

From Failure to Mastery: Generating Hard Samples for Tool-use Agents

Anonymous ACL submission

Abstract

The advancement of LLM agents with tool-use capabilities requires diverse and complex training corpora. Existing data generation methods, which predominantly follow a 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 with verifiable reasoning. *Firstly*, HardGen establishes a dynamic API Graph built upon agent failure cases, from which it samples to synthesize hard traces. *Secondly*, these traces serve as conditional priors to guide the instantiation of modular, abstract advanced tools, which are subsequently leveraged to formulate hard queries. *Finally*, the advanced tools and hard queries enable the generation of verifiable complex Chain-of-Thought (CoT), with a closed-loop evaluation feedback steering the continuous refinement of the process. 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 dataset 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., 2025a; Shen, 2024; Wang et al., 2025b). As agents are deployed in increasingly sophisticated scenarios—from enterprise workflow automation to intricate data analysis—the demand for high-quality, diverse, and complex training

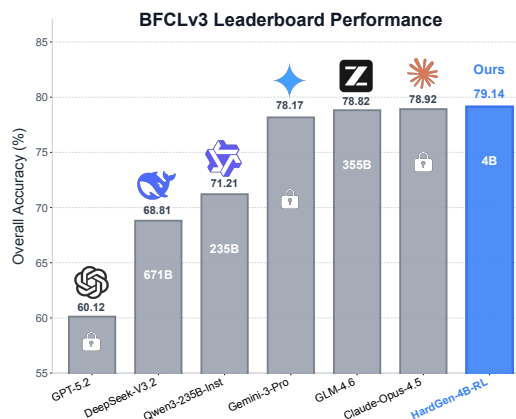


Figure 1: Performance comparison on the BFCLv3 Leaderboard (Patil et al.). A Qwen3-4B (Yang et al., 2025) model trained with our curated dataset, denoted as **HardGen-4B-RL**, consistently outperforms leading open-source and closed-source models.

corpora has become the primary bottleneck limiting their advancement (Liu et al., 2024; Zeng et al., 2025b). While recent works have made strides in synthesizing diverse datasets, existing data generation pipelines predominantly adhere to a paradigm of random sampling and shallow generation—randomly selecting API combinations from static tool pools and generating 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 *implicit logical dependencies* 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., 2025a; Yin et al., 2025).

¹We use tools and APIs interchangeably in this paper.

To bridge this gap, we introduce **HardGen**, an 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 *learn from failure and evolve through feedback*. 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 synthesize *hard traces* that target the specific weaknesses of models. Next, we utilize these traces as conditional priors to guide the instantiation of modular, abstract advanced tools. 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 complex 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. With HardGen, we construct a comprehensive dataset containing 27,000 trajectories with 2,095 APIs from real environments (see Section 3.5). Our dataset encompasses a rich spectrum of complex interaction scenarios, particularly hard queries necessitating implicit logical bridging, where agents must autonomously infer tool dependencies and perform multi-step reasoning without explicit guidance. This large-scale, high-fidelity corpus is designed to overcome the complexity bottleneck in tool-use agent research, providing the community with a verifiable and rigorous foundation for training models capable of mastering deep logical dependencies.

Remarkable Results. We evaluate our pipeline by performing SFT or RL training on the Qwen3-4B (Yang et al., 2025) model using the HardGen curated dataset. The resulting models, denoted as **HardGen-4B-SFT** and **HardGen-4B-RL**, are evaluated against several leading proprietary models on the challenging BFCLv3 (Patil et al.) and other benchmarks. Despite its compact **4B** scale,

HardGen-4B-RL attains an overall accuracy of **79.14%**, setting a new state-of-the-art record of its size (see Table 2). For example, despite the massive disparity in model scale, HardGen-4B-RL achieves superior advantages over the latest strong competitors, surpassing GPT-5.2 (OpenAI, 2025) by **19.02%**, DeepSeek-V3.2 (Liu et al., 2025) by **10.33%** and Grok-4.1-Fast (xAI, 2025) by **3.94%** (see Figure 1). Crucially, these gains generalize to **held-out** benchmark (BFCLv4 (Patil et al.)), validating that the model learns robust agentic reasoning patterns rather than memorizing templates.

Our contributions are summarized as follows:

- We propose HardGen, a novel pipeline targeting generating hard function-calling data 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 LLM. Empowering LLMs to interact with external APIs is pivotal for the realization of autonomous agentic systems (Wang et al., 2025a; Hao et al., 2025b). By bridging the gap between static knowledge and dynamic execution, this paradigm enables models to interrogate databases, synthesize code, and manipulate digital interfaces (Jimenez et al., 2023; Mohammadjafari et al., 2024; Xie et al., 2024; Gao et al.). Yet, the transition from isolated function invocations to coherent, multi-turn tool orchestration presents a substantial barrier (Guo et al., 2025; Chen et al., 2025; Patil et al.). 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; Yin et al., 2025).

Data Synthesis for Tool-use Training. Current paradigms in tool-use data generation focus pre-

