen3-4B base model.

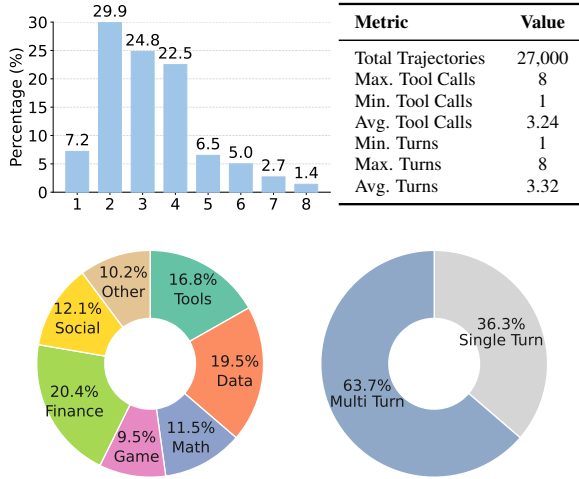


Figure 3: **Statistics and distribution of the generated dataset.** (Top) Histogram of total tool calls per trajectory (left) and key dataset metrics (right). (Bottom) Distribution of API domains (left) and the proportion of single and multiple dialogue turns (right).

steps along prerequisite chains. From $\mathcal{G}$ we synthesize 27,000 trajectories using Qwen3-30B-A3B-Thinking (Yang et al., 2025) as the agent backbone with $K_{\max} = 3$ refinement attempts. To quantitatively characterize dataset difficulty, we compute all three hardness metrics across the full corpus (Table 2): dependency depth averages $d = 2.14$ (max 6), latent variable count averages $\ell = 2.37$,

and the mean branching factor $b = 18.6$ (range from 3.2 to 47.8) confirms substantive decision complexity at each sampling step.

As shown in Figure 3, the dataset exhibits substantial diversity across task complexity, domain, and reasoning structure. Each trajectory contains 1 to 8 tool calls in total (average 3.24), with 62.9% containing three or more calls. API coverage spans seven domains, namely Finance (20.4%), Data (19.5%), Tools (16.8%), Social (12.1%), Math (11.5%), Other (10.2%), and Game (9.5%). Orthogonally to tool-call count, 63.7% of trajectories are multi-turn, requiring sustained implicit reasoning across dialogue turns. Notably, single-turn trajectories may still contain multiple parallel tool calls, so the 36.3% single-turn ratio does not correspond to the 37.1% of trajectories with fewer than three tool calls. Additional cases and prompts are provided in Appendix L and Appendix M.

Training-test Tool Overlap Audit. To rule out benchmark overfitting, we audit tool overlap between our 2,095-tool synthesis environment and the four evaluation benchmarks via (i) exact name/signature matching and (ii) semantic matching using sentence-embedding similarity at $\tau=0.85$. As shown in Table 4, the total overlap rate is at most 4.35% (BFCLv3) and remains below 3% on the other benchmarks. To further isolate the methodology from environment leakage, we remove all

Benchmark	Tools	Exact	Semantic	Overlap Rate
BFCLv3	2,000	28	59	4.35%
BFCLv4 Agentic	34	0	0	0.00%
APIBank	2,964	33	42	2.53%
ACEBench	4,538	45	64	2.40%

Table 4: **Tool-level overlap between the HardGen synthesis environment and evaluation benchmarks.**

overlapping tools and their associated trajectories ($\sim 0.4\%$ of the 27,000-trajectory corpus) and re-train HardGen-4B-RL under identical settings: the held-out model attains **78.87** BFCLv3 Overall, only 0.27 points below the full model (**79.14**). This confirms that HardGen’s gains stem from the generation method rather than from environment overlap.

4 Experiments

In this section, we evaluate our proposed pipeline by performing Supervised Fine-Tuning (SFT) and Reinforcement Learning (RL) on several baseline models using the HardGen curated dataset.

4.1 Experiment Setup

Baseline Models. We use Qwen3-4B (Yang et al., 2025) as the main baseline, and additionally evaluate Qwen2.5-7B-Instruct (Yang et al., 2024), Llama-3.2-3B-Instruct, Llama-3.1-8B-Instruct (Grattafiori et al., 2024), and Qwen3-0.6B/1.7B to verify generalization. We split the 27,000 trajectories at each assistant turn and train models to predict only the current response. SFT and RL are conducted with LLaMA-Factory (Zheng et al., 2024) and Verl (Sheng et al., 2025), respectively. Implementation details are provided in **Appendix G**.

Benchmarks. We evaluate on BFCLv3 (Patil et al., 2025), APIBank (Li et al., 2023), and ACEBench (Chen et al., 2025), covering both single-turn and multi-turn tool-calling scenarios (see **Appendix E**). To assess generalization and agentic capabilities, we additionally include BFCLv4 Agentic (Patil et al., 2025), which evaluates Web Search and Memory scenarios.

4.2 Experimental Results

Results on BFCLv3. As shown in **Table 3**, HardGen substantially improves Qwen3-4B on BFCLv3 (Patil et al., 2025), increasing the multi-turn score from 22.13 to 49.65 (**+27.52**) after SFT and to 63.13 (**+41.00**) after RL. Despite having only 4B parameters, HardGen-4B-RL surpasses all strong competitors, establishing a new state of

Model	Single-Turn	Multi-Turn	Overall
Base Model	77.30	7.62	54.07
TOUCAN	76.51	22.62	58.55
InfTool	81.18	22.75	61.70
ToolACE-MT	80.51	27.57	62.86
+ HardGen-SFT	83.99	40.75	69.58
Δ	+6.69	+33.13	+15.51

Table 5: **Comparison with state-of-the-art data synthesis pipelines on BFCLv3.** The baseline model is Qwen2.5-7B-Instruct.

Variant	P-I	P-II	P-III	Overall
Base Model	-	-	-	62.13
(a) Random	Random	-	-	64.90
(b) Failure	Failure graph	-	-	67.64
(c) Failure+ T_{adv}	Failure graph	T_{adv}	-	70.18
(d) Failure+Refine	Failure graph	-	Refinement	69.23
(e) HardGen (Full)	Failure graph	T_{adv}	Refinement	73.70

Table 6: **Ablation study of the HardGen training stages.** Each variant removes or modifies one Phase (P-I to P-III) of the pipeline.

the art at its scale. HardGen-trained models also achieve balanced gains across all sub-metrics, indicating strong generalization and stability. Notably, both HardGen-4B-SFT and HardGen-4B-RL outperform Qwen3-30B-A3B-Thinking (Yang et al., 2025) across all sub-metrics. These results show that failure-driven sampling, execution-conditioned tool evolution, and feedback-guided refinement jointly amplify capability beyond the generator model’s inherent limits. Llama-family results in **Appendix I** further demonstrate consistent gains across architectures.

Results on Other Benchmarks. **Figure 4(a)** and **Figure 4(b)** show results on APIBank and ACEBench. On APIBank (Li et al., 2023), our models achieve top-tier Level-1 accuracies of 71.9 and 69.9, surpassing GPT-4o (66.7). On the more challenging Level-2 tasks, they improve over the base model (33.3) by **+12.6** and **+8.2**, respectively. On ACEBench (Chen et al., 2025), our models further demonstrate strong generalization under both training settings. For single-turn tasks, they achieve 81.5 and 80.5 accuracy, outperforming GPT-4o (78.0) and Llama-3.1-8B-Instruct (39.8). The gains are even larger in multi-turn settings, where our models reach 76.0 and 74.0, exceeding Llama-3.1-8B-Instruct (28.0) by **+48.0** and **+46.0**. Overall, these results demonstrate strong and consistent tool-use capabilities across diverse benchmarks. Additional results on agent-specific benchmarks are provided in **Appendix F**.

