Nemotron-Research-Tool-N1: Tool-Using Language Models with Reinforced Reasoning

Shaokun Zhang , Yi Dong , Jieyu Zhang , Jan Kautz , Bryan Catanzaro , Andrew Tao , Qingyun Wu , Zhiding Yu , Guilin Liu *

Posted Date: 30 April 2025

doi: 10.20944/preprints202504.2471.v1

Keywords: Large Language Models; Reinforcement Learning; LLM Reasoning

 doi:10.20944/preprints202504.2471.v1

Article

Nemotron-Research-Tool-N1: Tool-Using Language Models with Reinforced Reasoning

Shaokun Zhang [1,2], Yi Dong [1], Jieyu Zhang [3], Jan Kautz [1], Bryan Catanzaro [1], Andrew Tao [1], Qingyun Wu [2], Zhiding Yu [1] and Guilin Liu [1,*]

[1] NVIDIA
[2] Pennsylvania State University
[3] University of Washington
* Correspondence: guilinl@nvidia.com

Abstract: Enabling large language models with external tools has become a pivotal strategy for extending their functionality beyond text generation tasks. Prior work typically enhances tool-use abilities by either applying supervised fine-tuning (SFT) to enforce tool-call correctness or distilling reasoning traces from stronger models for SFT. However, both approaches fall short, either omitting reasoning entirely or producing imitative reasoning that limits generalization. Inspired by the success of DeepSeek-R1 in eliciting reasoning through rule-based reinforcement learning, we develop the Nemotron-Research-Tool-N1 series of tool-using language models using a similar training paradigm. Instead of restrictively supervising intermediate reasoning traces distilled from stronger models, Nemotron-Research-Tool-N1 is optimized with a binary reward that evaluates only the structural validity and functional correctness of tool invocations. This lightweight supervision allows the model to autonomously internalize reasoning strategies, without the need for annotated reasoning trajectories. Experiments on the BFCL and API-Bank benchmarks show that Nemotron-Research-Tool-N1-7B and Nemotron-Research-Tool-N1-14B, built on Qwen-2.5-7B/14B-Instruct, achieve state-of-the-art results, outperforming GPT-4o on both evaluations. We will release the code in https://github.com/NVlabs/Tool-N1

Keywords: large language models; reinforcement learning; LLM Reasoning

1. Introduction

In recent years, equipping large language models (LLMs) with external tools or functions has attracted significant attention, demonstrating impressive performance across a wide range of domains [1,29]. For instance, LLMs augmented with search engines can answer questions that go beyond their training data [10], while those equipped with Python interpreters are capable of solving complex mathematical problems by leveraging external libraries [38]. These tools effectively enable LLMs to operate beyond purely textual tasks, substantially extending their functional capabilities.

To enhance LLMs tool-calling capability, existing research has primarily focused on synthesizing large volumes of tool-use trajectories using advanced language models [16,43,47], followed by supervised fine-tuning (SFT) on the generated data. Although step-by-step reasoning has been shown to play a critical role in enabling LLMs to solve complex tasks [37], these synthetic datasets often lack explicit reasoning steps. Consequently, current pipelines typically supervise only the tool calls step, without providing guidance on the underlying reasoning process during training. In many cases, reasoning is either omitted entirely in training stage or deferred to the inference stage via prompting techniques [41,42]. While one workaround is to distill reasoning trajectories from advanced models and then train student models via SFT [2], this approach often yields pseudo reasoning: models merely learn to mimic surface-level patterns from expert demonstrations without truly internalizing the decision-making process [2].

 doi:10.20944/preprints202504.2471.v1

Most recently, DeepSeek R1 [6] has demonstrated that simple rule-based reinforcement learning can substantially improve the complex reasoning capabilities of LLMs [18,19,31]. In the R1-style RL training, rewards are assigned based on the correctness of the final answer and reasoning format, allowing the model to learn the intermediate reasoning steps without explicit reasoning supervision. This paradigm naturally inspired us to explore the utilization of R1-style RL in training tool call models, for the following reasons: **(1)** Rule-based reward design provides interpretable supervision signals by verifying the correctness of tool calls. In contrast to SFT, which relies on exact next-token prediction and enforces strict output matching, RL allows for more flexibility. Semantically equivalent tool calls, such as those with reordered arguments, can still be rewarded correctly. This enables models to generalize beyond rigid string-level imitation. **(2)** R1-style reinforcement learning does not require tool-call data with explicit reasoning annotations. Instead, it imposes minimal constraints, requiring only that the model follow a structured output format. This makes it possible to train reasoning capabilities directly from available SFT data. As a result, the model can acquire reasoning skills without curating reasoning traces.

