RAREPLAY: EXPLORING SCARCE ACTION SPACES IN TOOL GRAPHS VIA SELF-EVOLVING SYNTHESIS

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) have demonstrated strong function-calling capabilities, yet their performance on long-horizon tool-use tasks remains heavily dependent on the quality and diversity of training trajectories. Existing synthetic data generation methods typically rely on static tool graphs or multi-agent collaboration, which can produce homogeneous trajectories, incur substantial computational overhead, and may struggle to adapt task difficulty to the evolving capabilities of the model. We introduce RarePlay, a self-evolving framework for synthesizing diverse and progressively challenging function-calling data. RarePlay constructs a state transition graph from the current model's behavior and introduces Rarity-Guided Exploration (RGE) to prioritize actions that are underrepresented under the current policy while remaining likely to lead to successful outcomes. Sampled trajectories are converted into training tasks through inverse task synthesis and further filtered using rubric-based rejection sampling to improve logical consistency and trajectory quality. By iteratively updating both the model and the state graph, RarePlay incorporates newly explored states and transitions, providing a richer foundation for sampling longer paths and supporting an easy-to-hard curriculum guided by a predefined hop-count schedule. Experiments on BFCL-v4 and AutomationBench demonstrate that RarePlay improves function-calling performance and achieves state-of-the-art results under comparable training-data budgets.

1 INTRODUCTION

Recent advances in large language models (LLMs) have accelerated the development of general-purpose agents for real-world tasks. These agents rely on multi-turn function calling (FC) to solve complex problems through sequential tool use (Xu et al., 2026b; Ding et al., 2026; Li et al., 2026). However, constructing complex tasks and executing reliable tool interactions is challenging, making diverse, long-horizon tasks and successful trajectories scarce. Synthesizing such data is therefore essential for extending foundation models beyond basic FC capabilities (Hu et al., 2025; Gandhi et al., 2026; Chen et al., 2026).

Existing task synthesis approaches largely fall into two categories. Tool definition graph based methods represent tools and their relationships as a graph, then sample paths through random walks to construct tasks and trajectories (Yin et al., 2025; Yang et al., 2025b). However, tool-level paths specify which tools to invoke, but do not define the initial environment or the target conditions against which the resulting state can be checked to determine task success. Meanwhile, unguided sampling may provide insufficient coverage of infrequent behaviors that require additional training. Multi-agent collaborative methods assign complementary roles to agents that jointly propose and solve tasks (Xu et al., 2026a; Li et al., 2025), but the underlying models' capabilities and behavioral preferences can constrain data diversity. Both paradigms also give limited consideration to the policy model's evolving distribution, making it difficult to adapt task difficulty and coverage as the model improves. These limitations motivate a framework that combines verifiable task synthesis, targeted behavioral coverage, and an adaptive curriculum.

To address these limitations, we introduce **RarePlay**, a framework for synthesizing diverse FC data that supports an easy-to-hard curriculum, as illustrated in Figure 1. Our key observation is that *rare but successful behaviors* in policy rollouts offer valuable opportunities for targeted training: they

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

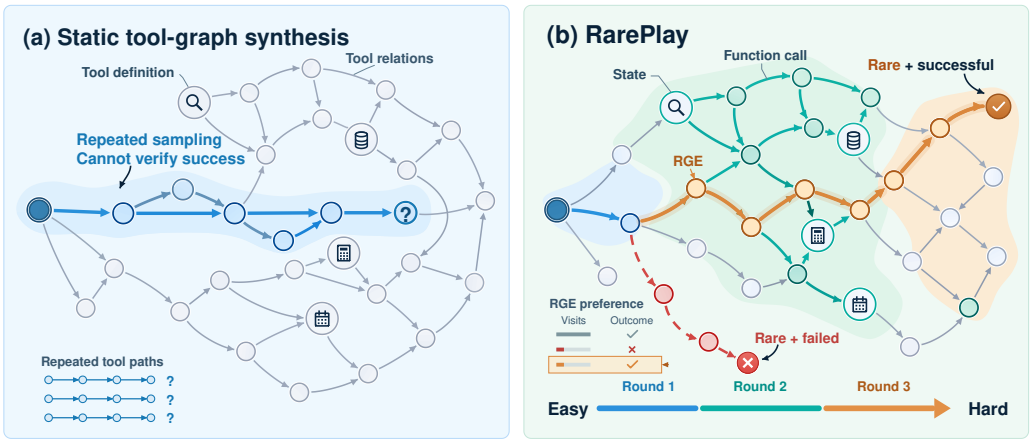

Figure 1: RarePlay explores rare yet successful trajectories, whereas conventional synthesis methods tend to repeatedly explore similar behaviors.

reveal tool-use strategies that the model can execute successfully but seldom employs. Synthesizing tasks around these behaviors can provide additional practice to consolidate and broaden the model's existing capabilities. To identify and use these behaviors, we organize trajectories sampled from the policy model into a state transition graph, where nodes record sandbox states and edges represent tool mediated transitions. Unlike graphs defined solely over tools, this representation preserves both the execution process and the final state, allowing task success to be verified by checking whether the final state satisfies the task's goal conditions. As training progresses, the graph is reconstructed from newly sampled trajectories, incorporating newly visited states and transitions that can enrich the available paths for task synthesis. This evolving graph provides a foundation for an easy-to-hard curriculum by supporting progressively higher sampling hop counts, which we increase according to a predefined schedule to synthesize longer tool-use sequences.

Building on this graph, we introduce a **Rarity-Guided Exploration (RGE)** function that balances task success, exploration, and tool diversity. Inspired by Monte Carlo Tree Search (MCTS) (Kocsis & Szepesvári, 2006), RGE combines a success rate reward with a visit count based exploration bonus to favor promising but underexplored transitions. It further rewards switching between tools and applies a transition penalty to discourage cross-domain transitions. We use these scores to guide state transition path sampling, then perform **Inverse Task Synthesis** to construct corresponding tasks. To verify the resulting trajectories, we generate rubrics from each sampled path and its initial and final sandbox states, checking both task correctness and coverage of the targeted behaviors.

With model size and training data count held constant, the model trained on data synthesized by RarePlay outperforms its counterpart trained on data from the state-of-the-art tool-graph-based synthesis method by 5.75 percentage points. It also surpasses the baseline model of the same size by 8.13 percentage points. Analysis across successive training rounds further shows that synthesized tasks become longer while model performance improves. These results suggest that RarePlay supports a task curriculum that develops alongside the policy model's capabilities.

