MultiHop-Tool-N1: An LLM-World-Model RL Environment and Judge-Based Reward for Long-Horizon Tool Use

Anonymous Authors

Abstract

Language models are increasingly expected to solve problems by coordinating sequences of tool calls rather than issuing a single function invocation. We study this setting as *multi-hop tool calling*, where a model must decompose a request, choose tools in the correct order, and carry intermediate results across steps. For small models, the central challenge is not just data scarcity but also how to supply a usable training signal for long-horizon tool chains.

We present a training setup centered on two components: an LLM-based world-model environment that executes tool calls during RL, and a judge-based reward that scores format correctness, tool selection, chaining, and order. We pair this setup with a new multi-hop extension of Tool-N1 [10] containing 3/6/9-hop trajectories and reasoning traces, and train Qwen2.5 3B and 7B models with supervised fine-tuning (SFT), reinforcement learning with verifiable rewards (RLVR), and their combination using Prime-RL [5] and Verifiers [1].

On ToolHop [9], the best 3B model improves from 1.91% to 10.05%, while the best 7B model reaches 24.62% on ToolHop Direct, making it competitive with GPT-4o under our evaluation protocol and closer to Claude 3.5 Sonnet than prior sub-10B baselines. A key finding is that reward design matters as much as data design: sparse binary rewards are easy to game, whereas judge-based rewards paired with interleaved reasoning produce substantially stronger long-horizon behavior.

Keywords

Tool Learning, Reinforcement Learning, Small Language Models, Multi-hop Reasoning, GRPO

1 Introduction

Tool use turns language models from static predictors into sequential decision makers. In multi-hop settings, success depends on more than picking a valid function: the model must decompose a query into subgoals, call heterogeneous tools in the right order, and reuse intermediate outputs in later arguments. This makes multi-hop tool use a practical instance of long-horizon planning over tool-induced state transitions. Figure 1 summarizes the end-to-end data-generation and training pipeline used in our study.

The challenge is especially acute for small language models. In realistic deployments, cost, latency, and privacy often require models below 10B parameters, yet these models remain weak at chained tool use. On ToolHop [9], our base Qwen2.5-3B model reaches only 1.91%, and even much larger open models remain below strong closed-source baselines.

Our core claim is that long-horizon tool learning depends critically on the *training interface*, not only on the dataset. To make RL workable in this setting, we pair an LLM world-model environment for tool execution with a judge-based reward that scores four aspects of each trajectory: formatting, tool choice, chaining of intermediate results, and global order. We then study this setup on **MultiHop-Tool-N1**, an 8,123-example extension of Tool-N1 with 3/6/9-hop trajectories and reasoning traces, comparing SFT-only, RL-only, SFT+RL with single-hop supervision, and SFT+RL with multi-hop supervision on Qwen2.5 3B and 7B models.

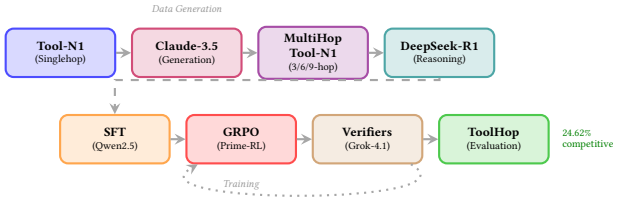

Figure 1: End-to-end pipeline overview. Top: data generation extends Tool-N1 to multi-hop scenarios using Claude-3.5-Sonnet, then augments them with reasoning traces from DeepSeek-R1. Bottom: training combines SFT on multi-hop data with GRPO using Grok-4.1 for tool execution and judge-based reward evaluation.

Our results support three conclusions. First, matched-complexity supervision matters: multi-hop SFT is consistently more useful than single-hop SFT for long-horizon tool calling. Second, reward design matters in its own right: sparse binary rewards invite reward hacking, while judge-based rewards produce much better generalization. Third, the best recipe is scale dependent: the 3B model benefits most from progressive SFT+RL, while the 7B model is stronger under RL and improves further with multi-hop SFT initialization. Full prompts, ablations, and extended benchmark results are deferred to the appendix.