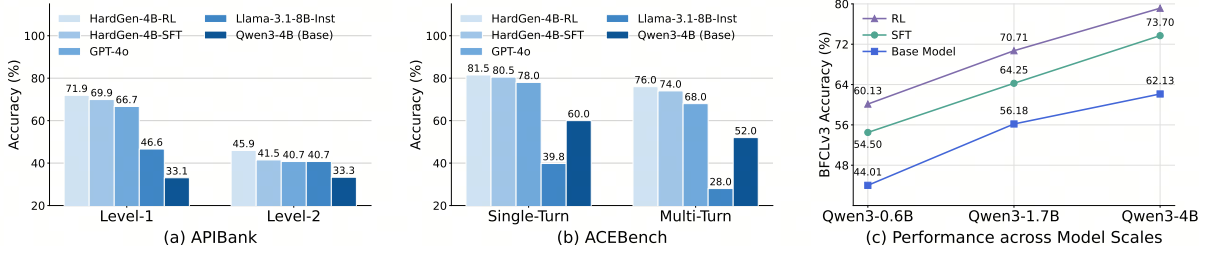


Figure 4: **Additional evaluations.** (a) Comparison of different models on APIBank. (b) Comparison of different models on ACEBench. (c) Scaling trends of HardGen-SFT and HardGen-RL on BFCLv3 across model parameters.

Comparison with State-of-the-art. To further demonstrate the efficacy of HardGen, we compare it against state-of-the-art data synthesis pipelines using Qwen2.5-7B-Instruct under an SFT-only setting, as reported in Table 5. While prior methods such as InfTool (Li et al., 2025), TOUCAN (Xu et al., 2025) and ToolACE-MT (Zeng et al., 2026) offer incremental gains over the base model, HardGen establishes a clear and consistent performance advantage. Notably, our method secures a score of 83.99 (+6.69) in single-turn tasks and delivers a striking 40.75 (+33.13) in multi-turn interactions. This substantial margin validates HardGen as a more robust and effective data generation strategy for complex tool-use scenarios.

4.3 Further Analysis

Component Ablation. We compare HardGen with four simplified variants, all built on the same 2,095 tool environment and 27,000 training trajectories for fair comparison. All models are trained with SFT on Qwen3-4B and evaluated on BFCLv3 (Table 6). Randomly sampled execution traces with direct query construction and CoT distillation (a) improve over the base model by +2.77, but remain 8.80 points below full HardGen. Failure-oriented execution sampling (b) brings an additional +2.74 gain over random sampling. Building on (b), advanced-tool-based hard query construction (c) and CoT refinement (d) further improve performance by +2.54 and +1.59. The full pipeline (e) achieves the best performance, showing that all phases are complementary. Fine-grained analyses for each phase are provided in Appendix H.

Hardness-Error Correlation. To analyze the relationship between difficulty and model performance, we conduct the hardness-error correlation analysis by randomly selecting 5,000 trajectories from the HardGen-generated dataset. Table 7 reports the Pearson correlation coefficients (r) between each

Metric	Range	r with Base Err.	r with Δ Acc
Dep. Depth (d)	0–6	0.71	0.58
Latent Vars (ℓ)	0–5	0.68	0.63
Branch Factor (b)	3.2–47.8	0.42	0.37

Table 7: **Correlation between hardness metrics and model behavior.** Pearson correlations (r) between hardness metrics, base model error rate, and accuracy gain (Δ Acc). All correlations are positive and statistically significant ($p < 0.01$).

hardness metric and two key outcomes: (1) the error rate of the base model (Qwen3-4B) on the corresponding trajectories, and (2) the accuracy improvement (Δ Acc) achieved after SFT on HardGen data. All correlations are positive and statistically significant ($p < 0.01$). The three metrics show positive correlations with the base model error rate ($r \geq 0.42, p < 0.01$), confirming that structurally more complex trajectories are indeed harder for untrained models. Importantly, the training benefit Δ Acc also correlates positively with hardness ($r \geq 0.37$), indicating that HardGen achieves the largest improvements on the hardest instances.

Tool-use and General Capability Trade-off. We study the trade-off between tool-use specialization and general capability using IFEval (Zhou et al., 2023) and BFCLv3 in Figure 5. Compared with the Qwen3-4B base model, training on HardGen leads to only modest IFEval degradation (83.4 to 81.4 for RL; 83.4 to 79.1 for SFT), while substantially improving BFCLv3 by +17.01 and +11.57, respectively. Among all compared models, HardGen-4B-RL achieves the strongest balance between the two capabilities, remaining close to the base model on IFEval while significantly outperforming it on BFCLv3. Moreover, RL is consistently more Pareto-efficient than SFT, improving both IFEval (+2.3) and BFCLv3 (+5.44). We attribute this favorable trade-off to failure-driven sampling and verifier-guided CoT refinement, which enhance tool-use

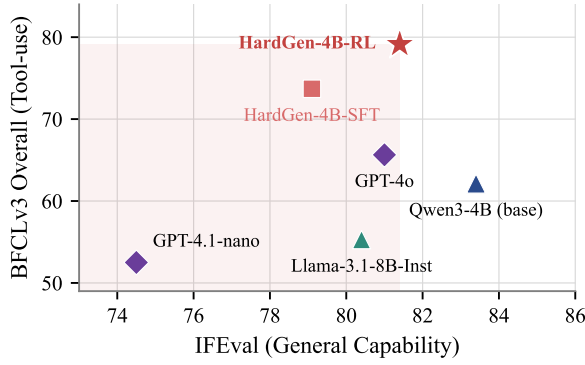


Figure 5: **Pareto analysis of tool-use and general capability.** HardGen-4B-RL (★) lies on the high-tool-use end of the Pareto frontier, achieving the best BFCLv3 score while outperforming most models on both axes.

ability while preserving natural reasoning.

5 Conclusion

In this work, we introduce HardGen, an automatic agentic pipeline designed to generate challenging tool-use training samples with verifiable reasoning. HardGen adopts a failure-driven approach, producing training samples that capture the implicit logical dependencies and multi-step reasoning characteristic of real-world tasks. Extensive experiments demonstrate that a 4B parameter model trained with our curated dataset achieves state-of-the-art performance on BFCLv3 for its scale, surpassing leading proprietary models. HardGen exhibits strong generalization across model architectures, parameter scales, and agentic benchmarks, paving the way for developing more robust models capable of complex tool-use and agentic reasoning.

Limitations

While HardGen demonstrates strong performance across multiple benchmarks and model architectures, several limitations merit consideration. 1) Although our evaluation covers 2,095 tools across diverse domains, the extent to which HardGen generalizes to entirely new API ecosystems or specialized domains remains to be fully explored. 2) Our approach requires executable environments for verification, which may not be feasible for proprietary APIs or tools with complex external dependencies. 3) The abstraction quality of advanced tools relies on the Tool Maker’s ability to correctly identify high-level operations from primitive tool sequences, which may occasionally produce sub-optimal abstractions for highly irregular or domain-

specific tool chains.

Ethical Considerations

This work aims to improve the tool-use and reasoning capabilities of large language model agents, which can enhance their usefulness in real-world applications but may also introduce risks such as unintended tool execution, unsafe actions, privacy leakage, and misuse in sensitive scenarios. Our experiments are conducted in simulated tool-use environments, and the training data are publicly available or synthetically generated without intentionally using personal or sensitive information. Models trained with our approach should be carefully evaluated and equipped with appropriate safeguards before real-world deployment.

Acknowledgments

We would like to thank all reviewers for constructive comments that helped us improve the paper. This research was partially supported by National Natural Science Foundation of China (No.62502404), Hong Kong Research Grants Council (Research Impact Fund No.R1015-23, Collaborative Research Fund No.C1043-24GF, RGC Research Fellow Scheme No.RFS2627-1S03, General Research Fund No.11218325, No.11212926), Institute of Digital Medicine of City University of Hong Kong (No.9229503), and Ant Group Research Internship Program.

References

- Emre Can Acikgoz, Jeremiah Greer, Akul Datta, Ze Yang, William Zeng, Oussama Elachqar, Emmanouil Koukoumidis, Dilek Hakkani-Tür, and Gokhan Tur. 2025. [Can a single model master both multi-turn conversations and tool use? coalm: A unified conversational agentic language model](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12370–12390.
- Chen Chen, Xinlong Hao, Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Yuefeng Huang, Xiangcheng Liu, Wang Xinzh, and Wu Liu. 2025. [ACEBench: A comprehensive evaluation of LLM tool usage](#). In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 12970–12998.
- Shen Gao, Yuntao Wen, Minghang Zhu, Jianing Wei, Yuhang Cheng, Qunzi Zhang, and Shuo Shang. 2024. Simulating financial market via large language model based agents. *arXiv preprint arXiv:2406.19966*.

- Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, Louis Rouillard, Thomas Mesnard, Geoffrey Cideron, Jean bastien Grill, Sabela Ramos, Edouard Yvinec, Michelle Casbon, Etienne Pot, Ivo Penchev, and 197 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.
- GLM Team, Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, Kedong Wang, Lucen Zhong, Mingdao Liu, Rui Lu, Shulin Cao, Xiaohan Zhang, Xuancheng Huang, Yao Wei, and 152 others. 2025. GLM-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.
- Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.
- 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, and 175 others. 2025. DeepSeek-R1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645:633–638.
- Bingguang Hao, Zengzhuang Xu, Maolin Wang, Yuntao Wen, Yicheng Chen, Cunyin Peng, Long Chen, Dong Wang, Xiangyu Zhao, Jinjie Gu, Chenyi Zhuang, and Ji Zhang. 2025. Reasoning through exploration: A reinforcement learning framework for robust function calling. *arXiv preprint arXiv:2508.05118*.
- Bingguang Hao, Zengzhuang Xu, Maolin Wang, Yuntao Wen, Yicheng Chen, Cunyin Peng, Long Chen, Xiangyu Zhao, Jinjie Gu, Chenyi Zhuang, and Ji Zhang. 2026. BalanceSFT: Improving LLM function calling with balanced training signals and data hardness. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 18094–18112.
- Chengrui Huang, Shen Gao, Zhengliang Shi, Dongsheng Wang, and Shuo Shang. 2025. TTPA: Token-level tool-use preference alignment training framework with fine-grained evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 16240–16255.
- Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.
- Long Hei Matthew Lam, Ramya Keerthy Thatikonda, and Ehsan Shareghi. 2024. A closer look at tool-based logical reasoning with llms: The choice of tool matters. In *Proceedings of the 22nd Annual Workshop of the Australasian Language Technology Association*, pages 41–63.
- Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116.
- Yuwen Li, Wei Zhang, Zelong Huang, Mason Yang, Jiajun Wu, Shawn Guo, Huahao Hu, Lingyi Sun, Jian Yang, Mingjie Tang, and Byran Dai. 2025. Close the loop: Synthesizing infinite tool-use data via multi-agent role-playing. *arXiv preprint arXiv:2512.23611*.
- Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Shirley Kokane, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh Murthy, Liangwei Yang, Silvio Savarese, Juan Carlos Niebles, Huan Wang, Shelby Heinecke, and Caiming Xiong. 2024. APiGen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37.
- Mistral AI. 2025. Mistral Small 3.2. Accessed on September 27, 2025.
- Ali Mohammadjafari, Anthony S Maida, and Raju Gotumukkala. 2024. From natural language to sql: Review of llm-based text-to-sql systems. *arXiv preprint arXiv:2410.01066*.
- Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The berkeley function calling leaderboard (BFCL): From tool use to agentic evaluation of large language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 48371–48392.
- Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Weiran Yao, Huan Wang, Silvio Savarese, and Caiming Xiong. 2025. Apigenmt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38.
- Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiuxi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. 2025. ToolRL: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38.
- Qwen Team. 2025. QwQ-32B: Embracing the power of reinforcement learning. Accessed on September 27, 2025.

- Zhuocheng Shen. 2024. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. [Hybridflow: A flexible and efficient rlhf framework](#). In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.
- Maolin Wang, Yingyi Zhang, Bowen Yu, Bingguang Hao, Cunyin Peng, Yicheng Chen, Wei Zhou, Jinjie Gu, Chenyi Zhuang, Ruocheng Guo, Wanyu Wang, and Xiangyu Zhao. 2026. [Function calling in large language models: Industrial practices, challenges, and future directions](#). *ACM Computing Surveys*, 58(9):1–37.
- Yanlin Wang, Xinyi Xu, Jiachi Chen, Tingting Bi, Wen-chao Gu, and Zibin Zheng. 2025. An empirical study of agent developer practices in ai agent frameworks. *arXiv preprint arXiv:2512.01939*.
- Zhangchen Xu, Adriana Meza Soria, Shawn Tan, Anurag Roy, Ashish Sunil Agrawal, Radha Pooven-dran, and Rameswar Panda. 2025. TOUCAN: Synthesizing 1.5m tool-agentic data from real-world mcp environments. *arXiv preprint arXiv:2510.01179*.
- An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.
- An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, and 23 others. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.
- Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, Hamid Palangi, and Tomas Pfister. 2025. [Magnet: Multi-turn tool-use data synthesis and distillation via graph translation](#). In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32600–32616.
- Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gao-hong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, and 17 others. 2025. [DAPO: An open-source llm reinforcement learning system at scale](#). *Advances in Neural Information Processing Systems*, 38.
- Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin Jiang, and Qun Liu. 2026. [Toolace-mt: Non-autoregressive generation for agentic multi-turn interaction](#). In *The Fourteenth International Conference on Learning Representations*.
- Kangning Zhang, Weiwen Liu, Wenxiang Jiao, Kouni-anhua Du, Yuan Lu, Weinan Zhang, and Yong Yu. 2026a. [LoopTool: Closing the data-training loop for robust LLM tool calls](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21144–21164.
- Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2026b. [Nemotron-research-tool-n1: Exploring tool-using language models with reinforced reasoning](#). In *The Fourteenth International Conference on Learning Representations*.
- Sheng Zhang, Junyi Li, Yingyi Zhang, Pengyue Jia, Yichao Wang, Xiaowei Qian, Wenlin Zhang, Maolin Wang, Yong Liu, and Xiangyu Zhao. 2026c. [MemSearch-o1: Empowering large language models with reasoning-aligned memory growth in agentic search](#). In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 925–943.
- Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyang Luo. 2024. [LlamaFactory: Unified efficient fine-tuning of 100+ language models](#). In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410.
- Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A Notation

To facilitate reading, we consolidate in [Table 1](#) all the symbols introduced throughout the HardGen pipeline. The notation is organized along four logical groups. **(1) Environment objects.** The simulation space $\mathcal{S}$ denotes the executable sandbox in which every tool call is grounded; the node set $\mathcal{V}$ contains all 2,095 tools registered in the API environment; the Failure API Set $\mathcal{T} \subseteq \mathcal{V}$ is the subset of 1,204 tools on which both evaluator models fail ([Section 3.1](#)); and the API Graph $\mathcal{G} = (\mathcal{V}, \mathcal{D}, \mathcal{P})$ aggregates the tool nodes $\mathcal{V}$, prerequisite edges $\mathcal{D}$, and parameter constraints $\mathcal{P}$ into a single dynamic structure that drives Phase I sampling, where the failure subset $\mathcal{T}$ specifies the set of admissible target tools. **(2) Tool-level sets.** For any candidate tool $T_c \in \mathcal{V}$, $\text{Pre}(T_c) \subseteq \mathcal{V}$ denotes its prerequisite tools (i.e., the in-neighbours of T_c in $\mathcal{G}$ induced by $\mathcal{D}$); $\mathcal{H} \subseteq \mathcal{V}$ tracks the history of tools that have

been executed so far at the current step (replacing the more easily confused $\mathcal{T}_{\text{called}}$); and $\mathcal{T}_{\text{legal}}(\mathcal{H})$ is the set of currently legal candidates derived from $\mathcal{H}$ and $\text{Pre}(\cdot)$. **(3) Agents.** Four specialized LLM agents play distinct roles in the pipeline: the Tool Maker A_T synthesizes advanced tools T_{adv} from execution sequences (Phase II), the Hard-query Generator A_Q produces hard queries Q_{hard} from T_{adv} , while the Reasoner A_R and Verifier A_V form the closed-loop refinement module of Phase III, where K_i^* denotes the iteration at which step i first succeeds. **(4) Trajectory dimensions.** A trajectory anchors on a selected target tool $T_a \in \mathcal{T}$ and contains M tool executions per turn over N turns, jointly characterizing the breadth and depth of the synthesized data.

These symbols recur consistently across Section 3 and the appendices, and the table can be consulted at any time as a one-stop reference for the meaning of each symbol used in equations, paragraph definitions, and pipeline descriptions.

B Construction of Self-evaluation Query Set

To systematically identify challenging tools, we construct a Self-evaluation Query Set over the full API environment of 2,095 tools. For each API T_a , we generate a self-evaluation instance through the following procedure.

Targeted Execution Sequence Generation. For each tool T_a in the tool library, HardGen treats T_a as the *target tool* and constructs an execution sequence Γ_a centered around it. Specifically, HardGen first identifies the prerequisite tools that T_a depends on within the API Graph $\mathcal{G}$, then executes them in valid order within the sandbox environment so that their outputs provide the necessary parameters for T_a . The sequence terminates with the invocation of T_a itself, producing a ground-truth execution sequence $\Gamma_a = (C_1, E_1, \dots, C_M^{T_a}, E_M)$, where $C_M^{T_a}$ is the call to the target tool and each E_k is the deterministic environment feedback.