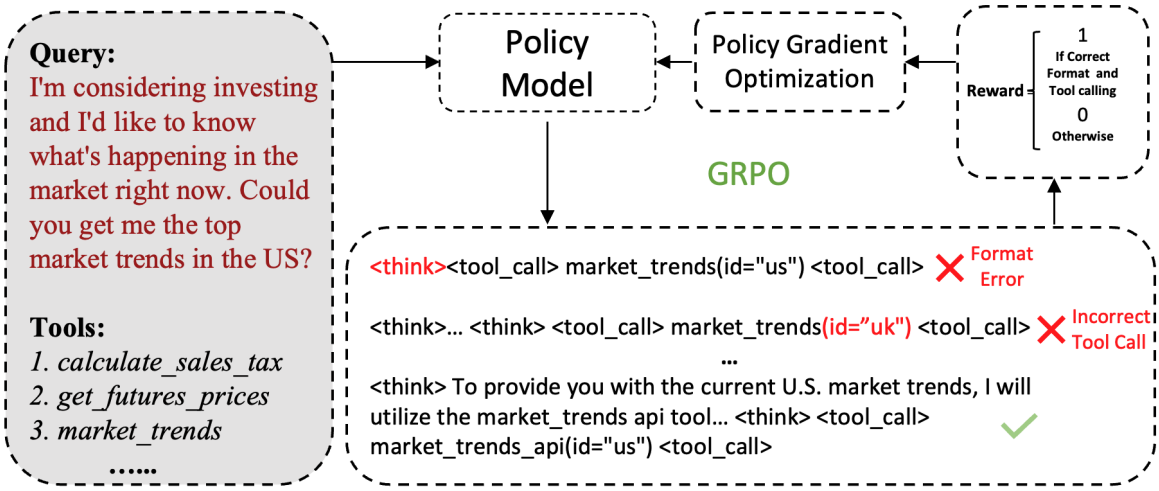

Figure 1. Overview of the training pipeline for Nemotron-Research-Tool-N1 (Tool-N1). Starting from standard SFT tool-calling data comprising user queries and candidate tools, we train LLMs to produce structured reasoning and tool calls using a binary reward function within the GRPO algorithm. As supervision is only applied to the format and tool-call correctness, the training process does not require curated reasoning trajectories.

Building on these insights, we introduce Nemotron-Research-Tool-N1 (Tool-N1)[1], a series of tool-using language models trained with a rule-based reinforcement learning. The training procedure enforces a simple-tructured reasoning-action format, guiding the model to produce explicit reasoning before invoking tools. We employ a binary reward function that evaluates both the structural correctness of the reasoning format and the accuracy of tool invocations. This reward design provides precise yet flexible supervision, allowing variation in argument ordering while ensuring functional correctness. We conduct extensive experiments on BFCL [39] and API-Bank [12]. Empirical results show that our models, Tool-N1-7B and Tool-N1-14B-built upon Qwen2.5-7B/14B-Instruct consistently outperform GPT-4o and other strong baselines across both benchmarks.

2. Related Work

Tool Learning. Enhancing large language models (LLMs) with external tools has demonstrated significant potential in addressing complex tasks [1,29,36]. Typical applications include integrating LLMs with search engines [10,11,32,49], calculator [24], vision tools [7,20], and Python interpreters [33, 35,38]. Research efforts aimed at improving LLMs' tool-use capabilities can be broadly divided into two

[1] Throughout this paper, we refer to Nemotron-Research-Tool-N1 as Tool-N1 for brevity.

 doi:10.20944/preprints202504.2471.v1

categories. The first one focuses on improving the LLM itself. This typically involves curating large-scale supervised datasets [16,21,28,43,47] and applying either SFT [16,28,47] or DPO reinforcement learning [44,45]. These approaches are primarily data-centric [46] and rely heavily on the availability of high-quality, large-scale training data. The second category explores prompting-based techniques that do not alter the underlying model. These methods aim to elicit tool-use behavior through strategies such as in-context learning with demonstrations [9,26] or the design of advanced prompting schemes [41,42,48]. However, due to the inherent limitations of the base models, such prompting approaches often struggle with tasks that require intricate reasoning or complex multi-step actions.

LLM Reasoning and Reinforcement Learning. Recent efforts in the LLM research community have increasingly focused on improving reasoning capabilities, marking a shift from train-time scaling to test-time scaling [23], with the goal of enabling models to handle complex, multi-step problem-solving tasks [37]. Earlier approaches often relied on step-level supervision or learned reward models to guide the model's reasoning trajectory [5,13]. More recently, DeepSeek-R1 [6] has demonstrated that simple rule-based reinforcement learning can effectively induce strong reasoning behaviors. In this framework, reward functions are defined based on predefined criteria, i.e., checking whether the model's final answer matches the ground truth in math problems. These rewards target only the correctness of the final answer, allowing the model to learn the intermediate reasoning steps without explicit reasoning supervision. This R1-style reasoning paradigm has shown success across various domains, including mathematics [30], coding [15], and vision tasks [31]. In parallel, several recent studies have integrated external tools such as search engines [8] and code interpreters [3] into reinforcement learning frameworks for LLM reasoning. These approaches primarily focus on training LLMs to effectively utilize a single, general-purpose tool to support one single reasoning chain. In contrast, our work aims to enhance LLMs' tool-calling ability in more general tool-use settings, where multiple tools with complex functionalities and argument structures are available.

3. Problem Formulation

We first provide the problem formulation. Consider a large language model (LLM) and a set of external tools $\mathcal{Z} = \{z_i\}_{i=i}^{I}$ that the LLM can access. Each tool z_i could be represented as a 3-tuple (n_i, d_i, k_i) which includes essential information for tool usage: n_i denotes the tool's name, d_i provides a natural language descriptions of the tool, and k_i specifies the tool's input parameters instructions. The model's objective is to respond to user queries in accordance with a policy π. To achieve this, the LLM may issue multiple tool calls throughout the interaction, each with appropriate parameters.