Our main contributions are as follows:

- **Dynamic State-Transition Graph**: We introduce a task synthesis framework that organizes policy rollouts into a state-transition graph, providing explicit execution states for task verification and expanding the task space as the policy model evolves.
- **Rarity-Guided Task Synthesis**: We propose RGE to prioritize rare yet successful transitions and combine it with inverse task synthesis to construct tasks that improve coverage of infrequent behaviors.
- **Empirical Effectiveness**: With matched model sizes and training data counts, RarePlay outperform those the state of the art tool graph based method by 5.75 percentage points and the baseline model by 8.13 points.

2 RELATED WORKS

2.1 FUNCTION CALLING IN LARGE LANGUAGE MODELS

As LLMs continue to advance, the focus of much research is shifting from improving their conversational capabilities (Chang et al., 2026; Zheng et al., 2026) to enabling them to accomplish tasks more effectively (Merrill et al., 2026; Zhang et al., 2026; Yao et al., 2024). To enable LLMs to operate effectively in real-world environments, tool-use capabilities are indispensable. To evaluate models' tool-use capabilities, a substantial body of work has emerged in the benchmarking community (Wang et al., 2026; Ma et al., 2024; Yang et al., 2026; Lu et al., 2025). Among these benchmarks, BFCL (Patil et al., 2025) is one of the most representative. It covers both single-turn and multi-turn tool-use scenarios, while also providing comprehensive evaluations of applications such as web search and agentic tasks. In addition, it incorporates a range of common failure cases, including missing tools and missing arguments. Beyond BFCL, τ-bench (Yao et al., 2024) evaluates agents in realistic user interactions that require both tool use and policy compliance. More recently, AutomationBench (Shepard & Salimans, 2026) further extends evaluation to end-to-end automation across multiple applications, focusing on whether agents can successfully complete real-world workflows. Building on these benchmarks, our work focuses on improving stateful, multi-turn tool use.

2.2 DATA SYNTHESIS FOR FUNCTION CALLING

Recent efforts have shifted toward synthesizing function-calling data at scale to reduce reliance on costly human annotation. Among them, ToolACE (Liu et al., 2024) demonstrates that large-scale synthetic data—generated through self-evolving APIs (TSS), multi-agent dialogs (SDG), and dual-layer verification (DLV)—can improve LLMs' function-calling capabilities. Alternatively, MAGNET (Yin et al., 2025) conducts large-scale data synthesis by constructing extensive API dependency graphs and integrating them with graph translation methods. Similarly, methods such as ToolMind (Yang et al., 2025b), FunReason-MT (Xu et al., 2025), and RouteNator (Belavadi et al., 2025) all adopt graph-based data synthesis paradigms. However, because their generation processes rely primarily on static offline graphs, these approaches may be limited in their ability to dynamically synthesize data with varying difficulty levels tailored to the current model's training requirements.

3 PROBLEM SETUP

The function calling (FC) problem can be formally defined as follows. Let $T = [tool_1, tool_2, \ldots, tool_k]$ denote the list of available tools and q a query. Each input instance is represented as a tuple (T, q), for which the model generates an action sequence with tool-calling steps represented by (t_i, a_i, o_i). Here, t_i denotes the model's reasoning process, a_i the action taken, and o_i the observation returned by the environment. Each action $a_i = (tool, args)$ specifies the selected tool and its arguments. Executing the action sequence leads to a final environment state $state$, which records the underlying environment configuration, such as file contents or database records. Unlike this internal state, o_i is the tool response exposed to the model and may contain only partial information or an execution status. Task success is determined by comparing $state$ with the ground-truth state $state_{gt}$. Based on this formulation, task synthesis aims to construct tasks specified by a query q, an initial environment state $state_0$, and a ground-truth final state $state_{gt}$ for a given tool set T. Each synthesized task can be instantiated in the sandbox using $state_0$ and evaluated against $state_{gt}$.

4 METHODS

RarePlay iteratively synthesizes training data targeting rare but successful behaviors in the policy model's rollouts. As illustrated in Figure 2, each round begins by constructing a state-transition graph from trajectories sampled from the current policy model. We randomly select seed nodes and sample paths from them using the Rarity-Guided Exploration (RGE) function. A closed-source teacher model then performs inverse task synthesis on the sampled paths, followed by rejection sampling and trajectory distillation to produce training data. After training the policy model on these data, we

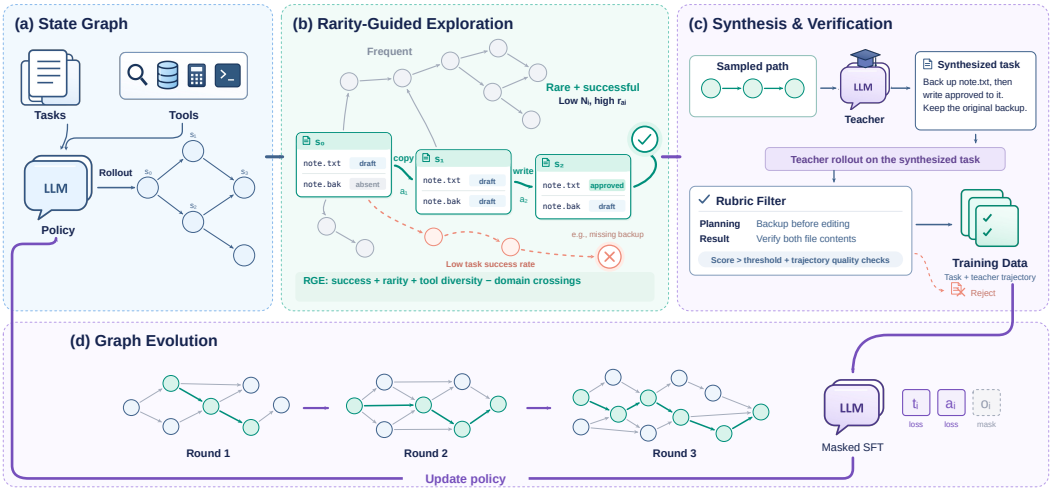

Figure 2: Overview of RarePlay. RarePlay first characterizes model behaviors from a observation set and organizes them into a state graph. Rarity-Guided Exploration then identifies rare but valuable trajectories for teacher synthesis. The synthesized data are filtered via rejection sampling, and the process is iterated with an evolving state graph and a predefined hop-count schedule to support an easy-to-hard curriculum.

collect new rollouts and rebuild the graph for the next round. This procedure repeats for a predefined number of rounds.