2 Background

Most prior work on tool learning emphasizes *single-hop* function calling: choosing one tool and emitting a schema-correct invocation [3, 4, 6]. Even when work considers multi-step behavior, it often focuses on repeated use of a narrow tool family such as search [2, 8]. In contrast, heterogeneous multi-hop tool use requires three additional capabilities: decomposing a request into dependent subgoals, chaining outputs from earlier calls into later arguments, and remaining robust across different schemas and interfaces.

Our setting targets this underexplored regime. We study long-horizon tool calling with heterogeneous tools, train on 3/6/9-hop examples, and evaluate on ToolHop chains of up to 14 hops [9]. To make RL feasible despite sparse end-task rewards, we use an LLM-based world model that executes tool calls and returns structured observations, enabling verifiable training without step-level human annotation.

ACM Conference, City, Country
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Figure 2: Interleaved thinking and tool calling for a 3-hop query. Each hop follows a `<think>` → `<tool_call>` → `[Result]` pattern. Dashed arrows show output chaining: the host country "France" from Hop 1 becomes the input to Hop 2, and GDP values from Hop 2 feed into Hop 3's calculation. This dependency structure causes errors to compound over long horizons.

3 Methods

3.1 Training Data

We extend Nemotron Tool-N1 [10] into a multi-hop corpus with 8,123 validated examples: 5,554 3-hop, 1,569 6-hop, and 1,000 9-hop trajectories. Each example requires sequential tool use in which later calls depend on outputs from earlier ones. For SFT, we augment these trajectories with reasoning traces generated from a stronger model so that the target model observes explicit decomposition, tool selection, and chaining behavior. We compare this **Multi-Hop** corpus with a **Singlehop** SFT set derived from the original Tool-N1 data.

3.2 Training Recipe

We study four configurations on Qwen2.5 3B and 7B instruction-tuned backbones: SFT-only, RL-only, SFT+RL with single-hop supervision, and SFT+RL with multi-hop supervision. RL uses GRPO [7] in Prime-RL. Model outputs follow an interleaved format in which a short reasoning span inside `<think>` tags precedes each `<tool_call>`; this forces explicit decomposition before action selection.

3.3 Environment and Reward

Our main methodological contribution is the RL interface itself: an LLM-based tool-execution environment paired with a judge-based reward for long-horizon tool trajectories. We use an LLM simulator rather than real APIs because RL requires repeated interactions at scale, whereas real tool backends are brittle, rate-limited, and hard to normalize across training runs. The simulator gives us a unified interface over heterogeneous tools while preserving the dependency structure that makes multi-hop tool use difficult.

During RL, Grok-4.1-fast serves as a deterministic tool executor: given a tool specification and arguments, it returns a direct result that is fed back to the policy model as the next observation. We run the executor at temperature 0 so that identical tool calls produce identical observations across rollouts; this reduces variance and makes reward attribution easier in long trajectories. The executor prompt is intentionally minimal so the returned value resembles an API response rather than an explanatory assistant message.

For reward evaluation, we again use Grok-4.1-fast as an LLM judge. Instead of exact-match rewards alone, the judge scores each trajectory on four criteria. **Format correctness** checks whether each step follows the required `<think>` then `<tool_call>` structure. **Tool selection** checks whether the chosen tool matches the expected subproblem. **Chaining correctness** checks whether arguments in later calls actually use information produced by earlier calls. **Order correctness** checks whether the overall sequence of tools respects the intended dependency structure. This produces a denser signal than binary success/failure and is better suited to long-horizon tool learning.

Using the same model as both executor and judge is a deliberate but imperfect design choice. It keeps the environment and reward function internally consistent and makes training practical, but it also introduces a possible source of circularity: the judge may favor behaviors that align with the executor's own conventions rather than with real external tools. We therefore treat this setup as a useful training interface rather than a faithful measurement of real-world API fidelity. Full prompts, formulas, and hyperparameters are provided in the appendix.

4 Experiments

We evaluate on ToolHop [9], a benchmark for multi-hop tool use with heterogeneous APIs. Our main metric is answer accuracy on **ToolHop Direct**, where the model must identify and execute the necessary tool chain to produce the final answer. We report closed-source baselines from the benchmark paper alongside open-source baselines and our 3B/7B models.