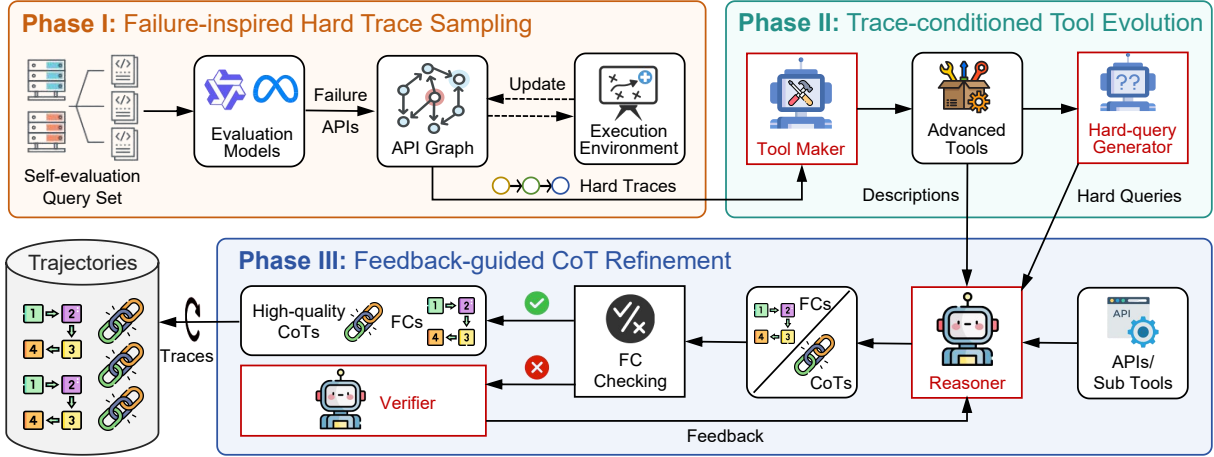


Figure 2: **Overview of the HardGen framework.** The pipeline operates in three phases: **I) Failure-inspired Hard Trace Sampling** to identify error-prone tool dependencies and construct hard tool traces; **II) Trace-conditioned Tool Evolution** to synthesize advanced tools and hard queries based on the constructed hard traces; and **III) Feedback-guided CoT Refinement** to verify and optimize reasoning chains through a closed-loop mechanism.

163 dominantly on expanding breadth and ensuring ex-
 164 executability (Yin et al., 2025; Acikgoz et al., 2025),
 165 typically via synthesis from fixed toolsets or multi-
 166 agent simulations (Prabhakar et al., 2025; Yin
 167 et al., 2025). Although these methods scale dataset
 168 size (Xu et al., 2025), they suffer from a critical lim-
 169 itation: the difficulty of the generated trajectories
 170 is often artificial—derived from explicit structural
 171 constraints—rather than reflecting the intrinsic rea-
 172 soning hurdles and implicit dependencies charac-
 173 teristic of authentic agentic workflows (Zeng et al.,
 174 2025b; Lam et al., 2024; Xu et al., 2025). In con-
 175 trast, our HardGen aims to generate complex, verifi-
 176 able tool-use trajectories that deliver grounded rea-
 177 soning for SFT and precise rewards for RL (Qian
 178 et al., 2025; Hao et al., 2025a), thereby driving
 179 superior performance in agentic tasks.

180 3 The Proposed HardGen Pipeline

181 In this section, we present HardGen, an automatic
 182 agentic pipeline designed to generate hard tool-use
 183 training samples with verifiable reasoning.

184 3.1 Overview

185 As the overview illustrated in Figure 2 and nota-
 186 tions shown in Table 1, HardGen consists of
 187 three coordinated phases: **First**, we identify error-
 188 prone function calls via model evaluation on a self-
 189 evaluation query set. These “failure APIs” are then
 190 structured into a dynamic API Graph, which is
 191 iteratively updated through interactions with an
 192 execution environment to capture complex tool de-
 193 pendencies. **Second**, we sequentially feed the gen-

Symbol	Concept
\mathcal{S}	Environment Simulation Space
\mathcal{T}	Failure API Set
$\mathcal{G} = (\mathcal{T}, \mathcal{D}, \mathcal{P})$	API Graph (Failure API Set \mathcal{T} , De- pendencies \mathcal{D} , Parameters \mathcal{P})
A_T, A_Q, A_R, A_V	Agents: Tool Maker, Hard-query Generator, Reasoner, Verifier
$T_a \in \mathcal{T}$	Selected Tool
Q_{hard}	Hard Query
T_{adv}	Advanced Tool
M	Number of Tool Calls Per Trace
N	Number of Turns Per Trajectory

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

194 erated tool traces from the API Graph into a *Tool*
 195 *Maker* and a *Hard-query Generator*, which evolves
 196 simple functions into advanced tools and synthesize
 197 corresponding high-complexity queries. **Third**, we
 198 employ a *Reasoner-Verifier* loop to execute these
 199 hard queries, using environment feedback to rig-
 200 orously filter and refine function calls and CoTs. By
 201 repeating this three-phase process, our HardGen
 202 generates reliable and complex multi-turn trajec-
 203 tories, which contain hard queries, primitive tools,
 204 verified CoTs and function calls.

205 3.2 Phase I: Failure-inspired Hard Trace 206 Sampling

207 Rather than uniformly sampling tool combinations,
 208 this phase leverages a failure-driven self-evaluation
 209 to surface tools and dependencies that the current
 210 model struggles with. The resulting API Graph
 211 serves as a structured representation of failure-
 212 prone tools and their latent interaction patterns,

from which a series of hard and valuable tool traces can be sampled.

Failure-driven Self-evaluation and Graph Construction. We deploy a large-scale API environment comprising 2,095 tools to perform model self-evaluation. For each tool, HardGen generates an execution trace 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 (Dubey et al., 2024). 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 are incrementally incorporated into both the Failure API Set \mathcal{T} and API Graph \mathcal{G} , enabling systematic capture of challenging tools and their dependencies. The graph construction and its update are shown in the Appendix G. Despite performing evaluation and construction solely on Qwen3 models, our approach demonstrates robust performance across heterogeneous architectures including Llama-3 (see Table 12) and Qwen2.5 (see Table 3), validating the strong generalization capability of the HardGen synthesized data.

Legality-constrained Sampling. To guarantee execution validity, tool selection is subject to a strict dependency constraint: a tool T_a is callable if and only if all its prerequisite dependency tools (\mathcal{D}_{T_a}) have been executed previously, denoted by $\mathcal{T}_{\text{called}}$. This legality condition is formalized as

$$I(T_a, \mathcal{T}_{\text{called}}) = \mathbf{1}_{\{\mathcal{D}_{T_a} \subseteq \mathcal{T}_{\text{called}}\}}, \quad (1)$$

where $\mathbf{1}_{\{\cdot\}}$ is the indicator function.