At any decision step t, the LLM receives two types of input: **(1)** the historical context c_t, which consists of all preceding tool-call and observation pairs, and **(2)** the set of currently available tools $\mathcal{Z}$ that can be utilized at this step. The LLM must then decide on the next action. Formally, the decision-making process is defined as:

$$\pi(c_t, \tilde{\mathcal{Z}}) \rightarrow a_t, \ \text{ s.t. } \ a_t \subseteq \mathcal{Z}, \tag{1}$$

where a_t denotes the action selected at step t. This action corresponds to one or more tool calls drawn from the accessible tool subset $\tilde{\mathcal{Z}}$. c_t represents the historical context. Specifically,

$$\begin{cases} a_t = \{z_0(p_0), \ldots, z_m(p_m)\}, \\ c_t = (a_0, o_0, \ldots, a_t, o_t) \end{cases} \tag{2}$$

where each z_m denotes the m-th tool invoked and p_m its corresponding parameters. The value of m indicates the number of tool calls made at time t. We employ o_t denotes the observation after action a_t is taken. The ultimate goal of tool learning is to enable LLMs with a generalized policy π that effectively addresses user queries by producing a coherent sequence of action-observation pairs (a_t, o_t).

 doi:10.20944/preprints202504.2471.v1

4. Nemotron-Research-Tool-N1

In this section, we introduce Nemotron-Research-Tool-N1, an R1-style reinforcement learning framework designed to train general-purpose tool-using language models. Tool-N1 is built upon the GRPO reinforcement learning algorithm [6,30], aiming to improve the model's tool-calling capabilities in complex scenarios where the LLM is tasked with solving queries using an accessible set of tools.

Formally, given the historical context c_t and the set of currently available tools $\mathcal{Z}$, the model generates a set of candidate responses $[O^1, O^2, ..., O^N]$. Each response consists of (1) textual reasoning and (2) an associated action a_n. These responses are evaluated using a reward function, yielding a reward set $\{r_1, r_2, ..., r_N\}$. The GRPO algorithm is then employed to estimate advantages and update the policy model, subject to KL divergence constraints. The relative advantage A_i of the i-th response is computed as follows:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_N\})}{\text{std}(\{r_1, r_2, \ldots, r_N\})}, \tag{3}$$

where mean and std represent the mean and standard deviation of the rewards, respectively. In the subsequent sections, we detail the following components of our approach: (1) the preparation of supervised fine-tuning data and its integration into the R1-style RL training pipeline, (2) the structured reasoning template used during RL training, and (3) the reward modeling strategy employed to guide the learning process.

4.1. Data Preparation

Numerous prior works have focused on collecting large-scale tool-calling trajectories [16,28,43,47], followed by supervised fine-tuning (SFT) to improve the tool-use capabilities of LLMs. These datasets typically consist of a natural language user query Q, paired with a sequence of ground-truth tool invocation steps in the form $(a_0, o_o, \ldots, a_t, o_t)$. The model is then trained to predict each subsequent action a_t, conditioned on the observed trajectory up to that point. However, SFT often exhibits limited generalization ability, as the model tends to memorize the training trajectories rather than developing robust, intrinsic reasoning capabilities for tool usage.

To fully leverage the available SFT data in the community, we unify and preprocess data from xLAM [47] and a subset of ToolACE [16], which provide single-turn and multi-turn synthetic tool-calling trajectories. As these datasets are generated by potentially unstable LLMs, they often contain inconsistencies and unstructured formats unsuitable for GRPO training. We standardize the dataset by filtering out samples with invalid tool calls, specifically, those involving tools absent from the candidate tool list. Available tools are extracted from the system prompt, and both candidate tools and ground-truth tool calls are parsed into structured dictionary formats. Instances that fail JSON parsing or contain formatting inconsistencies are discarded. This preprocessing yields a clean and consistent dataset suitable for reinforcement learning. For multi-turn data from the ToolACE subset, we further segment each trajectory into multiple single-step prediction instances, where each instance contains one target tool call and the preceding steps are treated as context. We train LLMs using R1-style GRPO to predict each tool invocation step based on this contextual information and provided tools. We present the statistic information of the proceed data in Table 1.

Table 1. The statistic information in the data preparation stage.

	xLAM		ToolACE	
	Single-Turn	Multi-Turn	Single-Turn	Multi-Turn
Raw Data	60000	0	10500	800
After Process	60000	0	8183	1470

4.2. Thinking Template

Following Guo et al. [6], we adopt a lightweight prompting template to elicit tool calls from the LLM which is shown in Figure 2. The prompt explicitly instructs the model to generate in-

 doi:10.20944/preprints202504.2471.v1

termediate reasoning within <think>...</think> tags, followed by the tool invocation enclosed in <tool_call>...</tool_call> tags. The design philosophy behind this template is to minimize reliance on overly rigid formatting rules, which could reducing the risk of overfitting to specific prompt patterns. By allowing greater flexibility in how the model expresses its reasoning, we aim to promote more robust generalization across diverse tool-use scenarios. Additionally, the use of this lightweight prompting design during training enables the resulting model to be more easily integrated with more sophisticated prompting strategies [41,42,48].

Thinking Template

```
Here is a list of functions in JSON format that you can invoke: <tools>{tools}
</tools>. In each action step, you MUST:
1. Think about the reasoning process in the mind and enclosed your reasoning within
<think></think> XML tags.
2. Then, provide a json object with function names and arguments within <tool_call>
</tool_call> XML tags. i.e., <tool_call>[{"name": <function-name>, "arguments": <
args-json-object>}}, {"name": <function-name2>, "arguments": <args-json-object2>}},
...]</tool_call>
3. Make sure both the reasoning and the tool call steps are included together in
one single reply.
```

Figure 2. The structured reasoning template used during training. The prompt guides the LLM to explicitly separate its internal reasoning (within `<think>` tags) from tool invocation actions (within `<tool_call>` tags).