Unless otherwise stated, all reported results use the judge-based reward and interleaved reasoning format described above. Complete hyperparameters, prompt templates, reward ablations, and results on the Mandatory and Free ToolHop settings appear in the appendix.

5 Results

5.1 Main Results

Table 1 shows that targeted multi-hop training substantially improves long-horizon tool use. For 3B, the best model reaches 10.05%, a 5.3× gain over the 1.91% base model. For 7B, multi-hop SFT+RL reaches 24.62%, making it competitive with GPT-4o under our evaluation protocol and substantially improving over the base model.

Model	Accuracy (%)
Closed-Source	
Claude 3.5 Sonnet	27.14
GPT-4o	23.12
Open-Source (>30B)	
Qwen2.5-72B-Instruct	17.89
Qwen2.5-32B-Instruct	20.00
LLaMA3.1-70B-Instruct	18.79
Ours (3B)	
Qwen2.5-3B-Instruct (Base)	1.91
+ SFT (Singlehop)	2.40
+ SFT (Multi-Hop)	4.90
+ RL (GRPO)	1.81
+ SFT+RL (Singlehop)	8.14
+ SFT+RL (Multi-Hop)	10.05
Ours (7B)	
Qwen2.5-7B-Instruct (Base)	8.84
+ SFT (Singlehop)	4.20
+ SFT (Multi-Hop)	8.10
+ RL (GRPO)	20.60
+ SFT+RL (Singlehop)	10.05
+ SFT+RL (Multi-Hop)	**24.62**†

Table 1: ToolHop Direct accuracy. † denotes the best result among models under 10B parameters.

Two patterns are especially important. First, **data complexity must match task complexity**: at both scales, multi-hop SFT is more effective than single-hop SFT, and for 7B the single-hop initialization is actively harmful relative to RL alone (10.05% vs. 20.60%). Second, **RL benefits from structured intermediate supervision**: judge-based rewards and interleaved `<think>` blocks give large gains over weaker variants. In our ablations, judge+think improves the 7B model from 12.56% to 24.62% and the 3B model from 3.52% to 10.05%.

A particularly important finding is **reward hacking under sparse binary rewards**. In Appendix A.2, the **Binary + No Think** variant reaches relatively high training reward while generalizing poorly at test time (12.56% for 7B), indicating that the model can exploit superficial matching signals without learning reliable multi-hop chaining. In contrast, **Judge + Think** reaches 24.62% by rewarding partial but structurally correct progress.

6 Conclusion

We study how to improve long-horizon tool calling in small language models. Across Qwen2.5 3B and 7B, RL works best when paired with supervision that already reflects multi-hop composition. Multi-hop SFT provides scaffolding for decomposition and chaining, while judge-based RL further improves execution over long trajectories.

The resulting 7B model reaches 24.62% on ToolHop Direct, making it competitive with GPT-4o under our evaluation protocol while remaining far smaller. A key remaining limitation is strategic tool-use judgment in ToolHop Free, where the model must decide whether tools are needed at all: our 7B SFT+RL (Multi-Hop) model reaches 9.45%, well below GPT-4o and Claude 3.5 Sonnet (Appendix A.4). This gap likely reflects a training setup focused on settings where tool use is expected.

Limitations. Our current setup also has four important caveats. First, the judge and executor are both LLM-based, so reward quality may partly reflect circular model preferences rather than only task correctness. Second, our main reported numbers are from single-seed runs, so variance is not yet characterized. Third, we study a single benchmark, ToolHop, and do not yet know how well the findings transfer. Fourth, the fidelity of the world-model tool simulator is not independently measured against real API behavior.

A natural next step is to train on mixed examples that require choosing between tool use and direct answering, and to extend evaluation beyond pass@1 to pass@k for longer chains. Additional directions include scaling to longer reasoning chains, parallel tool execution, and real API environments with latency and error handling.

References

[1] William Brown. 2025. Verifiers: Environments for LLM Reinforcement Learning. https://github.com/PrimeIntellect-ai/verifiers.