Hard Trace Sampling. To reach the selected target tool T_a while satisfying its prerequisite constraints, we design a *Sampler* that explicitly biases trace construction toward T_a . Specifically, the Sampler employs a greedy heuristic over the set of legal tools $\mathcal{T}_{\text{legal}}$, prioritizing tools that minimize the graph distance $\text{dist}(\cdot, \cdot)$ to T_a . Here, $\text{dist}(T_k, T_a)$ denotes the length of the shortest path from T_k to T_a in the API graph \mathcal{G} . The resulting sampling policy is formally defined as:

$$T_s = \text{Sampler}(T_a, \mathcal{T}_{\text{called}}) = \begin{cases} \text{rand}(\mathcal{T}_{\text{legal}}), & T_a \in \mathcal{T}_{\text{called}}, \\ T_a, I(T_a, \mathcal{T}_{\text{called}}) = 1 \wedge T_a \notin \mathcal{T}_{\text{called}}, \\ \arg \min_{T_k} \text{dist}(T_k, T_a), & \text{otherwise.} \end{cases} \quad (2)$$

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 set of executed tools $\mathcal{T}_{\text{called}}$, and the dynamic API graph \mathcal{G} . The resulting execution sequence is recorded as an executable hard trace $\Gamma_i = (C_1, E_1, C_2, E_2, \dots, C_M, E_M)$, which serves as a failure-aware prior for subsequent phases.

3.3 Phase II: Trace-conditioned Tool Evolution

The hard trace Γ_i generated in Phase I guarantees correct tool execution and faithfully captures the underlying operational sequence. However, a central challenge in training robust tool-use agents lies in enabling *implicit logical bridging*—the ability to autonomously infer the necessary intermediate steps when faced with hard queries that do not explicitly specify the full tool chain. Directly generating queries from execution traces, where all intermediate steps are explicitly enumerated, fails to cultivate this capability, as models can simply rely on pattern matching rather than learning to bridge logical gaps. To address this challenge, Phase II reinterprets the trace via a two-stage evolution process: it first abstracts the multi-step execution into a unified high-level operation, and then constructs a challenging hard query that explicitly requires this abstraction.

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

$$T_{\text{adv}} = A_T(\Gamma_i), \quad (3)$$

where A_T abstracts the multi-step execution trace 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 synthesized advanced tool T_{adv} , the *Hard-query Generator* (A_Q) generates a challenging query Q_{hard} that explicitly requires the use of T_{adv} for resolution. Formally, the process is expressed as:

$$Q_{\text{hard}} = A_Q(T_{\text{adv}}). \quad (4)$$

3.4 Phase III: Feedback-guided CoT Refinement

While the hard query Q_{hard} constructed in Phase II effectively challenges the model with implicit logical dependencies, it also amplifies the difficulty of

generating correct and coherent Chain-of-Thought (CoT) reasoning. To mitigate this issue, Phase III implements an iterative refinement mechanism that progressively corrects flawed reasoning via feedback-driven prompting, guiding the CoT from erroneous states toward correct 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. When the current attempt fails (i.e., $\text{FC}_i^{(k)} \neq C_i$), the *Verifier* A_V analyzes the discrepancy between the incorrect function call and the ground truth. The Verifier identifies the specific error and generates corrective hint feedback $\text{Error}_i^{(k)}$ that guides correction without exposing the answer.

CoT Refinement. For step $i \in \{1, \dots, M\}$ at its k -th attempt, we incorporate the corrective hint $\text{Error}_i^{(k)}$ into the current prompt through concatenation:

$$\text{Prompt}_i^{(k+1)} = \text{Concat}(\text{Prompt}_i^{(k)}, \text{Error}_i^{(k)}). \quad (5)$$

With this augmented prompt, A_R is re-invoked to produce a refined solution, yielding an updated reasoning process $\text{CoT}_i^{(k+1)}$ and function call $\text{FC}_i^{(k+1)}$. This iterative refinement continues until either the function call is correct or the maximum number of attempts K_{max} is reached.

Upon successfully generating the correct function call C_i at step i on the k -th attempt, the environment execution feedback E_i is incorporated into the prompt for the subsequent $i+1$ step via concatenation: $\text{Prompt}_{i+1}^{(1)} = \text{Concat}(\text{Prompt}_i^{(k)}, C_i, E_i)$. This design ensures that subsequent reasoning is explicitly conditioned on the complete execution history accumulated up to the current step. A trace is retained only if all M function calls are correctly generated across the entire sequence.

3.5 Generated Dataset

We deploy HardGen on a large-scale real-world API environment comprising 2,095 tools, of which 1,204 are identified as Failure Tools through systematic Self-evaluation. Training trajectories are synthesized using the HardGen pipeline, with

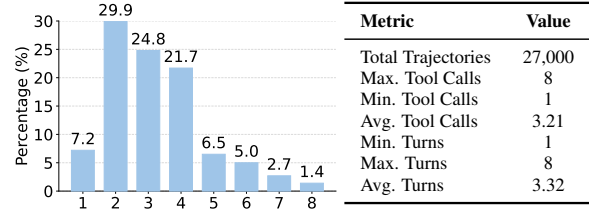


Figure 3: **Statistics and distribution of the generated dataset.** (Top) Histogram of tool calls per trajectory (left) and key dataset metrics (right). (Bottom) Distribution of API domains (left), the proportion of single-turn and multi-turn trajectories (right).

Qwen3-30B-A3B-Thinking (Yang et al., 2025) serving as the backbone of the four agents. The system employs a maximum attempt limit of $K_{\text{max}} = 3$, which we find offers a favorable balance between trajectory quality and computational cost. As shown in Figure 3, the resulting dataset comprises 27,000 high-quality trajectories that exhibit substantial diversity across multiple dimensions. First, in terms of task complexity, trajectories span 1 to 8 tool calls (with an average of 3.21), and 62.1% of them contain three or more tool calls within a single trace. Second, regarding API coverage, the dataset spans diverse domains including Finance (20.4%), Game (9.5%), Tools (16.8%), and Data (19.5%), ensuring broad applicability. Third, the trajectories capture rich reasoning patterns, from straightforward single-turn executions (36.3%) to complex multi-turn scenarios (63.7%) requiring sophisticated inference capabilities. A supplementary trajectory case and the prompts of corresponding agents are shown in Appendix H and Appendix I.