4.3. Reward Modeling

Following the data preparation described in Section 4.1, we construct a training dataset in which each ground-truth tool call is represented as a structured dictionary. This format enables reliable verification of tool names and argument-value pairs during reinforcement learning rather than simply string match. Leveraging this structure, we define a binary reward function in the R1-style, which jointly evaluates the correctness of the reasoning format and the accuracy of the tool call, including its name and arguments.

Formate Checking. Following prior work [6,22,30], we incorporate format checking during training to verify whether the model's output adheres to the expected structural conventions—specifically, whether the reasoning is enclosed within <think>...</think> tags and the tool call is properly placed within <tool_call>...</tool_call> tags. This structural constraint encourages the model to engage in explicit reasoning before tool invocation, rather than shortcutting to the final answer. By enforcing format adherence, we aim to cultivate the model's intrinsic reasoning ability, which may potentially contributes to improved generalization—particularly on out-of-distribution inputs.

Tool-Calling Checking. We also check the correctness of the tool call itself. The tool-call output is parsed as a dictionary, enabling exact matching against the ground-truth call. This involves checking whether the predicted tool name matches the ground-truth and whether all required arguments are present with correct values. This strict matching criterion ensures that the model learns to generate functionally precise and executable tool calls. Compared to the next-token prediction logic in SFT, this dictionary-based matching introduces greater flexibility. It allows argument order to vary without penalization, encouraging the model to focus on the underlying semantics of tool invocation rather than surface-level memorization. This design promotes a deeper understanding of tool usage and supports better generalization.

Binary Reward Definition. Given a context c_t and the predicted action a_t, we define a binary reward function $r(c_t, a_t) \in \{0, 1\}$ that assigns a reward of 1 if both of the following conditions are met: (1) **Format Correctness:** The model output adheres to the structural format, i.e., contains both `<think>...</think>` and `<tool_call>...</tool_call>` tags. (2) **Tool Call Correctness:** The

 doi:10.20944/preprints202504.2471.v1

predicted tool call a_t exactly matches the ground-truth call a_t^* in both tool name and all argument key-value pairs.

$$r(c_t, a_t) = \begin{cases} 1, & \text{if } \texttt{FormatCorrect}(a_t) \wedge \texttt{ToolCallMatch}(a_t, a_t^*) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where $\texttt{FormatCorrect}(a_t)$ returns true if the output is correctly wrapped in both required tags, and the $\texttt{ToolCallMatch}(a_t, a_t^*)$ returns true if a_t matches the ground-truth tool call a_t^* exactly in structure and content.

5. Experiments

We conduct experiments to prove the superiority of the proposed method. We begin by providing the experimental settings in Section 5.1. We then evaluate the training method on two typical benchmarks to verify its effectiveness in Section 5.2. Finally, we perform in-depth investigations of the proposed method.

5.1. Settings

Datasets. We primarily utilize subsets of ToolACE [16] and xLAM [47] as our training dataset. ToolACE [16] encompasses a wide range of tool-calling scenarios, including examples with multiple candidate tools and parallel function calls, and covers a pool of 26,507 diverse tools. In contrast, xLAM focuses on single-turn function calling, comprising 60,000 instances collected through APIGen [17].

Models. Unless otherwise noted, we use Qwen2.5-7B/14B-Instruct [40] as the primary backbone model throughout this work. To assess the generalization ability of our method, we also perform evaluations on alternative backbone models, including multiple variants from the LLaMA family across different scales. In our experiments, we compare against both general-purpose open-source models, such as the GPT series and Gemini-2.0, as well as specialized tool-calling models, including ToolACE-8B [16], xLAM-2 [27], and Hammer2.1 [14].

Benchmarks. We primarily evaluate performance on single-turn tool-calling queries. We evaluate our approach on several representative benchmarks, including the Berkeley Function Call Leaderboard (BFCL) [39] and API-Bank [12]. For BFCL, we conduct evaluations on both the Non-live and Live subsets, corresponding to synthetic and real-world data, respectively. Each subset includes four categories: Simple, Multiple, Parallel, and Parallel Multiple. Simple and Multiple scenarios both involve the invocation of a single tool, with the Multiple category featuring multiple candidate tools. In contrast, Parallel and Parallel Multiple scenarios require the simultaneous invocation of multiple tools. For API-Bank, we exclude multi-turn cases from our evaluation. Performance across all benchmarks is reported in terms of accuracy.

Other Implementation Details. We conduct all experiments using the open-source reinforcement learning library Verl. Training is performed with a batch size of 1024 and a learning rate of 1×10^{-6}. The temperature is fixed at 0.7. We set the entropy coefficient to 0, as we observe that introducing entropy negatively impacts exploration during training. The KL divergence loss coefficient is set to 1×10^{-3} across all experiments. All training runs are executed on a cluster of 4 nodes, each equipped with 8 NVIDIA H100 80GB GPUs.