Hard Query Construction. Based on the execution sequence Γ_a , HardGen constructs a corresponding hard query q_a that describes the end-goal accomplished by the *entire sequence* without explicitly enumerating the intermediate prerequisite steps, thereby requiring the model to autonomously infer the full tool chain needed to invoke T_a .

Dual-Model Evaluation and Tool Filtering.

Each constructed query q_a is fed to two heterogeneous lightweight models, Qwen3-4B (Yang et al., 2025) and Llama-3.2-3B-Instruct (Grattafiori et al., 2024). A tool T_a is designated as *challenging* if and only if **both** models produce incorrect outputs. Using two model families as dual evaluators mitigates single-model bias, while the conjunction criterion ensures that only *genuinely* difficult tools—rather than those that happen to confuse one specific architecture—are retained. Through this process, we identify **1,204** challenging tools, which are incrementally collected into the Failure API Set $\mathcal{T} \subseteq \mathcal{V}$ (while the full node set $\mathcal{V}$ of the API Graph $\mathcal{G}$ still contains all 2,095 tools as potential intermediate steps), enabling systematic capture of challenging tools and their structural dependencies for subsequent hard sample generation.

C Construction of the API Graph

The API Graph $\mathcal{G} = (\mathcal{V}, \mathcal{D}, \mathcal{P})$ is the core data structure underlying Phase I of HardGen. It encodes three types of information: (1) *Tool Node Set* $\mathcal{V}$: all 2,095 tools registered in the API environment, which form the vertices of $\mathcal{G}$ and from which the Failure API Set $\mathcal{T} \subseteq \mathcal{V}$ (the 1,204 failure APIs identified through dual-model self-evaluation in Section 3.1) is drawn as admissible *target* tools; (2) *Dependencies* $\mathcal{D}$: directed edges over $\mathcal{V}$ representing prerequisite relationships, where $(T_i, T_j) \in \mathcal{D}$ indicates that T_j requires T_i ’s output to be executed; and (3) *Parameter Constraints* $\mathcal{P}$: specifications defining valid parameter ranges, types, and inter-tool parameter mappings (e.g., the output type of T_i must match the input type of T_j), extracted from tool API schemas including type signatures, value ranges, and required fields. Keeping all 2,095 tools as nodes (rather than restricting $\mathcal{G}$ to the failure subset) is essential: many non-failure tools serve as legitimate intermediate steps that bridge prerequisite chains toward a failure target.

Graph Initialization. All 2,095 tools in $\mathcal{V}$ are registered as nodes. For every ordered pair (T_i, T_j) , HardGen performs static analysis over their API schemas: the output fields of T_i are compared against the input parameters of T_j by matching declared types (e.g., `string:zipcode` $\rightarrow$ `string:zipcode`). A candidate dependency edge is created whenever a type match is found. This schema-level process yields an initial graph with 8,462 candidate edges that captures the struc-

tural relationships latent in the API specifications. Since type matching alone may introduce false positives (*e.g.*, two string fields with incompatible semantics), these candidates are subsequently refined through the two-stage dependency filtering described in Section 3.1.

Dynamic Graph Update. Beyond the initial static construction, $\mathcal{G}$ is continuously refined through execution feedback collected during execution sequence sampling. Specifically, after each tool call $C_s = (T_s, P_s)$ is executed in the sandbox, two complementary update mechanisms are applied. For **edge activation**, when the execution of T_j succeeds using outputs produced by a previously executed T_i , the edge (T_i, T_j) is confirmed as a validated dependency; conversely, if T_j fails under the same conditions during sampling, the edge will be removed. For **constraint refinement**, parameter constraints in $\mathcal{P}$ are updated by tracking the value validity ranges and type requirements observed during execution—for example, if T_i ’s output field `account_id` consistently produces 8-digit numeric strings, this observed range is propagated to T_j ’s corresponding input constraint, tightening the original schema-level specification. This continual update process enables $\mathcal{G}$ to capture implicit value-level dependencies that extend beyond simple type-inclusion relationships, progressively biasing subsequent sampling toward tool sequences that are both executable and challenging.

D Data

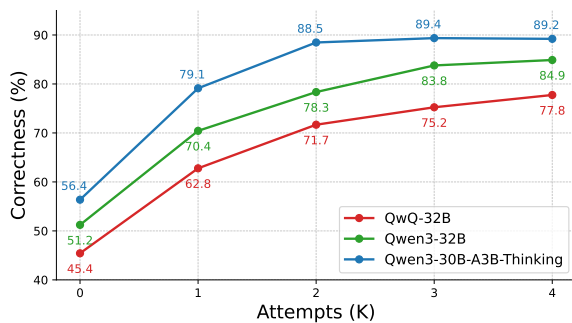


Figure 6: **Analysis of generator model selection.** Correctness rates of function call synthesis at different numbers of attempts (K) for three generator models.

Model Selection and Number of Attempts

To ensure computational efficiency for large-scale synthesis, we restrict candidate models around 30B parameter scale and evaluate three

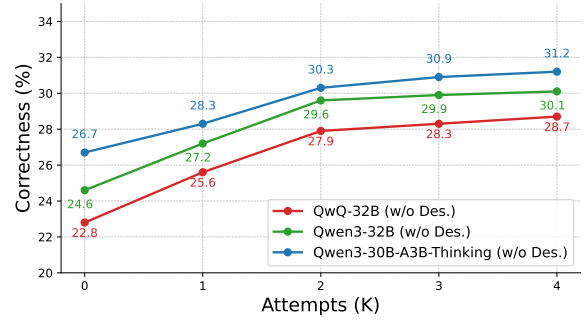


Figure 7: **Impact of advanced tool descriptions.** Correctness rates of the generator models across attempts (K) when advanced tool descriptions are omitted, illustrating the necessity of high-level abstractions for reasoning.

strong generators—QwQ-32B (Qwen Team, 2025), Qwen3-32B (Yang et al., 2025), and Qwen3-30B-A3B-Thinking (Yang et al., 2025) under identical HardGen configurations by having each model generate 2,000 trajectories. In Figure 6, Qwen3-30B-A3B-Thinking achieves the highest correctness rate across all K values, reaching 89% at $K=3$. Notably, this model activates only 3B parameters per forward pass through its mixture-of-experts architecture, enabling significantly faster generation than the dense 32B models while delivering superior performance. Its combination of efficient inference, robust multi-step reasoning, and high correctness on function call synthesis makes it the optimal choice for cost-effective large-scale data generation.

The Necessity of Advanced Tool Description for CoT.

Compared with the findings in Figure 6, Figure 7 further provides compelling evidence for the necessity of advanced tool descriptions in CoT generation. Across all models and attempt counts, removing descriptions results in uniformly poor performance, with maximum correctness rates below 32%. The limited differentiation between models—only 2.5 percentage points separate the best and worst performers at $K_{\max} = 4$, suggests that the correctness of reasoning is low without adequate tool information. Moreover, the diminishing returns from additional attempts (performance gains $< 1\%$ after $K = 2$) indicate that models cannot iteratively refine their selections through reasoning alone. These findings demonstrate that advanced tool descriptions constitute a fundamental prerequisite rather than a mere performance enhancement for enabling LLMs to engage in rigorous reasoning when confronted with hard queries.

Agent Backbone	Single-Turn	Multi-Turn	Overall
None (Base Model)	82.14	22.13	62.13
QwQ-32B	86.19	52.37	74.92
Qwen3-32B	86.87	58.29	77.34
Qwen3-30B-A3B-Thinking	87.14	63.13	79.14

Table 8: **Impact of data generation backbones.** Performance of the Qwen3-4B model trained on data synthesized by different agent backbones.

Data Generation with Other Agent Backbones.

To demonstrate the robustness and generalizability of HardGen across different agent backbones, we conduct ablation experiments using alternative model backbones for data generation. As shown in Table 8, we evaluate three models—Qwen3-32B (Yang et al., 2025), QwQ-32B (Qwen Team, 2025), and Qwen3-30B-A3B-Thinking (Yang et al., 2025), as the agent backbones for synthesizing training data, while keeping all other pipeline components identical. The results demonstrate the strong effectiveness of HardGen for synthetic data generation, yielding substantial improvements over the base model across all agent backbones. These consistent and pronounced gains indicate that HardGen’s strategy of generating challenging, agent-produced data effectively cultivates complex reasoning capabilities, with performance benefits scaling in tandem with the strength of the generation backbone.