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 employ Qwen3-4B as our main baseline model (Yang et al., 2025). To

Model	Parameter Counts	Single-Turn			Multi-Turn					Overall
		Non-Live	Live	Subset Overall	Base	Miss Func	Miss Param	Long Context	Subset Overall	
<i>Closed-source</i>										
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
Claude-Haiku-4-5-20251001	-	86.50	78.68	82.59	63.50	42.50	52.50	56.00	53.63	72.93
Gemini-3-Pro-Preview	-	90.65	83.12	86.89	64.50	60.00	54.50	64.00	60.75	78.17
Gemini-2.5-Flash	-	84.96	74.39	79.68	41.50	36.00	32.00	35.50	36.25	65.20
Grok-4-1-fast-reasoning	-	88.27	78.46	83.37	70.50	59.50	43.00	62.50	58.88	75.20
Grok-4-1-fast-non-reasoning	-	88.13	77.94	83.04	58.00	39.50	37.50	52.00	46.75	70.94
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
<i>Open-source</i>										
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 2: **Performance on BFCLv3** (last updated on 2025-12-16). All metrics are calculated using the official script and reported in terms of Accuracy (%).

validate the generalizability of our pipeline, we additionally conduct experiment with Qwen2.5-7B-Instruct (Team et al., 2024), Llama-3-3B/8B-Instruct (Dubey et al., 2024), and Qwen3-0.6B/1.7B. We split the 27,000 trajectories at each assistant response and train the model to generate only the current-turn response. We perform SFT with LLaMA-Factory (Zheng et al., 2024) and RL with Verl (Sheng et al., 2024). Implementations are detailed in Appendix D.

Benchmarks. We evaluate our models on BFCLv3 (Patil et al.), API-Bank (Li et al., 2023), and ACEBench (Chen et al., 2025), covering both single-turn and multi-turn tool-calling scenarios (see Appendix D). To evaluate out-of-distribution generalization and assess how the proposed training data impacts specific agentic capabilities, we include BFCLv4 as a **held-out** benchmark (Patil et al.). BFCLv4 offers a comprehensive evaluation of agentic behaviors in Web Search and Memory scenarios, with details provided in Appendix C.

4.2 Experimental Results

Results on BFCLv3. As shown in Table 2, on the BFCLv3 benchmark (Patil et al.), models trained with HardGen yield notable improvements on Qwen3-4B, raising the multi-turn score from

22.13 to 49.65 (**+27.52**) after SFT and to 63.13 (**+41.01**) after RL. Despite its 4B parameter size, the HardGen RL-trained model surpasses strong open-source models (e.g., Kimi-K2-Inst (Team et al., 2025), DeepSeek-V3.2 (Liu et al., 2025)) and leading closed-source models (e.g., GPT-5.2 (OpenAI, 2025), Gemini-3-Pro (Google, 2025), Claude-Opus-4.5 (Anthropic, 2025)), setting a new state-of-the-art record of its size. In addition, models trained with HardGen demonstrate balanced performance across all sub-metrics, indicating strong generalization and stability. A particularly compelling result is that both HardGen-4B-SFT and HardGen-4B-RL outperform Qwen3-30B-A3B-Thinking (Yang et al., 2025) by a substantial margin across all sub-metrics. This result validates that our failure-driven sampling, trace-conditioned tool evolution, and feedback-guided refinement jointly form a virtuous cycle of capability amplification, effectively transcending the inherent limitations of the generator model. To further assess the generalizability of HardGen-synthesized data, we report results on the Llama model family in Appendix F, demonstrating consistent improvements across diverse model architectures.

Results on APIBank and ACEBench. Figure 4(a) and Figure 4(b) present the performance of our

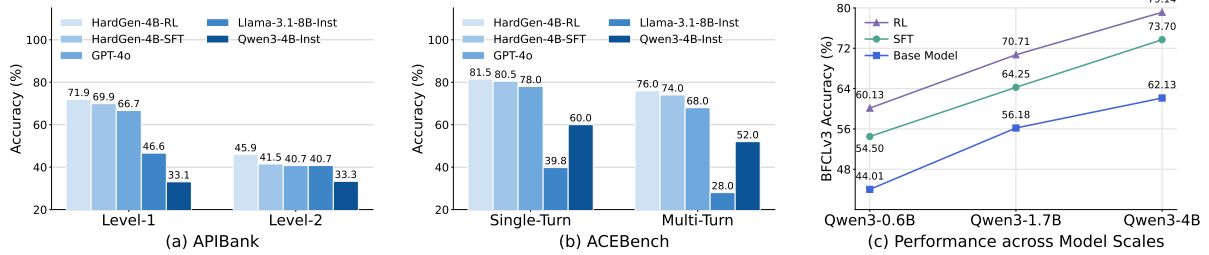


Figure 4: **Additional evaluations.** (a) (b) Comparison of different models on APIBank and ACEBench. (c) Scaling trends of HardGen-SFT and HardGen-RL on BFCLv3 across model parameters.