5.2. Main Results

Results on BFCL. Table 2 presents the evaluation results on the BFCL benchmark, including detailed accuracy scores for each subcategory. We use the official evaluation script from the BFCL leaderboard and report average accuracy across categories. The results clearly demonstrate that all the Tool-N1-7B/14B models achieving the best overall performance, outperforming both state-of-the-art closed-source models, such as GPT-4o, and specialized fine-tuned models, including xLAM-2-70B and ToolACE-8B. Notably, the trained tool-call reasoning model significantly outperforms supervised

 doi:10.20944/preprints202504.2471.v1

fine-tuning baselines trained on the same data sources (i.e., the ToolACE [16] and xLAM [47] series). The results prove that R1-style reinforcement learning offers a more effective paradigm for enhancing the tool-calling capabilities of LLMs compared to standard supervised fine-tuning.

Table 2. Comparison on the BFCL benchmark (last updated on 2025-04-13). Average performance is calculated using the official script which based on both weighted and unweighted accuracies across all subcategories. The best results in each category are highlighted in **bold**, while the second-best results are underlined.

Models	Non-Live				Live				Overall		
	Simple	*Multiple*	*Parallel*	*Parallel Multiple*	*Simple*	*Multiple*	*Parallel*	*Parallel Multiple*	*Non-live*	*Live*	*Overall*
GPT-4o-2024-11-20	79.42	95.50	94.00	83.50	**84.88**	79.77	87.50	75.00	88.10	79.83	83.97
GPT-4o-mini-2024-07-18	80.08	90.50	89.50	87.00	81.40	76.73	**93.75**	79.17	86.77	76.50	81.64
GPT-3.5-Turbo-0125	77.92	93.50	67.00	53.00	80.62	78.63	75.00	58.33	72.85	68.55	70.70
Gemini-2.0-Flash-001	74.92	89.50	86.50	87.00	75.58	73.12	81.25	**83.33**	84.48	81.39	82.94
DeepSeek-R1	76.42	94.50	90.05	88.00	84.11	79.87	87.50	70.83	87.35	74.41	80.88
Llama3.1-70B-Inst	77.92	**96.00**	**94.50**	91.50	78.29	76.16	87.50	66.67	89.98	62.24	76.11
Llama3.1-8B-Inst	72.83	93.50	87.00	83.50	74.03	73.31	56.25	54.17	84.21	61.08	72.65
Qwen2.5-7B-Inst	75.33	94.50	91.50	84.50	76.74	74.93	62.50	70.83	86.46	67.44	76.95
xLAM-2-70b-fc-r (FC)	78.25	94.50	92.00	89.00	77.13	71.13	68.75	58.33	88.44	72.95	80.70
ToolACE-8B (FC)	76.67	93.50	90.50	89.50	73.26	76.73	81.25	70.83	87.54	78.59	82.57
Hammer2.1-7B (FC)	78.08	95.00	93.50	88.00	76.74	77.4	81.25	70.83	88.65	75.11	81.88
Nemotron-Research- Tool-N1-7B	77.00	95.00	**94.50**	90.50	82.17	80.44	62.50	70.83	89.25	80.38	84.82
Nemotron-Research- Tool-N1-14B	**80.58**	**96.00**	93.50	**92.00**	84.10	**81.10**	81.25	66.67	**90.52**	**81.42**	**85.97**

Results on API-Bank. Additionally, to provide a more comprehensive evaluation, we conduct experiments on the API-Bank benchmark [12], with results shown in Table 3. The Tool-N1-7B and Tool-N1-14B models consistently outperform the baselines across most cases. Notably, Tool-N1-7B/14B achieve 4.12% and 5.03% higher accuracy than GPT-4o, respectively, clearly demonstrating the effectiveness of the method.

5.3. Deep Analysis

5.3.1. Scalability and Generalizability

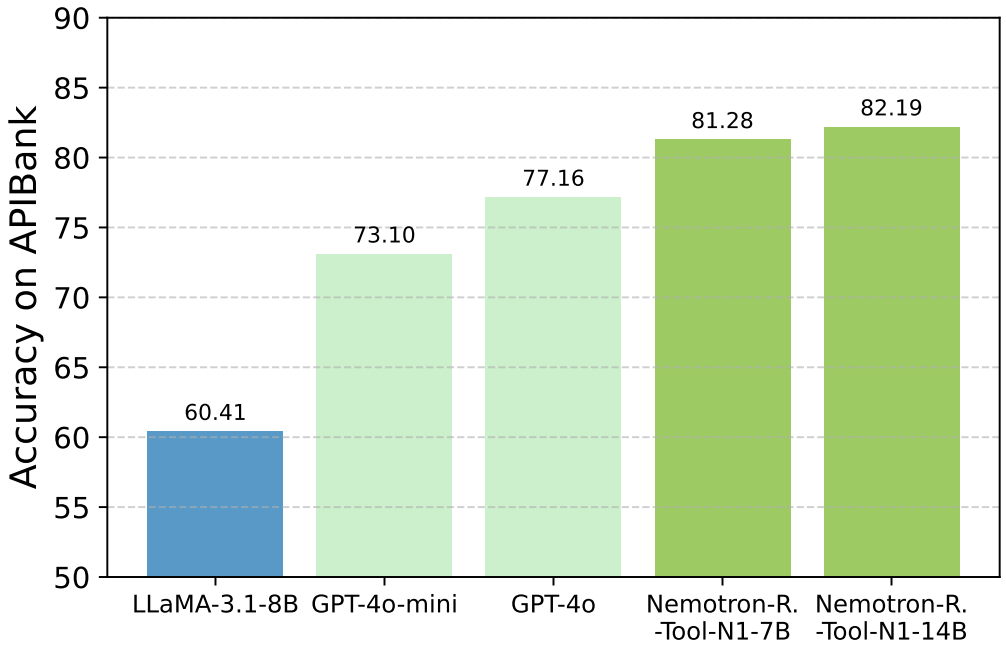

Figure 3. Comparison of model accuracy on the API-Bank. Tool-N1 models achieve the highest accuracy, demonstrating the effectiveness of the proposed training method.