[2] Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-R1: Training LLMs to Reason and Leverage Search Engines with Reinforcement Learning. *arXiv preprint arXiv:2503.09516* (2025). https://arxiv.org/abs/2503.09516

[3] Shishir G. Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E. Gonzalez. 2025. The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. In *Proceedings of the International Conference on Machine Learning (ICML)*. https://openreview.net/pdf?id=2GmDdhBdDk

[4] Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large Language Model Connected with Massive APIs. *arXiv preprint arXiv:2305.15334* (2023). https://arxiv.org/abs/2305.15334

[5] Prime Intellect. 2025. Prime-RL: Async RL Training at Scale. https://github.com/PrimeIntellect-ai/prime-rl.

[6] Yujia Qin, Shihao Liang, Liang Ye, et al. 2024. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/2307.16789

[7] Zhihong Shao et al. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *arXiv preprint arXiv:2402.03300* (2024). https://arxiv.org/abs/2402.03300

[8] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*. https://arxiv.org/abs/1809.09600

[9] Junjie Ye, Zhengyin Du, Xuesong Yao, Weijian Lin, Yufei Xu, Zehui Chen, Zaiyuan Wang, Sining Zhu, Zhiheng Xi, Siyu Yuan, Tao Gui, Qi Zhang, Xuanjing Huang, and Jiecao Chen. 2025. ToolHop: A Query-Driven Benchmark for Evaluating Large Language Models in Multi-Hop Tool Use. *arXiv preprint arXiv:2501.02506* (2025). https://arxiv.org/abs/2501.02506

[10] Shaokun Zhang, Yi Dong, Jieyu Zhang, Jan Kautz, Bryan Catanzaro, Andrew Tao, Qingyun Wu, Zhiding Yu, and Guilin Liu. 2025. Nemotron-Research-Tool-N1: Exploring Tool-Using Language Models with Reinforced Reasoning. *arXiv preprint arXiv:2505.00024* (2025). https://arxiv.org/abs/2505.00024

Tool Execution Prompt

System:

"You are executing the tool '<*tool_name*>' with the given arguments. Return ONLY the direct result value. No explanation."

User:

```
{"tool": "<tool_name>", "arguments": <arguments_dict>}
```

Figure 3: Tool execution prompt for Grok-4.1-fast. The minimalist prompt design ensures the tool executor produces concise, directly usable outputs that mimic real API responses without meta-commentary. Temperature=0 ensures deterministic tool responses across training runs.

A Supplementary Materials

A.1 Hyperparameters for RL Training

Table 2 summarizes the hyperparameters used for RL training with Prime-RL and the Verifiers framework. We perform hyperparameter tuning via grid search to ensure stable and efficient training, focusing primarily on the learning rate, KL coefficient, entropy coefficient, number of rollouts, and batch size. Different learning rates are used for 3B and 7B models to account for model capacity differences, while other hyperparameters remain constant across both model sizes.

Hyperparameter	Qwen2.5-3B	Qwen2.5-7B
Batch Size	32	32
Learning Rate	5e-7	2e-7
KL Coefficient	1e-3	1e-3
Entropy Coefficient	0	0
Rollout Number	4	4
Max Prompt Length	4096	4096
Max Response Length	8192	8192
Temperature	0.7	0.7
Max Steps	200	200

Table 2: Hyperparameters used for RL training with Prime-RL and the Verifiers framework.

Rollout Sampling and Evaluation Protocol. During GRPO training, we sample 4 completions per prompt (Rollout Number = 4) to estimate group-relative advantages. This follows the standard GRPO formulation where the group average reward serves as the baseline for advantage computation, eliminating the need for a separate value model [7].

For evaluation, we report **pass@1** accuracy using greedy decoding (temperature = 0), which measures the model's ability to produce correct answers in a single attempt. This is the standard evaluation protocol for ToolHop [9] and enables direct comparison with prior work.

A.2 Ablations

We compare our judge-based reward against a binary exact-match baseline that assigns a reward of 1 only when the predicted tool call exactly matches the ground-truth tool name and all argument key-value pairs (after JSON parsing), and 0 otherwise. This baseline mirrors prior tool-calling verification setups but provides no partial credit for correct intermediate steps.

Figure 4 compares binary per-step rewards against judge-based rewards, with and without interleaved thinking, for both Qwen2.5-3B and Qwen2.5-7B models.