Model	Single-Turn	Multi-Turn	Overall
Qwen2.5-7B-Instruct (Base Model)	77.30	7.62	54.07
TOUCAN (Xu et al., 2025)	76.51	22.62	58.55
MAGNET (Yin et al., 2025)	80.49	21.12	60.70
ToolACE-MT (Zeng et al., 2025b)	80.51	27.57	62.86
+ HardGen-SFT	83.99	40.75	69.58
Δ	+6.69	+33.13	+15.51

Table 3: **Comparison with state-of-the-art data synthesis pipelines on BFCLv3.** The baseline model is Qwen2.5-7B-Instruct.

models on two additional benchmarks, APIBank and ACEBench. On APIBank (Li et al., 2023), our models achieve top-tier Level-1 accuracies of 71.9 and 69.9, clearly outperforming GPT-4o (Hurst et al., 2024), which attains 66.7. For the more challenging Level-2 tasks, our models continue to demonstrate strong performance, yielding improvements of **+12.6** and **+8.2** percentage points over the base model (33.3), respectively. Evaluation on ACEBench (Chen et al., 2025) further confirms robust generalization under both training configurations. On the single-turn subset, our models reach accuracies of 81.5 and 80.5, surpassing both GPT-4o (78.0) and Llama-3.1-8B-Instruct (39.8) by substantial margins. This advantage is more pronounced in the multi-turn setting, where our models achieve scores of 76.0 and 74.0, exceeding Llama-3.1-8B-Instruct (28.0) by **48.0** and **46.0** percentage points, respectively. Overall, these results provide compelling evidence that models trained on HardGen synthesized dataset exhibit strong and consistent tool-use capabilities across diverse benchmarks and interaction settings.

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 3. While prior methods such as MAGNET (Yin et al., 2025), TOUCAN (Xu et al., 2025), and ToolACE-MT (Zeng et al., 2025b) offer incremental gains over the base

Hard Query	Qwen3-4B		Llama-3-3B	
	Single-Turn	Multi-Turn	Single-Turn	Multi-Turn
\times	84.17	54.38	79.08	32.23
\checkmark	87.14	63.13	84.05	40.13

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

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 Ablation Study

In this section, we present ablative results 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 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 **+13.74** points (rising from 34.93 to 48.67) via RL, whereas Qwen3-4B realizes a more substantial improvement of **+16.93** points (60.21 to 77.14). Furthermore, RL consistently outperforms SFT across all model sizes, with the performance gap widening as the base model scales up. 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

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

the constructed hard queries on data quality, we perform ablation studies by conducting RL training on Qwen3-4B and Llama-3-3B. As shown in Table 4, 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-3-3B 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 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 5, 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. Since the subjective nature of *difficulty* judgments, we further complement the automated evaluation with a human anno-

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 6: **Impact of feedback-guided CoT refinement.** The three baseline models are trained with RL using data with or without Verifier feedback.

tation study in Appendix E, using the same rubric for assessing logical-jump difficulty. The results show a high level of agreement between the automated judge and human annotators, supporting the conclusion that T_{adv} reliably increases implicit logical-bridging difficulty.

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 maximum number of refinement iterations. As reported in Table 6, 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 more 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 A.

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 held-out benchmarks, paving the way for developing more robust models capable of complex tool-use and agentic ability.

577 Limitations

578 While HardGen demonstrates strong performance
579 across multiple benchmarks and model architec-
580 tures, several limitations merit consideration. 1)
581 Although our evaluation covers 2,095 tools across
582 diverse domains, the extent to which HardGen gen-
583 eralizes to entirely new API ecosystems or spe-
584 cialized domains remains to be fully explored. 2)
585 Our approach requires executable environments for
586 verification, which may not be feasible for prop-
587 rietary APIs or tools with complex external depen-
588 dencies. 3) The abstraction quality of advanced
589 tools relies on the Tool Maker’s ability to correctly
590 identify high-level operations from primitive tool
591 sequences, which may occasionally produce sub-
592 optimal abstractions for highly irregular or domain-
593 specific tool chains.

594 References

595 Emre Can Acikgoz, Jeremiah Greer, Akul Datta,
596 Ze Yang, William Zeng, Oussama Elachqar, Em-
597 manouil Koukoumidis, Dilek Hakkani-Tur, and
598 Gokhan Tur. 2025. Can a single model master both
599 multi-turn conversations and tool use? *coalm: A*
600 *unified conversational agentic language model*. In
601 *Proceedings of the 63rd Annual Meeting of the As-*
602 *sociation for Computational Linguistics (Volume 1:*
603 *Long Papers)*, pages 12370–12390.

604 Anthropic. 2025. System card: Claude opus 4.5. Tech-
605 nical report, Anthropic.

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

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

616 Shen Gao, Yuntao Wen, Minghang Zhu, Jianing Wei,
617 Yuhan Cheng, Qunzi Zhang, and Shuo Shang.
618 Simulating financial market via large language
619 model based agents, 2024. URL <https://arxiv.org/abs/2406.19966>.

621 Google. 2025. Gemini 3 pro model card. Technical
622 report, Google.

623 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao
624 Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shi-
625 rong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025.
626 Deepseek-r1: Incentivizing reasoning capability in
627 llms via reinforcement learning. *arXiv preprint*
628 *arXiv:2501.12948*.

Bingguang Hao, Maolin Wang, Zengzhuang Xu,
Yicheng Chen, Cunyin Peng, Jinjie Gu, Chenyi
Zhuang, and Ji Zhang. 2025a. Reasoning through
exploration: A reinforcement learning frame-
work for robust function calling. *arXiv preprint*
arXiv:2508.05118. 629 630 631 632 633 634

Bingguang Hao, Maolin Wang, Zengzhuang Xu, Cun-
yin Peng, Yicheng Chen, Xiangyu Zhao, Jinjie
Gu, Chenyi Zhuang, and Ji Zhang. 2025b. Bal-
ancesft: Improving llm function calling with bal-
anced training signals and data hardness. *arXiv*
preprint arXiv:2505.20192. 635 636 637 638 639 640

Chengrui Huang, Shen Gao, Zhengliang Shi, Dong-
sheng Wang, and Shuo Shang. 2025. Ttpa: Token-
level tool-use preference alignment training frame-
work with fine-grained evaluation. *arXiv preprint*
arXiv:2505.20016. 641 642 643 644 645

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam
Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow,
Akila Welihinda, Alan Hayes, Alec Radford, and 1
others. 2024. Gpt-4o system card. *arXiv preprint*
arXiv:2410.21276. 646 647 648 649 650