Scalability. The scaling law, which characterizes the relationship between model size and performance, plays a critical role in understanding the effectiveness of the training methods. We assess the scaling behavior of the proposed training method by evaluating a range of model sizes, including

 doi:10.20944/preprints202504.2471.v1

0.5B, 1.5B, 3B, 7B and 14B from the Qwen2.5-Instruct series. For comparison, we also report the performance of the original instruction-tuned models without any additional training. We report average performance on both the Live and Non-Live categories of the BFCL benchmark, with detailed results presented in Figure 4. As expected, larger models consistently outperform smaller ones in both evaluation settings. Notably, performance improvements from post-training are limited for smaller models (0.5B and 1.5B), whereas larger models exhibit substantial gains. These findings suggest that the R1-style training method scales more effectively with increasing model size.

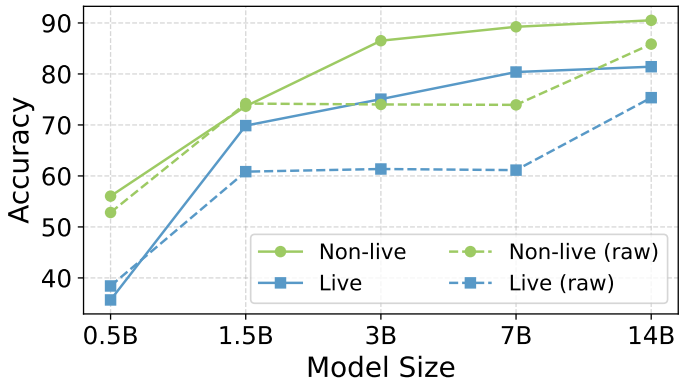

Figure 4. Scaling performance across model sizes. We use the Qwen2.5-Instruct series as backbones and report the performance of both the RL-trained the original instruction-tuned models without post-training.

Generalizability. We further evaluate the impact of different backbone LLMs [34] to investigate the generalization capability of the proposed training method. In addition to the Qwen series, we include experiments using LLaMA-based models: LLaMA3-8B-Instruct and LLaMA3.1-8B-Instruct. These evaluations are conducted on the BFCL benchmark, with results presented in Figure 5. Our findings show that Qwen2.5-Instruct significantly outperforms both LLaMA variants at the same model scale. This advantage is likely due to Qwen's inherently stronger reasoning capabilities, as previously observed by Gandhi et al. [4]. As a result, the R1-style training paradigm is able to elicit better performance when applied to Qwen.

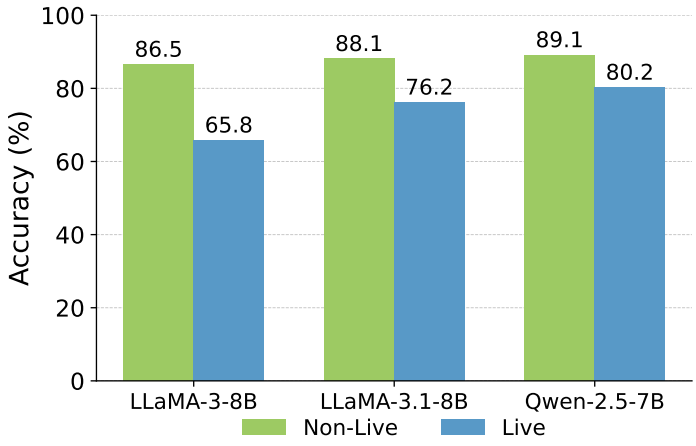

Figure 5. Performance across different backbones. For all involved models, we adopt their instruct-tuned variants. At comparable model scales, Qwen consistently outperforms both LLaMA 3 and LLaMA 3.1.

5.3.2. Ablations

Ablations on Reward Design. To assess how reward granularity affects model behavior, we evaluate Tool-N1-7B under two reward schemes: fine-grained and binary (Table 3). The fine-grained setting provides partial reward, 0.2 for correct reasoning format and an additional 0.2 for matching function names, even if the final function call is incorrect. In contrast, the binary setting only gives a

 doi:10.20944/preprints202504.2471.v1

reward of 1.0 when all components are correct, including reasoning, function name, and arguments. Tool-N1 achieves consistently better performance with binary rewards, particularly on the Live subset (80.38% vs. 76.61%), which involves more realistic inputs. We attribute this to reduced reward hacking [25]: under fine-grained schemes, the model may overfit to superficial cues such as formatting or partial matches, without ensuring full execution correctness. Furthermore, within the binary setup, we observe that removing the reasoning format constraint significantly hurts performance (dropping from 80.38% to 76.24%). This highlights the critical role of structured reasoning in guiding Tool-N1-7B toward reliable and generalizable tool use, especially in complex, real-world scenarios.