The binary reward function assigns 1 if the predicted tool call exactly matches ground-truth, and 0 otherwise. This sparse signal provides no gradient for partial correctness—a model that selects the correct tool with one wrong argument receives the same reward as one that fails entirely. For multi-hop reasoning, the probability of achieving perfect matches across all steps diminishes exponentially with chain length, making learning from binary signals increasingly difficult.

Our results reveal reward hacking through superficial matching. The **Binary + No Think** configuration achieves high training rewards (approximately 0.20 at 7B scale), yet yields only 12.56% test accuracy compared to 24.62% for **Judge + Think**. This gap indicates the model exploits the reward signal without learning genuine reasoning. Without the thinking requirement, the model emits tool calls that match ground-truth outputs through memorization or pattern matching, receiving full credit despite failing to generalize. We also observe models trained with binary rewards entering degenerate loops, repeating equivalent tool calls without progress since the sparse reward provides no signal for partial advancement.

At 3B scale, the effect is more pronounced. **Binary + Think** (3.52%) barely exceeds baseline, as the smaller model cannot discover valid tool-calling patterns from sparse reward alone. **Judge + Think** (10.05%) achieves 3× higher accuracy by providing dense feedback that guides limited-capacity models. This suggests granular rewards become increasingly important as model capacity decreases.

A.2.1 Judge-Based Reward Evaluation Prompt.

A.3 Multi-Hop Dataset Generation Methodology

We construct the MultiHop-Tool-N1 dataset by extending single-turn examples from the Nemotron-Research-Tool-N1 dataset [10] into multi-hop reasoning chains. This section details the complete generation pipeline for reproducibility.

A.3.1 Generation Pipeline. Our dataset generation follows a three-stage pipeline:

Stage 1: Multi-Hop Scenario Generation. Starting from each single-turn example in Nemotron-Tool-N1 (which contains a question, available tools, and expected single tool call), we prompt Claude-3.5-Sonnet to generate a new question requiring 3, 6, or 9 sequential tool calls to solve. Figure 6 presents the complete prompt template.

Stage 2: Reasoning Trace Augmentation. The scenarios generated by Claude-3.5-Sonnet contain tool-calling trajectories but often have brief reasoning. To enrich the `<think>` blocks, we pass each

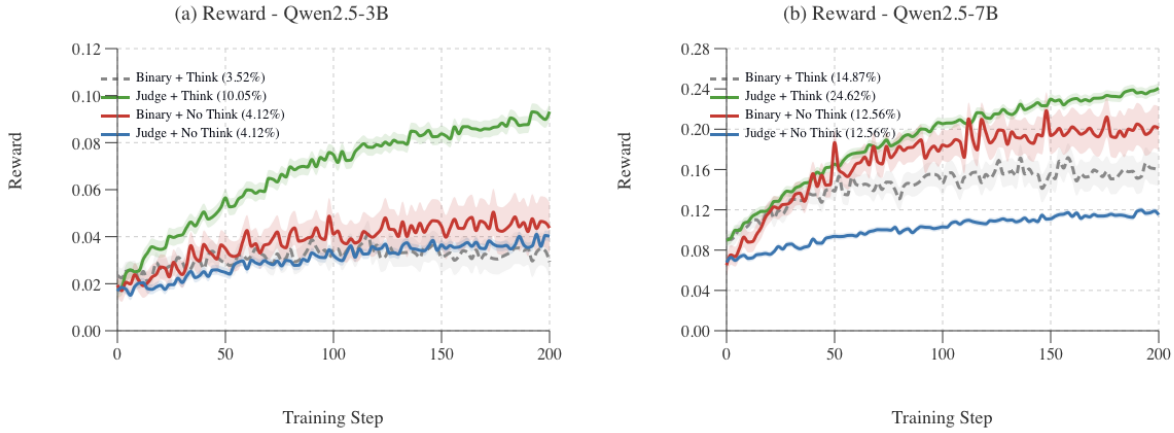

Figure 4: Reward ablation results comparing the judge-based reward against the binary exact-match baseline. Parenthetical percentages in the legend denote final ToolHop Direct test accuracy for each variant.

generated scenario through DeepSeek-R1 with a prompt requesting detailed step-by-step reasoning explaining: (1) why the current tool is needed, (2) how its output will be used in subsequent steps, and (3) how it contributes to answering the original question. The enriched reasoning traces replace the original `<think>` content while preserving the tool-calling structure.