E Evaluations

Recent efforts in benchmarking LLM tool-use have centered on three key axes: scalability, robustness, and realism.

BFCL. For scalability, the Berkeley Function Calling Leaderboard (BFCL) (Patil et al., 2025) introduces a novel validation strategy using Abstract Syntax Tree sub-string matching. This approach serves as a proxy for function execution, enabling large-scale, deterministic evaluation across diverse categories, including Single-Turn, Multi-Turn (BFCLv3), and Agentic scenarios (BFCLv4).

ACEBench. For robustness, the ACEBench (Chen et al., 2025) uses a sandbox environment that measures model performance on dynamic, simulated tasks. It uses two distinct metrics, End-to-End Accuracy, which compares the final instance attributes of the environment with the target state; and Process Accuracy, which measures the consistency between the actual function call process and the ideal process. This

Model	Web Search	Memory	Overall
Llama-3.1-8B-Instruct	3.00	10.75	6.88
CoALM-8B	0.00	2.80	1.40
ToolACE-2-8B	8.50	18.49	13.50
BitAgent-8B	4.00	12.47	8.24
xLAM-2-8b-fc-r	6.50	13.98	10.24
+ HardGen-RL	16.00	24.84	20.42
Δ	+13.00	+14.09	+13.54

Table 9: **Holistic Agentic Evaluation on BFCLv4.**

approach is designed to capture task completion in realistic, interactive scenarios.

APIBank. For realism, the APIBank (Li et al., 2023) establishes a framework for runnable evaluation, grading tasks into complex, multi-step sequences. The system verifies whether the same database queries or modifications are performed to ensure the ground-truth outcome is achieved.

F Performance on the Holistic Agentic Evaluation BFCLv4

Generalization of Agentic Capabilities. To assess the transferability of the tool-use skills induced by HardGen, we further evaluate our method on the holistic agentic evaluation BFCLv4 Agentic, which comprises two representative agentic subsets: Web Search and Memory.

Web Search. This subset evaluates an agent’s ability to answer complex multi-hop questions that require retrieving and synthesizing information from multiple web sources. Agents are equipped with a search API and a URL content retrieval tool, while the environment further introduces probabilistic network failures to simulate realistic deployment conditions. As shown in Table 9, HardGen-8B-RL achieves 16.00 on this subset, improving over the base Llama-3.1-8B-Instruct by **+13.00** points (from 3.00 to 16.00). It also substantially outperforms other 8B baselines, including CoALM-8B (0.00), BitAgent-8B (4.00), xLAM-2-8b-fc-r (6.50), and ToolACE-2-8B (8.50), demonstrating strong robustness in information-seeking scenarios that require multi-step retrieval and reasoning under realistic environmental uncertainty.

Memory. This subset evaluates an agent’s ability to maintain and utilize conversational context over extended interaction horizons. The evaluation is conducted across three memory backends: a structured Key-Value store for exact retrieval, a Vector Store for semantic retrieval, and a Recur-

Hyperparameter	Value
Batch Size	1024
Learning Rate	4e−5
Max Length	20480
Epoch Number	5

Table 10: Hyperparameters for SFT.

Hyperparameter	Value
Batch Size	512
Learning Rate	1e−6
KL Coefficient	1e−3
Entropy Coefficient	0
Max Length	20480
Temperature	0.7
Epoch Number	5
Rollout Number	16

Table 11: Hyperparameters for RL training.

sive Summarization store for narrative recall. On this subset, HardGen-8B-RL attains 24.84, yielding an improvement of **+14.09** points over the base model (10.75 to 24.84). In addition, it consistently surpasses all competing 8B baselines, including CoALM-8B (2.80), BitAgent-8B (12.47), xLAM-2-8b-fc-r (13.98), and ToolACE-2-8B (18.49). The larger gain on Memory suggests that HardGen is particularly effective at improving long-horizon agentic reasoning, where successful performance requires persistent context tracking and effective memory utilization across turns.

Overall, HardGen-8B-RL improves the overall BFCLv4 Agentic score from 6.88 to 20.42, corresponding to a gain of **+13.54** points over the base model. Moreover, it establishes a clear advantage over all other 8B baselines, outperforming ToolACE-2-8B by **+6.92** points, xLAM-2-8b-fc-r by **+10.18** points, BitAgent-8B by **+12.18** points, and CoALM-8B by **+19.02** points. These results demonstrate that the benefits of HardGen generalize effectively to agentic settings, enabling the model to acquire more robust capabilities in both external information retrieval and long-horizon contextual reasoning.

G Training details

Supervised Finetuning. We conduct our supervised finetuning experiments using the open-source LLaMA-Factory library (Zheng et al., 2024). The main hyperparameter settings are listed in Table 10.

RL Training. We conduct our reinforcement learning (RL) experiments using the open-source

Hard Query	Qwen3-4B		Llama-3.2-3B-Instruct	
	Single-Turn	Multi-Turn	Single-Turn	Multi-Turn
✗	84.17	54.38	79.08	32.23
✓	87.14	63.13	84.05	40.13

Table 12: **Impact of hard queries.** Both Qwen3-4B and Llama-3.2-3B-Instruct are trained with RL using data with or without hard queries.

Verl library (Sheng et al., 2025). To ensure stable and efficient training, we adopt the training settings of Zhang et al. (2026b), where reinforcement learning is applied directly without supervised fine-tuning stage. The key hyperparameter settings are summarized in Table 11. Both SFT and RL training runs are executed on a cluster equipped with 8 NVIDIA A100 80GB GPUs.

Reward Design. We adopt a simple yet effective reward that is widely used in prior works (Zhang et al., 2026b; Yu et al., 2025; GLM Team et al., 2025). Given a query q with reference answer g , the model’s output o is evaluated as follows. If o contains a tool call, it is considered correct only when it can be successfully parsed into valid function calls with proper parameters, exactly matches g , and follows the prescribed reasoning template. In contrast, if g does not contain a tool call, then o is considered correct only when it contains no valid function calls (*i.e.*, is free-form text) while still adhering to the required reasoning template. The reward is thus defined as:

$$\text{Reward} = \begin{cases} 1, & \text{format correct \& answer correct,} \\ 0, & \text{otherwise.} \end{cases} \quad (8)$$

This reward emphasizes the holistic integrity of the output, enforcing not only semantic correctness but also strict structural compliance, which is essential for reliable downstream execution. In non-tool-calling cases, the absence of valid function calls implicitly verifies that the model produces a purely textual response and prevents spurious or unnecessary tool invocations.

H Fine-grained Ablation

In this section, we present a fine-grained ablation to further scrutinize our proposed HardGen pipeline.

Does HardGen’s effectiveness scale with model size? To investigate the scalability of our approach, we evaluate the performance of the HardGen synthesized dataset across base models ranging from 0.6B to 4B parameters on the BFCLv3

Model	w/o T_{adv}	w/ T_{adv}
Qwen3-30B-A3B-Thinking	177	223
Qwen3-235B-A22B-Thinking	159	241
DeepSeek-V3.1 (671B)	125	275

Table 13: **Impact of the constructed advanced tools T_{adv} across different model scales.** The values denote the number of instances deemed more difficult by GPT-4o from a pool of 400 generated queries.

benchmark. As shown in Figure 4(c), model performance consistently improves with increasing scale under both training paradigms. Specifically, the Qwen3-0.6B model registers a gain of **+16.12** points (rising from 44.01 to 60.13) via RL, whereas Qwen3-4B realizes a more substantial improvement of **+17.01** points (62.13 to 79.14). Furthermore, RL consistently outperforms SFT across all model sizes. This trend indicates that HardGen-synthesized data is inherently well aligned with RL, enabling stronger base models—particularly at larger scales—to more effectively leverage RL for improving tool-use capabilities.