Table 3. Ablation study on reward granularity. We compare fine-grained reward designs, where partial credit is given for correct reasoning format and correct function names in function call stages, with binary rewards that assign full reward only when all conditions are fully satisfied. The results show that binary rewards consistently yield better performance, especially in the Live setting.

	Fine-Grained Reward Design		Binary Reward Design	
Split	w/ Reason Format Partial Reward	w/ Reason Format + Func name Partial Rewards	w/o Reason Format	w/ Reason Format
Non-Live	87.83	88.54	87.63	89.25
Live	79.64	76.61	76.24	80.38
Avg	83.74	82.58	81.94	84.82

Ablations on Training Data Composition. We then investigate how different data composition strategies affect performance on the BFCL benchmark. Experiments are conducted using the Tool-N1-7B model, with results presented in Table 4. Our key findings are as follows: (1) Compared to the raw model (Qwen2.5-7B-Instruct), R1-style training significantly enhances tool-calling capabilities. (2) ToolACE data yields particularly strong improvements in the live setting. Overall, these results suggest that progressively enriching the training data enhances the model's tool-calling proficiency. (3) Compared to models trained using SFT on the same data, the R1-style training consistently yields better performance. Specifically, the Tool-N1-7B model trained solely on xLAM data outperforms the xLAM-8B SFT model by 6.36%, and the Tool-N1-7B model trained solely on the ToolACE subset exceeds the ToolACE-8B SFT model by 1.62%, despite using only a subset of the data.

Table 4. Effect of training data composition on Tool-N1-7B performance. We compare the full training recipe against ablated variants that exclude xLAM or ToolACE data. Results show that both sources contribute to performance, with ToolACE providing greater gains.

Recipe	Raw Model	xLAM 8B	ToolACE 8B	xLAM w/o ToolACE	ToolACE w/o xLAM	ToolACE w/ xLAM
Non-Live	73.94	84.40	87.54	87.77	87.67	89.25
Live	61.14	66.90	78.59	76.24	79.58	80.38
Avg	67.54	75.65	82.57	82.01	83.63	84.82

6. Conclusion

We introduces Nemotron-Research-Tool-N1, a series of tool-using language models trained with a rule-based reinforcement learning. Unlike prior approaches that depend on supervised fine-tuning, Nemotron-Research-Tool-N1 leverages a reward function that supervises only the final answer and the structural format of reasoning. This allows the model to learn effective reasoning strategies without requiring annotated reasoning trajectories. Experimental results show that Nemotron-Research-Tool-N1 consistently outperforms existing baselines across multiple benchmarks, including BFCL and API-Bank. Furthermore, when trained on the same data, models using R1-style reinforcement learning

 doi:10.20944/preprints202504.2471.v1

achieve superior performance compared to their SFT-trained counterparts, affirming the benefits of reinforcement-based training over SFT.

References

1. Andres M Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Chemcrow: Augmenting large-language models with chemistry tools. *arXiv preprint arXiv:2304.05376*, 2023.
2. Hardy Chen, Haoqin Tu, Fali Wang, Hui Liu, Xianfeng Tang, Xinya Du, Yuyin Zhou, and Cihang Xie. Sft or rl? an early investigation into training r1-like reasoning large vision-language models. *arXiv preprint arXiv:2504.11468*, 2025.
3. Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.
4. Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*, 2025.
5. Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weilin Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. On designing effective rl reward at training time for llm reasoning. *arXiv preprint arXiv:2410.15115*, 2024.
6. Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
7. Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *arXiv preprint arXiv:2406.09403*, 2024.
8. Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.
9. Geunwoo Kim, Pierre Baldi, and Stephen McAleer. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36:39648–39677, 2023.
10. Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. *arXiv preprint arXiv:2107.07566*, 2021.
11. Angeliki Lazaridou, Elena Gribovskaya, Wojciech Stokowiec, and Nikolai Grigorev. Internet-augmented language models through few-shot prompting for open-domain question answering. *arXiv preprint arXiv:2203.05115*, 2022.
12. Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. Api-bank: A comprehensive benchmark for tool-augmented llms. *arXiv preprint arXiv:2304.08244*, 2023.
13. Wendi Li and Yixuan Li. Process reward model with q-value rankings. *arXiv preprint arXiv:2410.11287*, 2024.
14. Qiqiang Lin, Muning Wen, Qiuying Peng, Guanyu Nie, Junwei Liao, Jun Wang, Xiaoyun Mo, Jiamu Zhou, Cheng Cheng, Yin Zhao, et al. Hammer: Robust function-calling for on-device language models via function masking. *arXiv preprint arXiv:2410.04587*, 2024.
15. Jiawei Liu and Lingming Zhang. Code-r1: Reproducing r1 for code with reliable rewards. *arXiv preprint arXiv:2503.18470*, 2025.
16. 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.
17. Zuxin Liu, Thai Hoang, Jianguo Zhang, Ming Zhu, Tian Lan, Juntao Tan, Weiran Yao, Zhiwei Liu, Yihao Feng, Rithesh RN, et al. Apigen: Automated pipeline for generating verifiable and diverse function-calling datasets. *Advances in Neural Information Processing Systems*, 37:54463–54482, 2024.
18. Zhengxi Lu, Yuxiang Chai, Yaxuan Guo, Xi Yin, Liang Liu, Hao Wang, Guanjing Xiong, and Hongsheng Li. Ui-r1: Enhancing action prediction of gui agents by reinforcement learning. *arXiv preprint arXiv:2503.21620*, 2025.
19. Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*, 2025.
20. Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. m & m's: A benchmark to evaluate tool-use for m ulti-step m ulti-modal tasks. In *European Conference on Computer Vision*, pages 18–34. Springer, 2024.