Stage 3: Validation. Each generated example undergoes both structural and semantic validation:

- **Structural checks**: Verify exactly n tool calls, proper XML formatting, correct interleaving of `<think>` and `<tool_call>` blocks, and valid tool names/parameters.
- **Semantic validation**: Execute the generated tool sequence using our LLM-based simulator (Section 3.3) and verify that: (a) each tool call receives required parameters, (b) outputs from step i are referenced in step $i + 1$, and (c) the final answer is achievable. Examples failing validation are discarded.

This pipeline yielded 5,554 validated 3-hop, 1,569 validated 6-hop, and 1,000 validated 9-hop examples from approximately 12,000 candidates (67.7% validation pass rate).

A.4 Full ToolHop Results

Table 3 presents the complete performance comparison on the ToolHop benchmark across all three scenarios (Direct, Mandatory, and Free) for closed-source baselines and our trained models.

Judge-Based Reward Evaluation Prompt for Grok-4.1-fast

Configuration:
- Judge Model: `x-ai/grok-4.1-fast:free`
- Tool Executor: `x-ai/grok-4.1-fast:free`
- Evaluation: Per-step binary scoring across four criteria

1. FORMAT CORRECTNESS (per step):
- Does the step have `<think>...</think>` BEFORE `<tool_call>...</tool_call>`?
- Is the reasoning meaningful (not empty)?

Step score: 1 if both present and correct order, 0 otherwise

2. TOOL NAME CORRECTNESS (per step):
- Does the tool name match the expected tool for this step?
- Step 1 should call tool 1 from expected sequence, Step 2 should call tool 2, etc.

Step score: 1 if correct tool name, 0 otherwise

3. CHAINING CORRECTNESS (per step, starting from step 2):
- Does the tool call use information from the PREVIOUS tool's result?
- Check if arguments reference or incorporate previous `<result>...</result>` values

Step score: 1 if properly chained, 0 otherwise (step 1 gets 1 by default)

4. ORDER CORRECTNESS:
- Are tools called in the exact sequence specified in expected tools?
- Verify that tool_i appears before tool_{i+1} for all i

Score: 1 if all tools in correct order, 0 if any out of order

SCORING FORMULA:
Each step evaluated: format (0/1) + tool_name (0/1) + chaining (0/1) = max 3 points per step

$$r_{\text{judge}} = \frac{\sum_{\text{steps}} (\text{format} + \text{tool\_name} + \text{chaining})}{\text{num\_steps} \times 3}$$

If global order is incorrect, apply penalty:

$$r_{\text{final}} = r_{\text{judge}} \times \begin{cases} 1.0 & \text{if order correct} \\ 0.5 & \text{otherwise} \end{cases}$$

PROMPT OUTPUT INSTRUCTION:
"Analyze each step briefly, explaining your evaluation for each criterion. Then output ONLY a single number between 0.0 and 1.0 representing the overall quality of the multi-hop reasoning chain."

Figure 5: Complete judge-based reward evaluation prompt for Grok-4.1-fast. The judge evaluates each step across four criteria: format correctness (proper `<think>` before `<tool_call>`), tool name correctness (matches expected tool at each step), chaining correctness (uses previous results in arguments), and order correctness (tools called in correct sequence). The final reward is computed as the average of all step scores with a 0.5 penalty if global tool order is incorrect. This extends Nemotron-Tool-N1's binary format and tool-call checking with additional criteria for teaching function chaining in multi-hop scenarios.

Multi-Hop Scenario Generation Prompt

Configuration:
- Generation Model: `Claude-3.5-Sonnet`
- Temperature: 0.9
- Target: Generate [N]-hop scenarios where N ∈ {3, 6, 9}

TASK:
You are an expert in composing multi-step reasoning chains using function calls. You are given:
- An original single-turn question and its solution
- A set of available tools (functions) in JSON format (same format as Nemotron-Tool-N1)

Generate a NEW question that requires exactly **[N] sequential tool calls** to solve.