How much do hard queries affect the quality of the synthesized data? To assess the impact of the constructed hard queries on data quality, we perform ablation studies by conducting RL training on Qwen3-4B and Llama-3.2-3B. As shown in Table 12, incorporating hard queries leads to consistent performance improvements across both model architectures and evaluation tasks. Specifically, for Qwen3-4B, training with hard queries yields notable gains of **+8.75** points in the multi-turn subset (from 54.38 to 63.13) and **+2.97** points in the single-turn subset (from 84.17 to 87.14). Similarly, Llama-3.2-3B benefits from hard queries, achieving improvements of **+7.90** and **+4.97** points in the multi-turn and single-turn subsets, respectively. The consistent gains observed across architectures and evaluation tasks demonstrate that the constructed hard queries substantially enhance the quality of the synthesized data, leading to stronger tool-use capabilities, particularly in more challenging multi-turn scenarios, where the improvements are most pronounced.

Do advanced tools really help with making logical bridging? To evaluate the contribution of the constructed advanced tools T_{adv} to logical jump bridging, we conduct a controlled comparison between model variants with and without T_{adv} under identical configurations. Each model is evaluated

Models	Consistency		Pearson Corr.		Human Preference	
	w/o T_{adv}	w/ T_{adv}	w/o T_{adv}	w/ T_{adv}	w/o T_{adv}	w/ T_{adv}
Qwen3-30B-A3B-Thinking	0.86	0.87	0.72	0.74	182	218
Qwen3-235B-A22B-Thinking	0.89	0.91	0.78	0.82	163	237
DeepSeek-V3.1 (671B)	0.92	0.95	0.84	0.90	129	271

Table 14: **Agreement between manual and automatic annotations.** We report the Consistency rates and Pearson correlations between the two annotation methods, alongside the Human Preference.

on 400 query-synthesis instances, where every instance produces two queries: one that leverages T_{adv} for logical bridging and one that does not. GPT-4o is employed as an automated judge to determine which variant yields a more challenging query, *i.e.*, one involving more difficult logical jumps. As shown in Table 13, incorporating T_{adv} consistently increases the proportion of challenging queries across all model scales, with the number of such instances rising from 223 to 275 as model capacity increases from 30B to 671B. This trend indicates that T_{adv} effectively facilitates logical bridging, which is more pronounced with increasing model capacity. Given the subjective nature of *difficulty* judgments, we further complement a human annotation study.

The *difficulty* judgments are the product of automatic annotation (GPT-4o), and this evaluation task moves beyond simple surface-level text generation judgment. To further evaluate whether the model successfully navigates the hard logical jump that the non-advanced tool model failed, three PhD students in NLP field (three of the authors) form an annotation team to re-annotate the samples using logical jumps as the indicator. If there are differences between two annotations on a sample, the third annotation will be introduced to determine the final decision. From Table 14, we can observe a strong consistency between the automatic and manual annotations, with no significant differences between the human and model conclusions. Overall, these results demonstrate that the introduction of T_{adv} effectively enhances the model’s ability to construct logically jumping queries, with consistent benefits observed across model scales.

How significantly does feedback-guided CoT refinement boost reasoning? To quantify the impact of our feedback-guided CoT refinement, we compare model variants with and without Verifier feedback. Concretely, we set K_{max} to 3 as the max-

Model	w/o Feedback	w/ Feedback	Δ
Qwen3-30B-A3B-Thinking	77.80%	90.14%	+12.34%
Qwen3-235B-A22B-Thinking	82.57%	92.78%	+10.21%
DeepSeek-V3.1 (671B)	83.98%	95.60%	+11.62%

Table 15: **Impact of feedback-guided CoT refinement.** The Verifier feedback boosts the three models.

Models	Single-Turn	Multi-Turn	Overall
Llama-3.1-8B-Inst	77.38	11.12	55.29
+HardGen-RL	84.55	53.10	74.07
Llama-3.2-3B-Inst	70.50	4.00	48.33
+HardGen-RL	84.05	40.13	67.57

Table 16: **Generalization to the Llama model family.** Performance comparison of two Llama models before and after HardGen-RL training on BFCLv3 Single-Turn and Multi-Turn tasks.

imum number of refinement iterations. As reported in Table 15, incorporating the guided refinement loop results in substantial accuracy gains of **+12.34** on the 30B model, **+10.21** on the 235B model, and **+11.62** on the 671B model. These consistent improvements—each exceeding 10 percentage points across all evaluated model scales—highlight the critical role of feedback-guided refinement in steering model reasoning toward correct solutions, establishing it as an essential component of our framework. A detailed analysis of the refinement dynamics and the choice of $K_{\max}$ is provided in Appendix D.

I Results on Other Models

Results on Llama-3 Models. To assess the generalizability of our approach beyond Qwen models, we apply HardGen-RL to two Llama-3 variants, Llama-3.1-8B-Instruct and Llama-3.2-3B-Instruct. As shown in Table 16, both models demonstrate substantial improvements after reinforcement learning. Llama-3.1-8B-Instruct achieves an overall accuracy of 74.07 (**+18.78**), with particularly strong gains on multi-turn interactions (53.10, **+41.98**). Similarly, Llama-3.2-3B-Instruct improves from 48.33 to 67.57 overall (**+19.24**), with multi-turn performance increasing from 4.00 to 40.13 (**+36.13**). These results show that our approach generalizes across model families and scales, making HardGen a robust framework for improving capabilities beyond the Qwen architecture.

Failure-Labeling Criterion	Failure Tools	BFCLv3 Overall
Qwen3-4B only	1,386	71.92
Llama-3.2-3B only	1,451	70.88
Disjunction (either fails)	1,633	71.45
Random sampling (size-matched)	1,204	67.03
Conjunction (ours)	1,204	73.70

Table 17: **Ablation on the failure-labeling criterion for constructing the Failure API Set.** “Random sampling” draws the same number of tools as the conjunction rule uniformly at random from the 2,095-tool pool. All variants are trained under identical Phase II/III configurations with Qwen3-4B as the target model.

J Ablation on the Failure API Set Construction Criterion

The Failure API Set $\mathcal{T}$ is the cornerstone of Phase I: it determines which tools are eligible as target tools during trajectory sampling and, consequently, which trajectories are generated and further processed in Phases II and III. All 2,095 tools nevertheless remain as nodes in $\mathcal{G}$. We construct $\mathcal{T}$ via a *conjunction* criterion: a tool T_a is labeled as a failure tool only when **both** Qwen3-4B and Llama-3.2-3B-Instruct fail on its self-evaluation query (Section B). To validate this design choice, we vary the failure-labeling rule while keeping the rest of the pipeline and the compute budget fixed.

Table 17 shows that the conjunction criterion yields the cleanest learning signal. Tools on which two heterogeneous architectures independently fail are more likely to encode genuine structural difficulty rather than model-specific quirks. In contrast, the disjunction variant—specifically the “either-fails” rule, which one might assume is the most informative cross-architecture criterion—yields poorer performance. This is because most single-model failures stem from peculiarities in formatting or tokenization that do not generalize across different models. Random sampling, which lacks any difficulty signal, performs worst by a large margin (-6.67 points). The choice of Qwen3-4B and Llama-3.2-3B-Instruct as dual evaluators is both efficiency-driven (small models enable cheap self-evaluation over 2,095 tools) and diversity-driven (independently developed architectures, tokenizers, and pre-training corpora).

K Cross-Family Generalization Beyond Qwen and Llama

Table 16 demonstrates that HardGen-RL transfers to the Llama family. To further address whether the gains generalize beyond the Qwen and Llama initialization families, we additionally

Model	Base	+HardGen-SFT	Δ
Mistral-Small-2506	61.22	72.43	+11.21
Gemma-3-27b-it	57.50	68.13	+10.63
Qwen3-4B (reference)	62.13	73.70	+11.57

Table 18: **Cross-family generalization of HardGen-synthesized data.** BFCLv3 Overall accuracy before and after SFT on the 27,000 HardGen corpus. Improvements are consistent across distinct model families with independently developed tokenizers, pre-training corpora, and architectures.

evaluate HardGen-synthesized data on two additional model families, Mistral (Mistral AI, 2025) and Gemma (Gemma Team et al., 2025), under the same SFT configuration without any pipeline changes.

Table 18 shows that HardGen produces consistent double-digit gains in Mistral (+11.21) and Gemma (+10.63), closely matching the +11.57 gain observed on Qwen3-4B. Combined with Llama results in Table 16 (+18.78 on Llama-3.1-8B and +19.24 on Llama-3.2-3B), the improvements span four independently developed model families. This supports the view that the Failure API Set captures structurally challenging tools, long dependency chains, a high branching factor, and many latent variables (Table 2) rather than model-specific idiosyncratic vulnerabilities of any single training family.