Carlos E Jimenez, John Yang, Alexander Wettig,
Shunyu Yao, Kexin Pei, Ofir Press, and Karthik
Narasimhan. 2023. Swe-bench: Can language mod-
els resolve real-world github issues? *arXiv preprint*
arXiv:2310.06770. 651 652 653 654 655

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. 656 657 658 659 660 661

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. *arXiv preprint*
arXiv:2304.08244. 662 663 664 665 666

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingx-
uan Wang, Bingzheng Xu, Bochao Wu, Bowei
Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025.
Deepseek-v3. 2: Pushing the frontier of open large
language models. *arXiv preprint arXiv:2512.02556*. 667 668 669 670 671

Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian
Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao
Feng, Rithesh RN, and 1 others. 2024. Apigen: Au-
tomated pipeline for generating verifiable and diverse
function-calling datasets. *Advances in Neural Infor-*
mation Processing Systems, 37:54463–54482. 672 673 674 675 676 677

Ali Mohammadjafari, Anthony S Maida, and Raju Got-
tumukkala. 2024. From natural language to sql: Re-
view of llm-based text-to-sql systems. *arXiv preprint*
arXiv:2410.01066. 678 679 680 681

OpenAI. 2025. Gpt-5.2 system card. Technical report,
OpenAI. 682 683

684	Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie	An Yang, Anfeng Li, Baosong Yang, Beichen Zhang,	740
685	Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E	Binyuan Hui, Bo Zheng, Bowen Yu, Chang	741
686	Gonzalez. The berkeley function calling leaderboard	Gao, Chengen Huang, Chenxu Lv, and 1 others.	742
687	(bfc1): From tool use to agentic evaluation of large	2025. Qwen3 technical report. <i>arXiv preprint</i>	743
688	language models. In <i>Forty-second International Con-</i>	<i>arXiv:2505.09388</i> .	744
689	<i>ference on Machine Learning</i> .		
690	Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo	Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang,	745
691	Zhang, Tulika Awalgaonkar, Shiyu Wang, Zhiwei	Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang,	746
692	Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles,	Chen-Yu Lee, and 1 others. 2025. Magnet: Multi-	747
693	and 1 others. 2025. Apigen-mt: Agentic pipeline	turn tool-use data synthesis and distillation via graph	748
694	for multi-turn data generation via simulated agent-	translation. In <i>Proceedings of the 63rd Annual Meet-</i>	749
695	human interplay. <i>arXiv preprint arXiv:2504.03601</i> .	<i>ing of the Association for Computational Linguistics</i>	750
696		(Volume 1: Long Papers), pages 32600–32616.	751
697	Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang,		
698	Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and	Qiyang Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan,	752
699	Heng Ji. 2025. Toolrl: Reward is all tool learning	Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan,	753
700	needs. <i>arXiv preprint arXiv:2504.13958</i> .	Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo:	754
701		An open-source llm reinforcement learning system	755
702	Zhuocheng Shen. 2024. Llm with tools: A survey.	at scale. <i>arXiv preprint arXiv:2503.14476</i> .	756
703	<i>arXiv preprint arXiv:2409.18807</i> .		
704	Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin	Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin	757
705	Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin	Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao	758
706	Lin, and Chuan Wu. 2024. Hybridflow: A flexible	Zeng, Jiajie Zhang, and 1 others. 2025a. Glm-4.5:	759
707	and efficient rlhf framework. <i>arXiv preprint arXiv:</i>	Agentic, reasoning, and coding (arc) foundation mod-	760
708	<i>2409.19256</i> .	els. <i>arXiv preprint arXiv:2508.06471</i> .	761
709	Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen,	Xingshan Zeng, Weiwen Liu, Lingzhi Wang, Liangyou	762
710	Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru	Li, Fei Mi, Yasheng Wang, Lifeng Shang, Xin	763
711	Chen, Yuankun Chen, Yutian Chen, and 1 others.	Jiang, and Qun Liu. 2025b. Toolace-mt: Non-	764
712	2025. Kimi k2: Open agentic intelligence. <i>arXiv</i>	autoregressive generation for agentic multi-turn inter-	765
713	<i>preprint arXiv:2507.20534</i> .	action. <i>arXiv preprint arXiv:2508.12685</i> .	766
714	Qwen Team. 2025. Qwq-32b: Embracing the power of	Kangning Zhang, Wenxiang Jiao, Kounianhua Du,	767
715	reinforcement learning.	Yuan Lu, Weiwen Liu, Weinan Zhang, Lei Zhang,	768
716	Qwen Team and 1 others. 2024. Qwen2 technical report.	and Yong Yu. 2025a. Looptool: Closing the data-	769
717	<i>arXiv preprint arXiv:2407.10671</i> , 2(3).	training loop for robust llm tool calls. <i>arXiv preprint</i>	770
718	Maolin Wang, Yingyi Zhang, Cunyin Peng, Yicheng	<i>arXiv:2511.09148</i> .	771
719	Chen, Wei Zhou, Jinjie Gu, Chenyi Zhuang,	Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz,	772
720	Ruocheng Guo, Bowen Yu, Wanyu Wang, and 1	Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding	773
721	others. 2025a. Function calling in large language	Yu, and Guilin Liu. 2025b. Nemotron-research-tool-	774
722	models: Industrial practices, challenges, and future	n1: Exploring tool-using language models with rein-	775
723	directions.	forced reasoning. <i>arXiv preprint arXiv:2505.00024</i> .	776
724	Yanlin Wang, Xinyi Xu, Jiachi Chen, Tingting Bi, Wen-	Yaowei Zheng, Richong Zhang, Junhao Zhang, Yan-	777
725	chao Gu, and Zibin Zheng. 2025b. An empirical	han Ye, Zheyang Luo, Zhangchi Feng, and Yongqiang	778
726	study of agent developer practices in ai agent frame-	Ma. 2024. Llamafactory: Unified efficient fine-	779
727	works . <i>Preprint</i> , arXiv:2512.01939.	tuning of 100+ language models. <i>arXiv preprint</i>	780
728	xAI. 2025. Grok 4.1 model card. Technical report, xAI.	<i>arXiv:2403.13372</i> .	781
729		A Data	782
730	Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan	Model Selection and Number of Attempts. To	783
731	Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhou-	ensure computational efficiency for large-scale	784
732	jun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu,	synthesis, we restrict candidates models around	785
733	Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caim-	30B parameter scale and evaluate three strong	786
734	ing Xiong, Victor Zhong, and Tao Yu. 2024. Os-	generators—QwQ-32B (Team, 2025), Qwen3-	787
735	world: Benchmarking multimodal agents for open-	32B (Yang et al., 2025), and Qwen3-30B-A3B-	788
736	ended tasks in real computer environments . <i>Preprint</i> ,	Thinking (Yang et al., 2025) under identical Hard-	789
737	arXiv:2404.07972.	Gen configurations by having each model gener-	790
738	Zhangchen Xu, Adriana Meza Soria, Shawn Tan,	ate 2,000 trajectories. In Figure 5 , Qwen3-30B-	791
739	Anurag Roy, Ashish Sunil Agrawal, Radha Pooven-	A3B-Thinking achieves the highest correctness rate	792
	dran, and Rameswar Panda. 2025. Toucan: Synthe-	across all K values, reaching 89% at K=3. Notably,	793
	sizing 1.5 m tool-agentic data from real-world mcp		
	environments. <i>arXiv preprint arXiv:2510.01179</i> .		