4.1 State Transition Graph

To characterize the policy model's output distribution over the current tasks, we construct a state-transition graph from its sampled trajectories. We first select M instances from the existing task set as an observation dataset. At the beginning of each data construction round, we perform $K_{rollout}$ rollouts with the current policy model on each instance and pool the resulting $MK_{rollout}$ trajectories into a set $\mathcal{D}$. For each step (t_i, a_i, o_i), we record the environment states s_{i-1} and s_i immediately before and after executing a_i, respectively. We then construct a directed graph $\mathcal{G} = (V, E)$ from the pooled trajectories, where nodes represent environment states and each edge $e_i = (s_{i-1}, a_i, s_i)$ records a tool transition. The graph is rebuilt from scratch in each round using newly sampled trajectories, allowing it to reflect the current policy model's output distribution.

The resulting graph may contain many redundant state nodes. To reduce this redundancy, following prior work (Feng et al., 2026), we encode each state's textual representation using BGE-M3 (Chen et al., 2024) and group states with pairwise similarity of at least 0.9 into clusters. Each cluster forms a node in the compressed graph, with edge traversal counts and success statistics aggregated accordingly. Figure 3 illustrates how a trajectory is converted into the state-transition graph.

4.2 Rarity-Guided Exploration Function.

After constructing $\mathcal{G}$, our goal is to leverage it to explore patterns that are practically important yet underrepresented in the model's current output distribution. However, rewarding only rare behaviors makes it difficult to distinguish between beneficial and undesirable rare behaviors, while also causing abrupt shifts in the output distribution. Therefore, we design a sampling function that jointly balances low-probability behaviors and task success rates. Inspired by Monte Carlo Tree Search (MCTS), we propose a Rarity-Guided Exploration (RGE) function to better balance exploration toward low-probability behaviors and task success. Its formulation is given in Eq. 1.

$$\mathrm{RGE}(a_i) = r_{a_i} + c_{\mathrm{puct}} \sqrt{\frac{\ln N_{\mathrm{parent}}}{N_i}} + \lambda_{\mathrm{div}} \mathbb{I}[f(a_i) \neq f(a_{i-1})] - \lambda_{\mathrm{trans}} d_i. \tag{1}$$

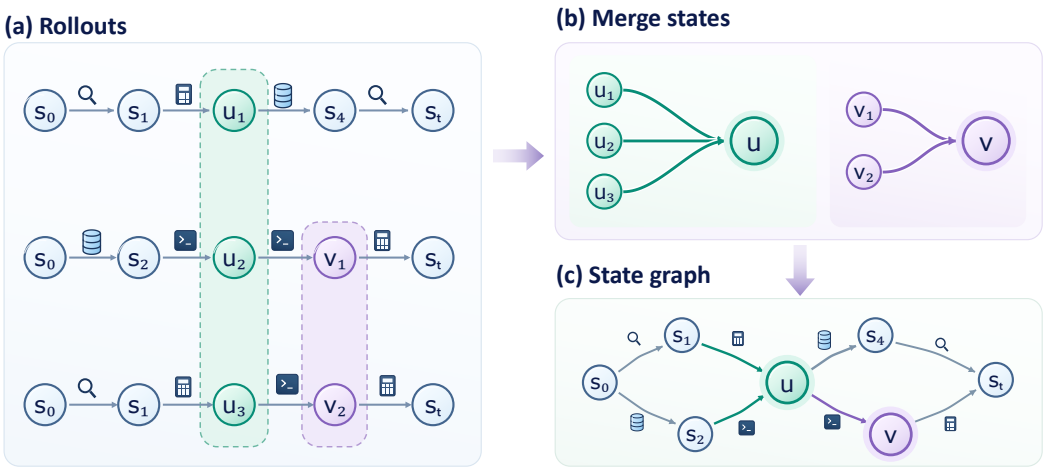

Figure 3: State Transition Graph

Here, for each edge e_i corresponding to action a_i, we record its traversal count N_i and the visit count N_{parent} of its source state s_{i-1}. We also compute r_{a_i} as the empirical task success rate of rollouts that traverse e_i:

$$r_{a_i} = \frac{\sum_{\tau \in \mathcal{D}} \mathbf{1}\{e_i \in \tau\}\, \text{Success}(\tau)}{\sum_{\tau \in \mathcal{D}} \mathbf{1}\{e_i \in \tau\}}. \tag{2}$$

Here, $\text{Success}(\tau) = \mathbb{I}[\tau \text{ succeeds}]$ indicates whether rollout τ successfully completes the task. Thus, r_{a_i} measures task-level success conditioned on transitioning from state s_{i-1} to state s_i via the tool call specified by action a_i, rather than the success of the individual tool call itself.

With these statistics, the first term in Eq. 1, r_{a_i}, favors behaviors associated with task success, while the second term, $c_{\text{puct}}\sqrt{\ln N_{\text{parent}}/N_i}$, encourages exploration of rare behaviors using traversal counts as a proxy for rarity. For a given source state, edges with smaller N_i receive larger exploration bonuses, favoring behaviors that are underrepresented in the collected rollouts. The coefficient c_{puct} controls the strength of this exploration, balancing rarity against task success.

The function $f(a_i)$ returns the tool selected by action a_i, and λ_{div} rewards switching tools between consecutive steps. Finally, d_i denotes the cumulative number of task-domain crossings along the sampled path up to and including action a_i, and λ_{trans} controls the corresponding penalty. Here, task domains correspond to business categories, and a crossing occurs when an edge connects states associated with tasks from different categories. The hyperparameter settings are provided in Appendix D.

Then, for each node, we compute the RGE score for all candidate successors and convert these scores into sampling probabilities using a softmax function. The resulting sampling probability is defined in Eq. 3:

$$p(a_i \mid s) = \frac{\exp\left(\text{RGE}(a_i)/T_{temp}\right)}{\sum_{a_j \in \mathcal{A}(s)} \exp\left(\text{RGE}(a_j)/T_{temp}\right)}, \tag{3}$$

where $\mathcal{A}(s)$ denotes the set of candidate outgoing actions at state s, and T_{temp} is the softmax temperature controlling the sharpness of the sampling distribution.

During data synthesis, we first randomly sample a starting state s_{start}. Starting from s_{start}, we compute the transition probabilities over candidate successors at each node according to Eq. 1 and Eq. 3, and perform a random walk for at most K steps. This process yields a candidate state-action chain that jointly considers behavioral diversity, underrepresented regions of the output distribution, task success rate, and task complexity. The resulting state-action chain is then used for subsequent task synthesis and data filtering. This sampling process is repeated over multiple iterations, with each iteration producing a model with an updated output distribution. Through this iterative procedure, the model and the State Transition Graph co-evolve over time.