L Supplementary Case

This section presents a case study of HardGen, detailing the synthesis process along with a complete trajectory.

Hard Query Construction with Logical Jump.

Figure 8 illustrates the construction of hard queries with logical jumps. Unlike the previous methods, which explicitly instruct the model to first check zip codes before purchasing tickets, the hard query directly requests ticket purchase between city names without specifying intermediate steps. This formulation requires the model to autonomously infer the necessary tool chain—recognizing that city names must first be converted to zip codes via `get_zipcode` before invoking `buy_tickets`. The advanced tool `buy_tickets_adv` demonstrates the desired end-to-end capability of purchasing tickets directly from city names, representing the ideal abstraction, a single function that internally handles the multi-step process.

Model Reasoning for Hard Query. Figure 9 illustrates the model’s reasoning process for a hard query. Given the task to purchase tickets between two cities by name, the model correctly recognizes the mismatch between the query (city names) and available tool inputs (zip codes). In its internal reasoning, the model analyzes the available tools, identifies the dependency structure, and decomposes the task into three steps: 1) retrieve the zip code for Rivermist, 2) retrieve the zip code for Stonebrook, and 3) purchase tickets using both zip codes. The model then executes this planned sequence through appropriate tool calls, successfully completing the multi-step task without explicit instructions.

Complete Trajectory. Figures 10 and 11 illustrate a complete multi-turn trajectory demonstrating sequential reasoning and context retention. In Turn 1, the model is asked to determine the working directory and search for all files recursively. The model correctly selects `pwd` and `find` tools, retrieves the directory structure, and summarizes the results. In Turn 2, building on the previous context, the user requests file contents from subdirectories identified in Turn 1. The model recognizes tool constraints—that `cat` and `tail` only operate within the current directory—and constructs a four-step plan involving directory navigation (`cd`) to access the required files. This example demonstrates the model’s ability to maintain context across turns and adapt its strategy based on tool limitations.

M Prompts

This section provides the prompts used in the HardGen pipeline. Our framework employs four specialized agents, each with carefully designed prompts to fulfill specific roles in the data generation process. The Tool Maker (Figure 12) synthesizes advanced tools from multi-step tool-executions by abstracting primitive tool sequences into high-level operations. The Hard Query Generator (Figure 13) creates challenging queries that require implicit logical bridging, forcing models to infer necessary intermediate steps rather than following explicit instructions. During the reasoning refinement phase, the Reasoner (Figure 14 and Figure 16) attempts to solve hard queries with hints from advanced tool descriptions, while the Verifier (Figure 15) analyzes incorrect attempts and provides targeted corrective feedback without revealing answers directly.

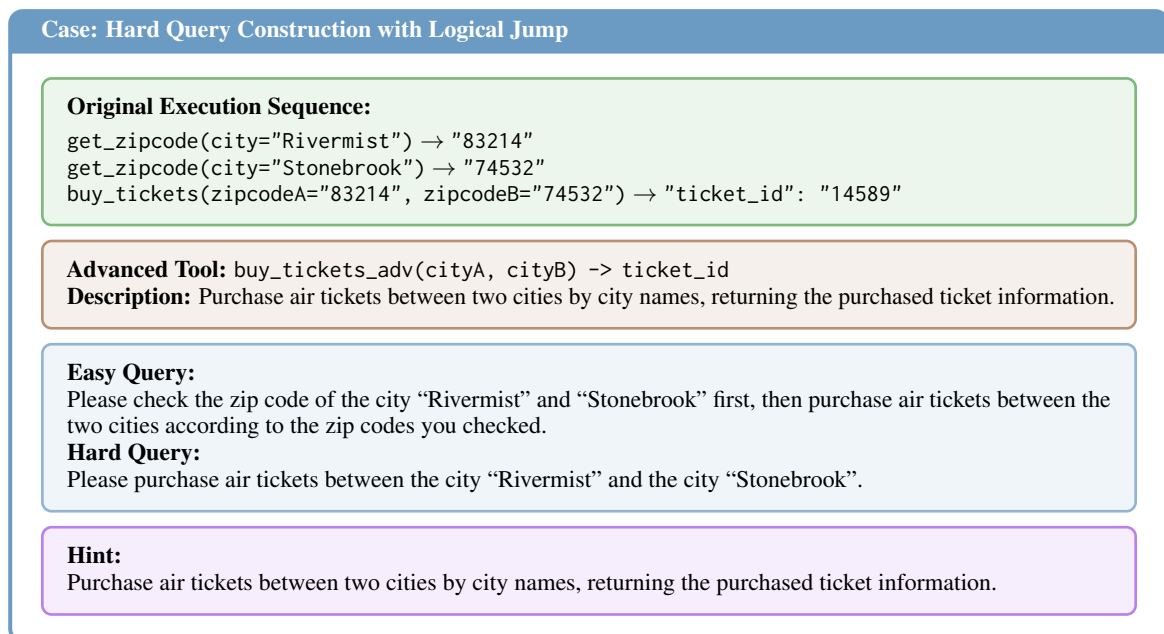


Figure 8: **Case Study: Hard Query Construction.** Illustration of the data synthesis process where an original execution sequence is abstracted into an "Advanced Tool" to generate a Hard Query that omits explicit intermediate steps (zipcode lookup).



Figure 9: **Case Study: Reasoning process for a Hard Query.** Visualization of the model’s Chain-of-Thought (CoT) process, showing how it decomposes a high-level query into necessary primitive tool calls (*e.g.*, retrieving zip codes before purchase).

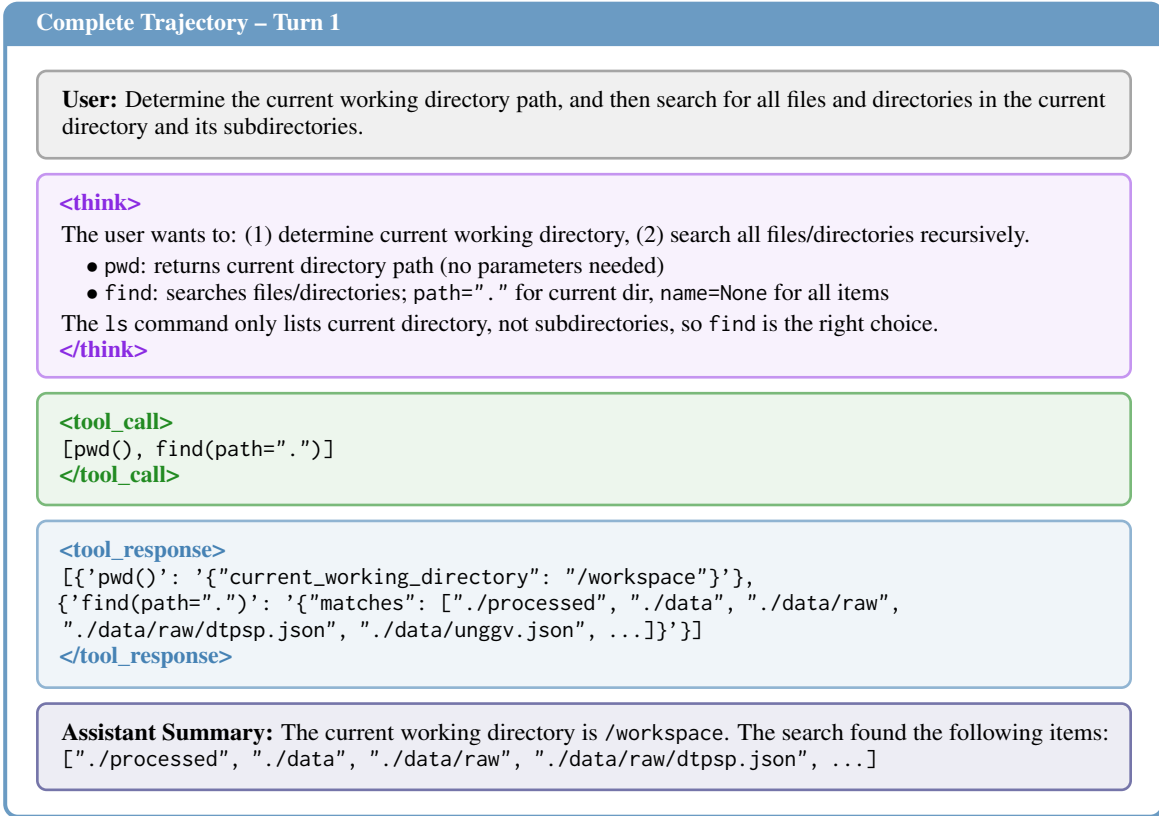


Figure 10: **Qualitative Example: Multi-turn Directory Exploration (Turn 1).** A trajectory step where the model correctly selects the pwd and find commands to satisfy a recursive search requirement.