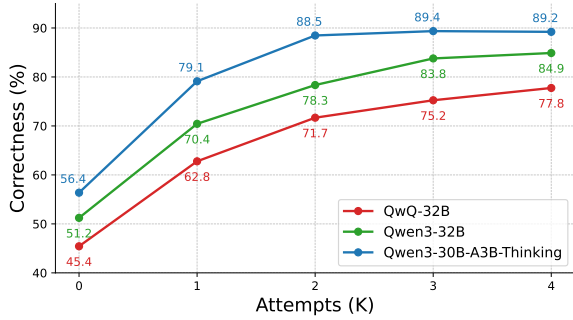


Figure 5: **Analysis of generator model selection.** Correctness rates of function call synthesis at different numbers of attempts (K) for three generator models.

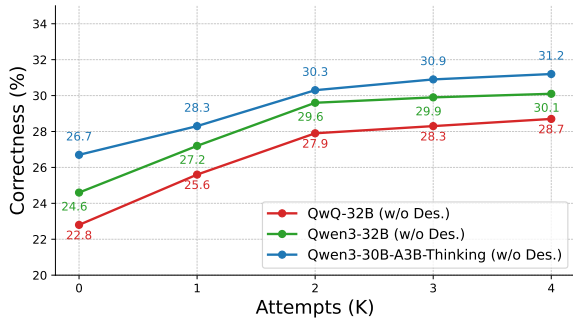


Figure 6: **Impact of advanced tool descriptions.** Correctness rates of the generator models across attempts (K) when advanced tool descriptions are omitted, illustrating the necessity of high-level abstractions for reasoning.

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 5, Figure 6 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

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

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.

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 7, we evaluate three models—Qwen3-32B (Yang et al., 2025), QwQ-32B (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.

B 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.) 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 models 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 Pro-

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 8: Out-of-distribution evaluation on BFCLv4.

859 cess Accuracy, which measures the consistency
860 between the actual function call process and the
861 ideal process. This approach is designed to capture
862 task completion in realistic, interactive scenarios.
863 **APIBank.** For realism, the API-Bank (Li et al.,
864 2023) establishes a framework for runnable eval-
865 uation, grading tasks into complex, multi-step se-
866 quences. The system verifies whether the same
867 database queries or modifications are performed to
868 ensure the ground-truth outcome is achieved.

869 C Performance on the Held-out 870 Benchmark BFCLv4

871 **Generalization of Agentic Capabilities.** To assess
872 the transferability of the tool-use skills induced by
873 HardGen, we further evaluate our method on the
874 out-of-distribution BFCLv4 benchmark, compris-
875 ing two representative agentic tasks: the Search
876 and Memory subsets. From the results shown in
877 Table 8, the HardGen-8B-RL model exhibits re-
878 markable robustness, delivering a substantial uplift
879 over the base model. Specifically, the overall per-
880 formance surges by **+13.54** points (rising from 6.88
881 to 20.42). This advantage is most pronounced in
882 the Memory subset, where accuracy improvement
883 is **+14.09** (from 10.75 to 24.84), accompanied by
884 a performance boost of **+13.00** in the Web Search
885 subset (3.00 to 16.00). Crucially, HardGen-RL con-
886 sistentlly outperforms all other baselines built on the
887 same backbone. These results strongly suggest that
888 our data generation paradigm effectively instills
889 robust agentic behaviors, providing a solid founda-
890 tion for future research in agentic reinforcement
891 learning.

892 D Training details

893 **Supervised Finetuning.** We conduct our super-
894 vised finetuning experiments using the open-source
895 Llama Factory library (Zheng et al., 2024). The
896 main hyperparameter settings are listed in Table 9.
897 **RL Training.** We conduct our reinforcement learn-
898 ing (RL) experiments using the open-source Verl

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

Table 9: 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 10: Hyperparameters for RL training.

899 library (Sheng et al., 2024). To ensure stable and
900 efficient training, we adopt the training settings
901 of (Zhang et al., 2025b), where reinforcement learn-
902 ing is applied directly without supervised finetun-
903 ing stage. The key hyperparameter settings are
904 summarized in Table 10. Both SFT and RL train-
905 ing runs are executed on a cluster equipped with 8
906 NVIDIA H100 80GB GPUs.

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

$$921 \text{Reward} = \begin{cases} 1, & \text{format correct \& answer correct,} \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

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

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

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 12: **Generalization to the Llama model family.** Performance comparison of Llama-3.1-8B and Llama-3.2-3B models before and after HardGen-RL training on BFCLv3 Single-Turn and Multi-Turn tasks.