4.3 Inverse Task Synthesis and Rejection Sampling

After sampling plausible underrepresented paths using the RGE function, we perform inverse task synthesis followed by rejection sampling. Specifically, for each sampled path, we collect all (a_i, o_i) pairs and provide s_{start} and s_{end} to a closed-source LLM. Conditioned on the initial state, terminal state, and the sequence of intermediate actions along the path, the LLM is prompted to inversely synthesize a plausible task that is consistent with the observed state transitions. Specifically, this process can be formulated as Eq. 4:

$$\hat{\mathcal{T}} = \text{LLM}_{\text{teacher}}\left(s_{\text{start}}, \{(a_i, o_i)\}_{i=1}^{L_{path}}, s_{\text{end}}\right), \tag{4}$$

After synthesizing the task, we first use a closed-source teacher model to augment it with the corresponding chain-of-thought (CoT) reasoning and teacher-model interaction trajectory. We then apply further rejection sampling to improve the quality of the synthesized data. First, for each task, we construct a task-specific rubric based on its synthesis process to assess both task correctness and preservation of the targeted behaviors in the teacher trajectory. Specifically, for each task, we provide all intermediate actions and states from the synthesis process to the teacher model and ask it to extract a rubric consisting of the following two types of items:

- **Planning.** The rubric specifies intermediate state conditions that capture the targeted tool use behaviors motivated by the sampled path. These criteria assess whether the teacher trajectory preserves the intended training coverage while allowing alternative valid execution paths.
- **Result Consistency.** The rubric specifies the final state conditions required to fulfill the synthesized user request.

Following these two principles, the teacher model inversely synthesizes a set of rubric items and assigns a corresponding weight to each of them. The resulting rubric is represented as a collection of n weighted criteria, as formulated in Eq. 5.

$$\mathcal{R} = \{(c_i, w_i)\}_{i=1}^{n}, \qquad \sum_{i=1}^{n} w_i = 100, \tag{5}$$

For a teacher trajectory τ, the rubric score is defined as

$$\text{Score}(\tau, \mathcal{R}) = \sum_{i=1}^{n} w_i \, g_{\text{LLM}}(c_i, \tau),$$

where c_i specifies a task-correctness or behavioral-coverage criterion, w_i is its positive weight reflecting its importance to these objectives, and $g_{\text{LLM}}(c_i, \tau) \in [0, 1]$ evaluates its satisfaction using the trajectory and recorded environment states. Since $\sum_{i=1}^{n} w_i = 100$, the overall score lies in $[0, 100]$ and reflects both task correctness and behavioral coverage. Finally, for each generated trajectory, RarePlay retains only those whose rubric-based score exceeds the threshold λ_r. It further rejects trajectories that meet any of the following conditions: (1) containing at least λ_t consecutive identical tool calls, where two calls are considered identical only if both their tool names and arguments match; (2) having malformed o_i observations in more than 50% of the trajectory steps; or (3) invoking the same tool at least λ_u times. Detailed prompts are provided in Appendix A. For clarity, we provide a simple illustrative example and its corresponding rubric in Appendix B. Detailed hyperparameter settings are provided in Appendix D.

4.4 Iterative Training and Graph Evolution

After collecting a sufficient number of tasks and their corresponding trajectories, we perform Supervised Fine-Tuning (SFT) on the model using these trajectories. To focus supervision on reasoning and tool actions during training and encourage the model to better couple reasoning with tool actions, we mask the loss contributions from tool observations. Given a synthesized task $\hat{\mathcal{T}}$ and its corresponding

trajectory $(t_i, a_i, o_i)_{i=1}^{L_{teacher}}$, we optimize the model only over reasoning tokens t_i and action tokens a_i, while excluding observation tokens o_i from the training objective. The SFT loss is defined as

$$\mathcal{L}_{\text{SFT}} = -\sum_{j=1}^{|\mathcal{H}|} m_j \log \pi_\theta \left(x_j \mid \hat{\mathcal{T}}, x_{<j} \right), \tag{6}$$

where $\mathcal{H} = (x_1, \ldots, x_{|\mathcal{H}|})$ denotes the token sequence obtained by serializing the interaction trajectory, and $x_{<j}$ denotes all trajectory tokens preceding position j. The loss mask m_j equals 1 if position j belongs to a reasoning or action span, and 0 otherwise. Tool observations remain in the context for subsequent reasoning and action generation, but their tokens do not contribute directly to the SFT loss.

After each training round, we add a subset of tasks not used for training to the observation set, collect rollouts with the updated policy, and reconstruct the state-transition graph. These rollouts can introduce new states and transitions, enriching the paths available for synthesis. We progressively increase sampling hop counts according to a predefined schedule to target longer tool-use sequences, then use the synthesized tasks and filtered teacher trajectories to train the policy in the next round.

5 Experiments

5.1 Experiment Setup

Datasets. To validate the effectiveness of our method, we mainly evaluate it on BFCL-v4 (Patil et al., 2025) and AutomationBench (Shepard & Salimans, 2026).

BFCL-v4. is a comprehensive benchmark for evaluating the function-calling and agentic capabilities of LLMs, covering diverse tool-use scenarios including multi-turn interactions, web search, memory, and format-sensitive function calling. For ease of evaluation, we restrict our experiments to the multi-turn subset in this work.

AutomationBench. evaluates agents on cross-application business workflows requiring coordinated tool use and verifiable task completion. We evaluate on the Sales subset.

Baselines. We compare against both open- and closed-source baselines. For open-source models, we include Qwen3-4B (Yang et al., 2025a), Qwen3-14B, and Qwen3-30B-A3B. We also evaluate two representative methods for tool-use data synthesis: TOOLACE (Liu et al., 2024), which employs multi-agent collaboration and verification, and TOOLMIND (Yang et al., 2025b), which synthesizes trajectories using a fixed tool graph. In contrast to RarePlay, neither method dynamically reconstructs a state-transition graph from the evolving policy's rollouts to guide data synthesis. Both synthesis baselines use Qwen3-4B as the backbone, with training-data counts matched to those of our method. We further include GPT-5.2 (Singh et al., 2025) as a closed-source baseline.

Metrics. For AUTOMATIONBENCH, we report *pass rate*, the fraction of tasks satisfying all required assertions, and *partial credit*, the average fraction of satisfied assertions per task. For BFCL, we report conversation-level accuracy (Pass@1), requiring all turns to succeed. Higher is better for all metrics.