REQUIREMENTS:
1. Break down the solution into [N] sub-questions, each requiring exactly one tool
2. Chain dependencies: output from tool call i must feed as input to tool call $i + 1$
3. Use interleaved format: `<think>reasoning</think>` followed by `<tool_call>[...]</tool_call>`
4. Ensure all tool names come from the provided tool set
5. Make the reasoning chain logically coherent and realistic

EXAMPLE (3-hop scenario):
Original question: "What is the flight schedule for route XYZ on 2022-04-30?"
Original tool: `ensure-flight(route_id, date)`

Generated multi-hop question: "What is the average delay for flights from the airport where the company with phone +1-555-0100 is headquartered to New York on 2022-04-30?"

Solution trajectory:
<think>First, I need to find which company has phone +1-555-0100 to determine headquarters.</think>
<tool_call>[{"name": "Get Company Details", "arguments": {"internationalNumber": "+1-555-0100", "countryCode": "US"}}]</tool_call>
[Result: Headquarters in Chicago]

<think>Now I know headquarters is Chicago, I need the route ID for Chicago to New York.</think>
<tool_call>[{"name": "find-route", "arguments": {"origin": "Chicago", "destination": "New York"}}]</tool_call>
[Result: route_id = "CHI-NYC-001"]

<think>With route CHI-NYC-001, I can retrieve the flight schedule for 2022-04-30.</think>

<tool_call>[{"name": "ensure-flight", "arguments": {"route_id": "CHI-NYC-001", "date": "2022-04-30"}}]</tool_call>

OUTPUT FORMAT:
- New question: [your generated question]
- Expected tool sequence: [tool_1, tool_2, ..., tool_N]
- Solution trajectory: [complete trajectory with <think> and <tool_call> blocks]

Figure 6: Complete prompt template for multi-hop scenario generation using Claude-3.5-Sonnet. The prompt takes single-turn examples from Nemotron-Tool-N1 and instructs the model to create chained reasoning scenarios requiring multiple sequential tool calls with explicit dependencies between steps. Each generated scenario includes interleaved thinking and tool-calling blocks following the format from the original dataset.

Model	Direct	Mandatory	Free	Avg.
Closed-Source Baselines				
Claude 3.5 Sonnet	27.14	39.90	45.23	37.42
GPT-4o	23.12	42.50	44.80	36.81
Gemini 1.5 Flash	18.59	29.35	32.76	26.90
Qwen2.5-3B-Instruct (Ours)				
Qwen2.5-3B-Instruct (Base)	1.91	1.61	1.21	1.57
Qwen2.5-3B + SFT (Singlehop)	2.40	4.90	5.40	4.25
Qwen2.5-3B + SFT (Multi-Hop)	4.90	8.50	7.20	6.87
Qwen2.5-3B + RL (GRPO)	1.81	1.81	1.41	1.68
Qwen2.5-3B + SFT+RL (Singlehop)	8.14	13.87	0.40	7.47
Qwen2.5-3B + SFT+RL (Multi-Hop)	10.05	12.66	12.76	**11.82**
Qwen2.5-7B-Instruct (Ours)				
Qwen2.5-7B-Instruct (Base)	8.84	7.54	3.12	6.50
Qwen2.5-7B + SFT (Singlehop)	4.20	2.30	5.30	3.95
Qwen2.5-7B + SFT (Multi-Hop)	8.10	6.80	4.50	6.47
Qwen2.5-7B + RL (GRPO)	20.60	9.20	7.40	12.40
Qwen2.5-7B + SFT+RL (Singlehop)	10.05	7.14	2.41	6.53
Qwen2.5-7B + SFT+RL (Multi-Hop)	**24.62**	12.16	9.45	**15.41**

Table 3: Full performance comparison on ToolHop benchmark across all three scenarios (Direct, Mandatory, and Free). Results show percentage accuracy. Bold indicates best performance among our trained models. Our 7B SFT+RL (Multi-Hop) achieves 24.62% on Direct, making it competitive with GPT-4o (23.12%) under our evaluation protocol. However, our models show lower performance on Mandatory and Free scenarios compared to closed-source models, reflecting our training focus on direct tool-use patterns rather than strategic tool selection.