Complete Trajectory – Turn 2

User: Display the contents of the file named `unggv.json` in the data directory, and then display the last 7 lines of the file named `dtsp.json` in the raw subdirectory.

<think>

Current directory: `/workspace`. Need to access files in subdirectories.

Constraints: `cat` and `tail` only work on files in the current directory.

Plan:

- (1) `cd data` → change to data directory
- (2) `cat ungv.json` → display file contents
- (3) `cd raw` → change to raw subdirectory
- (4) `tail dtsp.json lines=7` → display last 7 lines

</think>

<tool_call>

```
[cd(folder="data"), cat(file_name="unggv.json"), cd(folder="raw"),
tail(file_name="dtsp.json", lines=7)]
```

</tool_call>

<tool_response>

```
[{'cd(folder="data")': '{"current_working_directory": "data"}'},
{'cat(file_name="unggv.json")': '{"file_content": {"data": {"id": "94"}}}'},
{'cd(folder="raw")': '{"current_working_directory": "raw"}'},
{'tail(...)': '{"last_lines": {"data": {"id": "27"}}}'}]
```

</tool_response>

Assistant Summary:

The contents of `"unggv.json"`: `{"data": {"id": "94"}}`

The last 7 lines of `"dtsp.json"`: `{"data": {"id": "27"}}`

Figure 11: **Qualitative Example: Context-Dependent File Retrieval (Turn 2).** A subsequent trajectory step demonstrating context retention, where the model navigates directories (`cd`) to access files based on tool constraints.

Tool Maker Agent Prompt

System Prompt:

You are a Tool Maker agent responsible for synthesizing advanced tools from execution sequences.

Given the following execution sequence:

{execution_sequence}

Execution Sequence Details:

- Tool Call 1: {tool_1}({params_1}) → {result_1}
- Tool Call 2: {tool_2}({params_2}) → {result_2}
- ...
- Tool Call M: {tool_M}({params_M}) → {result_M}

Your task:

Create an advanced tool that abstracts this multi-step execution sequence into a single high-level operation.

Requirements:

1. The advanced tool should encapsulate ALL steps in the execution sequence
2. It should have a clear, intuitive name describing the end-to-end functionality
3. Parameters should be high-level inputs (not intermediate values)
4. The description should explain the overall goal, not individual steps

Output format:

```
{
  "advanced_tool_name": "descriptive_name",
  "parameters":
  [
    {"name": "param1", "type": "type",
     "description": "what it represents"}
  ]
}
```

Example:

Original execution sequence:

```
get_zipcode(city="A") → zipA
get_zipcode(city="B") → zipB
buy_tickets(zipA, zipB) → ticket_id
```

Advanced tool created:

```
buy_tickets_adv(cityA, cityB) → ticket_id
```

Description: "Purchase air tickets between two cities by city names, returning the purchased ticket information."

The advanced tool hides the intermediate step of zipcode lookup, allowing users to work directly with high-level city names.

Figure 12: **Prompt template for the Tool Maker Agent.** Instructions used to synthesize advanced tools from multi-step execution sequences.

Hard Query Generator Prompt

System Prompt:
You are a Hard Query Generator agent.

Given the following advanced tool:
{advanced_tool_specification}

Advanced Tool: {tool_name}
Parameters: {parameters}
Description: {description}

Your task:
Generate a challenging query that:

1. Requires the use of this advanced tool’s functionality
2. Does NOT explicitly mention the intermediate steps
3. Forces the model to perform implicit logical bridging
4. Uses high-level language that matches the advanced tool’s abstraction level

Requirements for the hard query:

- ✗ Do NOT say “first do X, then do Y, then do Z”
- ✗ Do NOT mention the primitive tools by name
- ✓ DO frame the request at the level of the end goal
- ✓ The query should appear simple but require complex multi-step reasoning

Example Comparison:

BAD (easy) query:
“Please check the zip code of Rivermist and Stonebrook first, then purchase air tickets between the two cities according to the zip codes you checked.”

GOOD (hard) query:
“Please purchase air tickets between the city Rivermist and the city Stonebrook.”
The GOOD query requires the model to infer: need zipcodes → must call get_zipcode twice → then call buy_tickets

Figure 13: **Prompt template for the Hard Query Generator.** Instructions used to create challenging queries that necessitate implicit logical bridging based on advanced tool specifications.

Reasoning Agent Prompt (Initial Attempt)

System Prompt:
You are a reasoning agent tasked with solving tool-use queries.

Query: {hard_query}

Available Tools:
{tool_descriptions}

Hint (Advanced Tool Capability):
{advanced_tool_description}

Note: The hint describes a high-level capability. You must use the available primitive tools to achieve this capability.

Your task:

1. Analyze the query and understand what needs to be accomplished
2. Identify which tools are needed and in what order
3. Plan the execution sequence considering tool dependencies
4. Generate appropriate function calls with correct parameters

Output format:

```
<think>
[Your reasoning process]
</think>

<tool_call>
[generated_function_call_1,
 generated_function_call_2, ...]
</tool_call>
```

Remember:

- Think carefully about implicit dependencies
- Check if tool prerequisites are satisfied
- Ensure parameters are correctly mapped between tools

Figure 14: **Prompt template for the Reasoner Agent (Initial Attempt).** Instructions for solving tool-use queries using hints from advanced tool descriptions.

Verifier Agent Prompt

System Prompt:
You are a Verifier agent responsible for analyzing incorrect function calls and providing corrective feedback.

Query: {hard_query}

Model's Attempt: {incorrect_function_call}

Ground Truth: {correct_function_call}

Execution Result: {execution_result}

Your task: Compare the model's attempt with the ground truth and identify:

- Error Type:** Wrong tool selection, missing tool calls, incorrect parameters, wrong order, type mismatch, or missing dependencies
- Root Cause:** Why did the model make this error? What logical step was missed? What dependency was not recognized?
- Corrective Hint:** Specific guidance to fix this error without giving the answer directly

Output format:

```
{
  "error_type": "specific error category",
  "error_location": "which tool call or parameter",
  "root_cause": "explanation of why error occurred",
  "corrective_hint": "targeted guidance without
                    revealing the answer",
  "should_reconsider": ["aspect1", "aspect2"]
}
```

Example:
Error: Model called buy_tickets directly without getting zipcodes first
Hint: "You attempted to purchase tickets directly, but the buy_tickets tool requires zipcode parameters, not city names. Consider what information you need to obtain first before making the purchase."

Remember:

- Do NOT give the answer directly—guide the model to discover the correct approach
- Focus on the logical reasoning gap, not just the technical error
- Hints should be specific enough to be helpful but general enough to require thinking

Figure 15: **Prompt template for the Verifier Agent.** Instructions for analyzing incorrect function calls and providing targeted corrective feedback.

Reasoner Agent Prompt (Refinement Iteration)

System Prompt:
You are refining your previous attempt based on feedback.

Original Query: {hard_query}
Available Tools: {tool_descriptions}
Hint: {advanced_tool_description}

Your Previous Attempt:

```
<think>
{previous_reasoning}
</think>
<tool_call>
{previous_function_call}
</tool_call>
```

Feedback from Verifier: {error_diagnosis_and_hint}

Your task:

1. Carefully read the feedback
2. Identify what went wrong in your previous attempt
3. Revise your reasoning process
4. Generate a corrected function call sequence

Output format:

```
<think>
[Updated reasoning addressing the feedback]
- What was wrong in my previous attempt?
- What does the feedback tell me?
- How should I adjust my approach?
- New execution plan
</think>

<tool_call>
[corrected_function_call_1, corrected_function_call_2, ...]
</tool_call>
```

Remember:

- The feedback is designed to guide you toward the correct solution
- Focus on understanding WHY your previous attempt failed
- Each refinement should address the specific issues mentioned in feedback
- Maintain the same output format: <think> followed by <tool_call>

Figure 16: **Prompt template for the Reasoner Agent (Refinement Iteration).** Instructions for revising reasoning and function calls based on feedback from the Verifier.