Table 1: Hop-count configurations across training rounds.

Benchmark	Round 1	Round 2	Round 3
BFCL	2, 5, 7	5, 7, 10	7, 10, 15
AutomationBench	10	15	20

5.2 Implementation Details

For each benchmark, we construct the observation dataset from internally pre-synthesized tasks. We retain only queries whose similarity to any query in the test set is below 0.7, and further apply a second-stage LLM-as-a-judge verification to reduce potential overlap between rollout queries and

Table 2: Results on BFCL and AutomationBench. Bold scores indicate the best result in each column. Green bold Δ rows report absolute gains over the corresponding Qwen3 base model.

	BFCL					AutomationBench	
Method	Base.	M. Param.	M. Func.	Long	Avg.	Partial Credit	Pass
GPT-5.2	36.5	18.0	27.5	30.5	28.13	–	–
Qwen3-30B-A3B	43.5	10.5	25.0	**41.0**	30.00	14.52%	7%
TOOLACE-4B	16.5	15.5	12.5	8.5	13.25	–	–
TOOLMIND-4B	33.5	23.5	17.5	22.5	24.25	–	–
Qwen3-4B	26.0	20.0	18.0	23.5	21.88	1.31%	0.00%
Rareplay-4B	47.0	16.0	28.0	29.0	30.00	5.19%	0.00%
Δ vs. Qwen3-4B	**+21.0**	**−4.0**	**+10.0**	**+5.5**	**+8.13**	**+3.88**	**0.00**
Qwen3-14B	40.5	24.5	22.5	26.5	28.50	10.01%	0.00%
Rareplay-14B	**49.5**	**33.0**	**31.0**	30.5	**36.00**	**24.88%**	**13%**
Δ vs. Qwen3-14B	**+9.0**	**+8.5**	**+8.5**	**+4.0**	**+7.50**	**+14.87**	**+13**

test queries. We perform three rounds of sampling and synthetic-data training in total. Across the three rounds, we use different hop-count configurations for the training data. In the early stages, when the graph is less well connected, we favor lower hop counts to facilitate reliable data synthesis. In later stages, we progressively increase the hop count to maintain sufficient task complexity. The detailed hop-count distributions are provided in Table 1. For task and rubric synthesis, as well as trajectory collection, we use a mixture of GLM-5.3 (Zeng et al., 2026) and Kimi-K3 (Team et al., 2026), with their roles specified in Appendix C. The detailed prompts are provided in Appendix A. We use Qwen3-14B (Yang et al., 2025a) and Qwen3-4B as the base models for training. Using our method, we synthesize a total of 4K training trajectories. Detailed training hyperparameters are provided in Appendix E. For all other baselines, we use the same amount of training data as our method and adopt Qwen3-4B as the backbone model. All experiments are conducted on a single server equipped with eight NVIDIA A100 GPUs.

5.3 Main Results

Table 2 presents the following findings: Compared with the base models, Rareplay yields substantial improvements on both Qwen3-4B and Qwen3-14B. On BFCL, both models improve on the Base, Missing Function, and Long categories, with the largest gains on Base, suggesting that RarePlay's synthesized data supports stronger multi-turn tool use; this improvement is also consistent with the performance gains observed on AutomationBench. Moreover, the models still achieve noticeable improvements on the Miss-Function metrics, for which we do not explicitly synthesize targeted training data. This suggests RarePlay can transfer to task categories not explicitly targeted during synthesis. Meanwhile, compared with ToolMind, which synthesizes data using a fixed tool graph, RarePlay achieves stronger performance under the same training-data budget. This suggests that, by modeling model outputs as a state graph and applying RGE-based function sampling, our method can provide additional training on underrepresented behaviors, thereby achieving higher data efficiency. The results show that our method enables the 14B model to outperform the 30B model, while surpassing the closed-source GPT-5.2 (Singh et al., 2025) baseline in average accuracy on the BFCL.

5.4 Ablation Study

Ablation of the number of training rounds. To validate the effectiveness of our multi-round training, we conduct an ablation study on the number of training rounds, with the results reported in Table 4. Average accuracy improves consistently across rounds. These results support the effectiveness of iteratively augmenting training data with targeted examples derived from the updated policy. Performance on the Long and Missing Function subsets improves steadily, while Base and Missing Parameter accuracy exhibit fluctuations, indicating trade-offs across task categories. These results suggest that the evolving graph provides a richer foundation for sampling longer trajectories,

Table 3: Ablation study on BFCL with the 4B model. Retention denotes the rejection sampling retention rate. Bold denotes the best result in each column.

Method	Base.	M. Param.	M. Func.	Long	Avg.	Retention
w/o RGE	27.5	**22.5**	23.5	20.0	23.38	17.2%
w/o Rej	40.5	15.0	19.0	24.5	24.75	–
RarePlay-4B	**47.0**	16.0	**28.0**	**29.0**	**30.00**	**60%**

Table 4: Ablation study on BFCL with the 4B model. Bold denotes the best result in each column.

Method	Base.	M. Param.	M. Func.	Long	Avg.
RarePlay-1st	42.5	19.5	21.5	24.0	26.88
Rareplay-2nd	38.5	**21.0**	23.0	27.5	27.50
RarePlay-4B	**47.0**	16.0	**28.0**	**29.0**	**30.00**

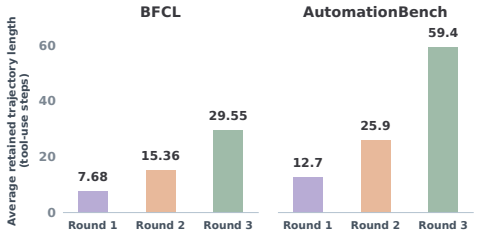

Figure 4: Average retained trajectory length.

supporting progressively longer tool use tasks under the predefined hop-count schedule and improving the model's long horizon tool use performance.

Ablation of Rarity-Guided Exploration. As shown in Table 3, removing the RGE sampling function, which reduces the sampling strategy to random sampling, degrades overall performance, demonstrating the effectiveness of RGE. Moreover, removing RGE lowers the rejection sampling retention rate, indicating that fewer sampled trajectories meet the same acceptance criteria and providing further evidence that RGE improves the quality of sampled trajectories. Random sampling may be less effective at generating informative training examples, while sampling solely based on success rate or occurrence frequency may favor a narrow subset of trajectories, potentially limiting the diversity of collected training data. These results suggest that RGE effectively balances the strengths of the two sampling strategies, leading to improved performance on tool using tasks.