E Manual Annotations

Impact of the Advanced Tools. Our proposed method introduces advanced tool (T_{adv}) to help LLMs bridge logical jumps during hard query synthesis. We present the proportion of challenging queries across all model scales in Table 5. However, 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 navigate 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 annotate the samples anew 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 11, 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.

F Results on Other Models

Results on Llama3 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 12, 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 confirm that our approach effectively transfers across model families and parameter scales, establishing HardGen as a robust data generation framework for enhancing tool-use capabilities beyond the Qwen architecture.

G Construction of the API Graph

Structure of API Graph. The API Graph $\mathcal{G} = (\mathcal{T}, \mathcal{D}, \mathcal{P})$ encodes three types of information critical for hard trace generation: (1) *Failure Tool Set* \mathcal{T} : the set of 1,204 failure APIs identified through self-evaluation; (2) *Dependencies* \mathcal{D} : directed edges representing prerequisite relationships, where $(T_i, T_j) \in \mathcal{D}$ indicates that T_j requires T_i to be executed first; and (3) *Parameter Constraints* \mathcal{P} : specifications defining valid parameter ranges, types, and inter-tool parameter mappings (e.g., output type of T_i must match input type of T_j). Parameter constraints \mathcal{P} are extracted from tool API schemas (type signatures, value ranges and required fields).

Update of API Graph. The API graph is updated through execution feedback from the environment. After each tool call, we analyze the feedback to identify dependencies. When tool T_j serves as a preceding tool of T_i , the execution of T_j activates T_i , making it eligible for selection in subsequent steps. Concurrently, parameter constraints in \mathcal{P} are refined by tracking value validity ranges and type requirements observed during execution. This enables the graph to capture implicit value-level dependencies that extend beyond simple tool-set inclusion relationships. Through this continual update process, the API graph progressively refines both structural and parameter-level dependencies, thereby biasing subsequent sampling toward tool sequences that are both executable and challenging.

H 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 7 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 8 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 9 and 10 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.

I Prompts

This section provides the prompts used in the Hard-Gen 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 11) synthesizes advanced tools from multi-step execution hard traces by abstracting primitive tool sequences into high-level operations. The Hard Query Generator (Figure 12) 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 13 and Figure 15) attempts to solve hard queries with hints from advanced tool descriptions, while the Verifier (Figure 14) analyzes incorrect attempts and provides targeted corrective feedback without revealing answers directly. These prompts collectively enable HardGen’s closed-loop refinement mechanism, where each component builds upon the outputs of previous stages. The Tool Maker and Hard Query Generator transform executable traces into challenging learning scenarios, while the Reasoner-Verifier loop iteratively refines chain-of-thought reasoning toward correct solutions. All prompts are designed to be model-agnostic and can be adapted to different LLM architectures. The structured output format ensures consistency and inherent executability, enabling direct execution without additional validation.

Case: Hard Query Construction with Logical Jump

Original Trace:
get_zipcode(city="Rivermist") → "83214"
get_zipcode(city="Stonebrook") → "74532"
buy_tickets(zipcodeA="83214", zipcodeB="74532") → "ticket_id": "14589"

Advanced Tool: buy_tickets_adv(cityA, cityB) -> ticket_id
Description: Purchase air tickets between two cities by city names, returning the purchased ticket information.

Easy Query:
Please check the zip code of the city "Rivermist" and "Stonebrook" first, then purchase air tickets between the two cities according to the zip codes you checked.
Hard Query:
Please purchase air tickets between the city "Rivermist" and the city "Stonebrook".

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

Figure 7: **Case Study: Hard Query Construction.** Illustration of the data synthesis process where an original execution trace is abstracted into an "Advanced Tool" to generate a Hard Query that omits explicit intermediate steps (zipcode lookup).



Figure 8: **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).

Complete Trajectory – Turn 1

User: Determine the current working directory path, and then search for all files and directories in the current directory and its subdirectories.

<think>
 The user wants to: (1) determine current working directory, (2) search all files/directories recursively.
 • pwd: returns current directory path (no parameters needed)
 • find: searches files/directories; path="." for current dir, name=None for all items
 The ls command only lists current directory, not subdirectories, so find is the right choice.
</think>

<tool_call>
 [pwd(), find(path=".")]
</tool_call>

<tool_response>
 [{"pwd()": '{"current_working_directory": "/workspace"}'},
 {"find(path=".)": '{"matches": ["/processed", "/data", "/data/raw",
 "/data/raw/dtjsp.json", "/data/unggv.json", ...]}'}]
</tool_response>

Assistant Summary: The current working directory is /workspace. The search found the following items:
 ["/processed", "/data", "/data/raw", "/data/raw/dtjsp.json", ...]

Figure 9: **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 `dtpsp.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 unggv.json` → display file contents
- (3) `cd raw` → change to raw subdirectory
- (4) `tail dtpsp.json lines=7` → display last 7 lines

</think>

<tool_call>

```
[cd(folder="data"), cat(file_name="unggv.json"), cd(folder="raw"), tail(file_name="dtpsp.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", "type": "product"}}`

The last 7 lines of `"dtpsp.json"`: `{"data": {"id": "27", "type": "user"}}`

Figure 10: **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 traces.

Given the following execution trace:
 {execution_trace}

Trace 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 trace
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 trace:
 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 11: **Prompt template for the Tool Maker Agent.** Instructions used to synthesize advanced tools from multi-step execution traces.

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 12: **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 13: **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:

1. **Error Type:** Wrong tool selection, missing tool calls, incorrect parameters, wrong order, type mismatch, or missing dependencies
2. **Root Cause:** Why did the model make this error? What logical step was missed? What dependency was not recognized?
3. **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 14: **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 15: **Prompt template for the Reasoner Agent (Refinement Iteration)**. Instructions for revising reasoning and function calls based on feedback from the Verifier.