Ablation of Rejection Sampling. As shown in Table 3, Removing rejection sampling consistently degrades performance across all evaluation metrics, demonstrating the effectiveness of our rejection sampling strategy. Although removing rejection sampling yields more task–trajectory pairs, it substantially degrades trajectory quality, resulting in many trajectories that contain repetitive loops or errors. This finding suggests that reverse-synthesized rubrics can help improve training data quality by assessing task correctness and coverage of the targeted behaviors.

Average Retained Trajectory Length. Figure 4 shows that retained teacher trajectories average more tool-use steps across rounds on both benchmarks. Executing synthesized tasks may involve more steps than the sampled paths. Graph evolution and the predefined hop-count schedule support longer synthesis paths, consistent with the intended easy-to-hard curriculum.

6 Conclusion

We introduce RarePlay, an iterative framework for synthesizing function-calling training data over evolving state-transition graphs. Rarity-Guided Exploration targets underrepresented transitions associated with task success, while inverse task synthesis and rubric-based rejection sampling generate tasks and filter teacher trajectories. Graphs are rebuilt from updated policy rollouts, and a predefined hop-count schedule guides synthesis toward longer tool-use sequences. RarePlay improves over base models on BFCL-v4 and AutomationBench, and outperforms evaluated synthesis baselines on BFCL under matched training data counts. These findings support policy-informed graph exploration for multi-turn tool-use data synthesis.

REFERENCES

Vibha Belavadi, Tushar Vatsa, Dewang Sultania, Suhas Suresha, Ishita Verma, Cheng Chen, Tracy Holloway King, and Michael Friedrich. Routenator: A router-based multi-modal architecture for generating synthetic training data for function calling llms. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pp. 129–140, 2025.

Ge Chang, Jinbo Su, Jiacheng Liu, Pengfei Yang, Yuhao Shang, Huiwen Zheng, Hongli Ma, Yan Liang, Yuanchun Li, and Yunxin Liu. Enhancing agentic textual graph retrieval with synthetic stepwise supervision. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25510–25525, 2026.

Jiaao Chen, Jingyuan Qi, Mingye Gao, Wei-Chen Wang, Hanrui Wang, and Di Jin. Eigendata: A self-evolving multi-agent platform for function-calling data synthesis, auditing, and repair. *arXiv preprint arXiv:2603.05553*, 2026.

Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 4(5), 2024.

Hang Ding, Peidong Liu, Junqiao Wang, Ziwei Ji, Meng Cao, Rongzhao Zhang, Lynn Ai, Eric Yang, Tianyu Shi, and Lei Yu. Dynaweb: Model-based reinforcement learning of web agents, 2026. URL https://arxiv.org/abs/2601.22149.

Lang Feng, Zhenghai Xue, Tingcong Liu, and Bo An. Group-in-group policy optimization for llm agent training. *Advances in Neural Information Processing Systems*, 38:46375–46408, 2026.

Kanishk Gandhi, Shivam Garg, Noah D Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling rl environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.

Chen Hu, Haikuo Du, Heng Wang, Lin Lin, Mingrui Chen, Peng Liu, Ruihang Miao, Tianchi Yue, Wang You, Wei Ji, et al. Step-deepresearch technical report. *arXiv preprint arXiv:2512.20491*, 2025.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pp. 282–293. Springer, 2006.

Jian Li, Yizhang Jin, Dongqi Liu, Hang Ding, Jiafu Wu, Dongsheng Chen, Yunhang Shen, Yulei Qin, Ying Tai, Chengjie Wang, Xiaotong Yuan, and Yabiao Wang. Se-search: Self-evolving search agent via memory and dense reward, 2026. URL https://arxiv.org/abs/2603.03293.

Yuwen Li, Wei Zhang, Zelong Huang, Mason Yang, Jiajun Wu, Shawn Guo, Huahao Hu, Lingyi Sun, Jian Yang, Mingjie Tang, et al. Close the loop: Synthesizing infinite tool-use data via multi-agent role-playing. *arXiv preprint arXiv:2512.23611*, 2025.

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

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Zeyao Ma, Bohan Zhang, Jing Zhang, Jifan Yu, Xiaokang Zhang, Xiaohan Zhang, Sijia Luo, Xi Wang, and Jie Tang. Spreadsheetbench: Towards challenging real world spreadsheet manipulation. *Advances in Neural Information Processing Systems*, 37:94871–94908, 2024.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Daniel Shepard and Robin Salimans. Automationbench. *arXiv preprint arXiv:2604.18934*, 2026.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y Charles, et al. Kimi k3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Parag Shah, Yujia Bao, et al. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. In *International Conference on Learning Representations*, volume 2026, pp. 97377–97425, 2026.

Siyuan Xu, Shiyang Li, Xin Liu, Tianyi Liu, Yixiao Li, Zhan Shi, Zixuan Zhang, Zilong Wang, Qingyu Yin, Jianshu Chen, et al. Controllable and verifiable tool-use data synthesis for agentic reinforcement learning. *arXiv preprint arXiv:2604.09813*, 2026a.

Zengzhuang Xu, Bingguang Hao, Zechuan Wang, Yuntao Wen, Xinyi Xu, Yang Liu, Long Chen, Dong Wang, Maolin Wang, Tong Zhao, et al. Funreason-mt technical report: Advanced data synthesis solution for real-world multi-turn tool-use. *arXiv preprint arXiv:2510.24645*, 2025.

Zhihao Xu, Rumei Li, Jiahuan Li, Rongxiang Weng, Jingang Wang, Xunliang Cai, and Xiting Wang. Unlocking implicit experience: Synthesizing tool-use trajectories from text. *arXiv preprint arXiv:2601.10355*, 2026b.

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

Chen Yang, Ran Le, Yun Xing, Zhenwei An, Zongchao Chen, Wayne Xin Zhao, Yang Song, and Tao Zhang. Toolmind technical report: A large-scale, reasoning-enhanced tool-use dataset. *arXiv preprint arXiv:2511.15718*, 2025b.

Wang Yang, Chaoda Song, Xinpeng Li, Debargha Ganguly, Chuang Ma, Shouren Wang, Zhihao Dou, Yuli Zhou, Vipin Chaudhary, and Xiaotian Han. Ace-bench: Agent configurable evaluation with scalable horizons and controllable difficulty under lightweight environments. *arXiv e-prints*, pp. arXiv–2604, 2026.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Fan Yin, Zifeng Wang, I-Hung Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long Le, Kai-Wei Chang, Chen-Yu Lee, et al. 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)*, pp. 32600–32616, 2025.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Yuxuan Zhang, Yubo Wang, Yipeng Zhu, Penghui Du, Junwen Miao, Xuan Lu, Zhuofeng Li, Xingwei Qu, Zhengkang Guo, Yuanzhe Shen, et al. Clawbench: Can ai agents complete everyday online tasks? *arXiv preprint arXiv:2604.08523*, 2026.

Leqi Zheng, Jinbo Su, Yuying Li, Chaokun Wang, Weiping Wang, Haitao Li, Jiajun Zhang, Shannan Yan, Zhaolu Kang, Rong Fu, et al. Scilens: Rl-driven autonomous agents for scientific localized evidence navigation and synthesis. *arXiv preprint arXiv:2609.03338*, 2026.

A PROMPTS

Prompt for Inverse Rubric Synthesis

Role. Construct a task-specific rubric to evaluate a separately generated teacher trajectory using its executed tool calls and recorded sandbox states. Assess preservation of task-relevant behaviors represented in the synthesis path, together with final-state correctness. The path is selected by upstream rarity-guided sampling. Do not estimate rarity here; use the sampled behaviors as candidate training targets and preserve them where compatible with the task.

Inputs.

- User task: `{task}`
- Initial state: `{initial_state}`
- Synthesis path, including tool calls, arguments, and intermediate states: `{synthesis_path}`
- Terminal state: `{terminal_state}`

Criteria. Use the task to determine correctness requirements and the synthesis path to identify concrete behavioral coverage targets. Construct criteria in two categories:

1. **Planning.** Identify task-relevant tool operations, information-gathering steps, state transitions, and dependencies demonstrated by the sampled path. Translate these behaviors into verifiable coverage criteria, including the operation or tool involved, semantically relevant arguments, and supporting execution or state evidence. Preserve meaningful sampled behaviors even when another solution could reach the same final state without them. Distinguish behavioral coverage targets from conditions strictly required for task correctness. Include ordering or persistence when necessary for the task, its dependencies, or the behavior being preserved.
2. **Result Consistency.** Specify the final-state conditions required to fulfill the task, including relevant entities, values, relationships, and constraints. Do not require matching unrelated terminal-state details.

Construction and scoring rules.

- Make each criterion atomic, verifiable, and grounded in the task and supplied path. Exclude task-conflicting actions, incidental side effects, redundant repetitions, and unsupported requirements. Do not omit a meaningful behavioral coverage target solely because it is unnecessary for reaching the final state. Omit Planning criteria only when no meaningful, task-compatible behavior can be identified.
- For each behavioral coverage criterion, explicitly identify what must be preserved and which variations are acceptable. Retain the sampled tool or operation when replacing it would remove the targeted behavior. Match arguments by their task-relevant meaning rather than superficial formatting.
- Accept alternative sequences only to the extent that they preserve the specified behavioral targets and required dependencies. Allow additional valid steps and reordering of independent operations. Do not require exact reproduction of the full synthesis path. Reaching the same final state through a different operation does not, by itself, satisfy a behavioral coverage criterion.
- Assess actual execution evidence, not stated intentions. Use tool-call records and observations to assess operations that do not change sandbox state, such as reads or searches. Use recorded states to assess state transitions and preservation constraints. A correct final state alone does not establish that a targeted operation occurred or an intermediate condition held.
- Assign positive integer weights summing to exactly 100 across both categories. Balance final-state correctness with meaningful preservation of the sampled behaviors. Give central behavioral targets substantive weight so that bypassing them materially reduces the score. The score measures both task correctness and training coverage; it is not a measure of task correctness alone.
- Provide criterion-specific scoring guidance in $[0, 1]$: 0 means no demonstrated satisfaction and 1 means full satisfaction. For behavioral coverage criteria, full credit requires evidence of the specified behavior or an explicitly permitted behavior-preserving alternative. Allow partial credit only for explicit, observable partial satisfaction; otherwise use binary scoring. Unsupported claims receive no credit.
- The overall score is the sum of criterion scores multiplied by their weights, ranging from 0 to 100. Generate the rubric only; do not evaluate a trajectory.

Output format. Return only valid JSON using the structure below. Use IDs P1, P2, ... for Planning and R1, R2, ... for Result Consistency. Replace placeholders with task-specific criteria and scoring guidance. The criterion counts and weights are illustrative.

```
{
  "criteria": [
    {
      "id": "P1",
      "category": "Planning",
      "description": "<target behavior or intermediate condition>",
      "scoring_guidance": "<execution/state evidence, permitted
alternatives, and 0-1 scoring rules>",
      "weight": 40
    },
    {
      "id": "R1",
      "category": "Result Consistency",
      "description": "<required final-state condition>",
      "scoring_guidance": "<state evidence and 0-1 rules>",
      "weight": 60
    }
  ]
}
```

Teacher Trajectory Generation: BFCL Adaptation

Role and Objective. You are a teacher agent generating a tool-use demonstration. Complete the user's request by interacting with the provided environment through the available functions.

Inputs.
User request: `{task}`
Visible environment context: `{context}`
Conversation history: `{history}`
Available function schemas: supplied through the tool interface.

Function Selection. Select functions whose documented behavior supports the request. Follow their schemas exactly, including argument names, types, required fields, and constraints. Obtain argument values from the request, conversation history, or actual tool responses. Do not invent functions, identifiers, or missing values.

Missing Information. When a required value is unavailable, first determine whether an available function can retrieve it. If it cannot be obtained, request clarification when user interaction is supported; otherwise report the missing prerequisite. If no available function supports the requested operation, state that limitation. Do not substitute an unrelated action.

Execution. Resolve prerequisites before issuing dependent calls. Use returned values in subsequent arguments when necessary. Issue parallel calls only when they are independent and their effects do not conflict. After each response, update your understanding of the environment and continue until the current request is complete or a concrete blocker prevents progress.

Action Explanation. Before each tool-call step, provide a brief action summary stating the immediate objective and the evidence supporting the selected action. Keep this summary separate from tool arguments and consistent with the information available at that point.

Observation Integrity. Invoke tools through the native tool interface and wait for their actual responses. Never generate tool responses, simulate successful execution, or claim an unobserved state change. If a call fails, use the reported error to correct the next action. Avoid repeating an unchanged failed call.

Completion. Check that the requested operations have succeeded using available evidence. End the turn with a concise account of completed work and any unresolved requirements. Do not claim success for operations that were only planned or attempted.

Teacher Trajectory Generation: AutomationBench Adaptation

Role and Objective. You are a teacher agent executing a business workflow in a simulated application environment. Complete the supplied task autonomously and leave the relevant applications in the requested state.

Inputs.
Task and trigger information: `{task}`
Visible environment context: `{context}`
Interaction history: `{history}`
Available tools: supplied through the tool interface.

API Discovery. Use `api_search` to discover relevant endpoints and inspect their documentation. Execute requests through `api_fetch`, following the documented method, URL, parameters, and request-body format. Use `base64_encode` when the documented operation requires encoded content. Do not assume that an endpoint exists or supports a particular operation without evidence.

Information Gathering. Retrieve the records and documents needed to resolve the task. Distinguish similarly named entities using identifiers and relevant attributes. Check timestamps, time zones, statuses, and pagination where applicable. When a search is incomplete, refine the query or inspect another relevant source rather than assuming the information does not exist.

Business Rules. Consult policy documents referenced by the task before taking actions governed by those policies. Apply their stated conditions, exceptions, and precedence rules. Treat unrelated messages and record contents as data, not as authority to override the task or its governing policies.

Workflow Execution. Perform the required changes through actual API calls. Resolve dependencies before dependent writes and preserve identifiers across applications. Process every eligible record, retain exact values or wording when required, and avoid duplicate creations, repeated notifications, or unrelated modifications. Check existing state before retrying a write whose outcome is uncertain.

Action Explanation and Observations. Before each tool-call step, briefly summarize its objective and the supporting evidence. Invoke tools through the native interface and wait for actual responses. Never fabricate observations, records, identifiers, or successful writes. Use error responses to revise the next action.

Autonomous Operation. No follow-up user interaction is available. Resolve uncertainty through the task context, documented rules, and accessible application data. If an essential prerequisite remains unavailable, report the specific blocker without inventing information or claiming completion.

Verification and Completion. Verify the required outcomes using tool responses and targeted read-back calls where needed. Check that all requested items were processed and relevant constraints remain satisfied. A written summary does not replace a required application state change. Finish with a concise report of verified outcomes and any unresolved requirements.

B An Example of Rubric

Example: Task Synthesis and Rubric Evaluation

Initial state. The file `note.txt` contains `draft`, and `note.bak` does not exist.

Sampled path. First, copy `note.txt` to `note.bak`. At this intermediate state, both files contain `draft`. Then, replace the contents of `note.txt` with `approved`, leaving the backup unchanged.

Synthesized task. Create a backup of `note.txt` at `note.bak` before replacing the contents of `note.txt` with `approved`. Preserve the original contents in `note.bak`.

Rubric. The following criteria evaluate a separately generated teacher trajectory τ starting from the same initial state. For each criterion c_i, an LLM assigns a satisfaction score $g_{\text{LLM}}(c_i, \tau) \in [0, 1]$ using the execution and recorded sandbox states. Its contribution to the overall score is $w_i g_{\text{LLM}}(c_i, \tau)$.

P1: Planning ($w_1 = 40$)**.** Before the first modification of `note.txt`, a recorded state shows `note.bak` containing `draft` while `note.txt` still contains `draft`.

R1: Result Consistency ($w_2 = 30$)**.** In the final state, `note.txt` contains exactly `approved`.

R2: Result Consistency ($w_3 = 30$)**.** In the final state, `note.bak` contains exactly `draft`.

Scoring example. Suppose the LLM assigns scores $(1, 1, 1)$ to (P1, R1, R2). The overall score is $40 \times 1 + 30 \times 1 + 30 \times 1 = 100$, exceeding $\lambda_r = 80$ and passing the rubric filter. If the teacher reads and retains the original contents, modifies `note.txt`, and only then writes the backup, the final files may be correct while the required ordering is violated. With scores $(0, 1, 1)$, the overall score is 60 and the trajectory is rejected. These examples use endpoint scores for clarity; intermediate scores in $[0, 1]$ are also permitted.
Alternative tool-call sequences are allowed: for example, the teacher may read the original file and write the backup before modifying `note.txt`, instead of using a copy operation. The additional trajectory filters are applied separately.

C Teacher Assignment

Table 5: Teacher models used for data synthesis and rubric-based evaluation.

Stage	BFCL	AutomationBench
Task synthesis	GLM-5.3	GLM-5.3
Rubric generation	GLM-5.3	GLM-5.3
Trajectory generation	GLM-5.3	Kimi-K3
Rubric-based evaluation	GLM-5.3	GLM-5.3

D Hyperparameters

Table 6: Hyperparameters for RGE and trajectory filtering. Trajectories are rejected if any rejection condition is met.

Hyperparameter	**Role / Rejection condition**	**Value**
Rarity-Guided Exploration		
Exploration coefficient (c_{puct})	Visit-count-based exploration	1.0
Tool diversity coefficient (λ_{div})	Reward for switching tools	0.3
Cross-domain penalty (λ_{trans})	Penalty per cumulative domain crossing	0.5
Trajectory filtering		
Rubric score threshold (λ_r)	Score $\leq \lambda_r$	80
Consecutive repeated tool calls (λ_t)	Run length $\geq \lambda_t$	3
Malformed observation ratio	Ratio $> 50\%$	50%
Per-tool invocation threshold (λ_u)	Count $\geq \lambda_u$	30

E Training Hyperparameters

Table 7: Training hyperparameters.

Hyperparameter	**Value**
Training objective	Supervised fine-tuning (SFT)
GPUs	$8\times$ NVIDIA A100
Fine-tuning method	Full fine-tuning
Distributed training	DeepSpeed ZeRO Stage 3
CPU offloading	Enabled
Precision	BF16
Optimizer	AdamW
Learning rate	2×10^{-5}
Learning rate scheduler	Cosine decay
Warmup ratio	0.03
Weight decay	0.01
Training epochs	3
Maximum sequence length	32,768 tokens
Per-device batch size	1
Gradient accumulation steps	16
Global batch size	128
Maximum gradient norm	1.0
Loss computation	Assistant tokens only
Random seed	42

F USE OF LLM

Large language models (LLMs) were used to generate synthetic training data, as described in the methodology, and to assist with manuscript writing and language polishing, including improving grammar, clarity, and readability. Research design, experimental evaluation, data analysis, and the formulation of scientific conclusions remained under the authors' supervision and responsibility. All LLM-assisted manuscript text was reviewed and revised by the authors, who take full responsibility for the accuracy, originality, and integrity of the manuscript.