 doi:10.20944/preprints202504.2471.v1

21. Zixian Ma, Jianguo Zhang, Zhiwei Liu, Jieyu Zhang, Juntao Tan, Manli Shu, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Caiming Xiong, et al. Taco: Learning multi-modal action models with synthetic chains-of-thought-and-action. *arXiv preprint arXiv:2412.05479*, 2024.
22. Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Daocheng Fu, Botian Shi, Wenhai Wang, Junjun He, Kaipeng Zhang, et al. Mm-eureka: Exploring visual aha moment with rule-based large-scale reinforcement learning. *arXiv preprint arXiv:2503.07365*, 2025.
23. Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.
24. Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021.
25. Alexander Pan, Erik Jones, Meena Jagadeesan, and Jacob Steinhardt. Feedback loops with language models drive in-context reward hacking. *arXiv preprint arXiv:2402.06627*, 2024.
26. Bhargavi Paranjape, Scott Lundberg, Sameer Singh, Hannaneh Hajishirzi, Luke Zettlemoyer, and Marco Tulio Ribeiro. Art: Automatic multi-step reasoning and tool-use for large language models. *arXiv preprint arXiv:2303.09014*, 2023.
27. Akshara Prabhakar, Zuxin Liu, Weiran Yao, Jianguo Zhang, Ming Zhu, Shiyu Wang, Zhiwei Liu, Tulika Awalgaonkar, Haolin Chen, Thai Hoang, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *arXiv preprint arXiv:2504.03601*, 2025.
28. Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023.
29. Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-Rong Wen. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8):198343, 2025.
30. Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
31. Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, et al. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*, 2025.
32. Kurt Shuster, Jing Xu, Mojtaba Komeili, Da Ju, Eric Michael Smith, Stephen Roller, Megan Ung, Moya Chen, Kushal Arora, Joshua Lane, et al. Blenderbot 3: a deployed conversational agent that continually learns to responsibly engage. *arXiv preprint arXiv:2208.03188*, 2022.
33. Linxin Song, Jiale Liu, Jieyu Zhang, Shaokun Zhang, Ao Luo, Shijian Wang, Qingyun Wu, and Chi Wang. Adaptive in-conversation team building for language model agents. *arXiv preprint arXiv:2405.19425*, 2024.
34. Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.
35. Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*, 2024.
36. Zhiruo Wang, Zhoujun Cheng, Hao Zhu, Daniel Fried, and Graham Neubig. What are tools anyway? a survey from the language model perspective. *arXiv preprint arXiv:2403.15452*, 2024.
37. Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.
38. Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. Mathchat: Converse to tackle challenging math problems with llm agents. *arXiv preprint arXiv:2306.01337*, 2023.
39. Fanjia Yan, Huanzhi Mao, Charlie Cheng-Jie Ji, Tianjun Zhang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. Berkeley function calling leaderboard. https://gorilla.cs.berkeley.edu/blogs/8_berkeley_function_calling_leaderboard.html, 2024.
40. An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

 doi:10.20944/preprints202504.2471.v1

41. Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.
42. Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.
43. Fan Yin, Zifeng Wang, I Hsu, Jun Yan, Ke Jiang, Yanfei Chen, Jindong Gu, Long T Le, Kai-Wei Chang, Chen-Yu Lee, et al. Magnet: Multi-turn tool-use data synthesis and distillation via graph translation. *arXiv preprint arXiv:2503.07826*, 2025.
44. Yuanqing Yu, Zhefan Wang, Weizhi Ma, Zhicheng Guo, Jingtao Zhan, Shuai Wang, Chuhan Wu, Zhiqiang Guo, and Min Zhang. Steptool: A step-grained reinforcement learning framework for tool learning in llms. *arXiv preprint arXiv:2410.07745*, 2024.
45. Yirong Zeng, Xiao Ding, Yuxian Wang, Weiwen Liu, Wu Ning, Yutai Hou, Xu Huang, Bing Qin, and Ting Liu. Boosting tool use of large language models via iterative reinforced fine-tuning. *arXiv preprint arXiv:2501.09766*, 2025.
46. Daochen Zha, Zaid Pervaiz Bhat, Kwei-Herng Lai, Fan Yang, Zhimeng Jiang, Shaochen Zhong, and Xia Hu. Data-centric artificial intelligence: A survey. *ACM Computing Surveys*, 57(5):1–42, 2025.
47. Jianguo Zhang, Tian Lan, Ming Zhu, Zuxin Liu, Thai Hoang, Shirley Kokane, Weiran Yao, Juntao Tan, Akshara Prabhakar, Haolin Chen, et al. xlam: A family of large action models to empower ai agent systems. *arXiv preprint arXiv:2409.03215*, 2024.
48. Shaokun Zhang, Jieyu Zhang, Dujian Ding, Mirian Hipolito Garcia, Ankur Mallick, Daniel Madrigal, Menglin Xia, Victor Rühle, Qingyun Wu, and Chi Wang. Ecoact: Economic agent determines when to register what action. *arXiv preprint arXiv:2411.01643*, 2024.
49. Shaokun Zhang, Jieyu Zhang, Jiale Liu, Linxin Song, Chi Wang, Ranjay Krishna, and Qingyun Wu. Training language model agents without modifying language models. *arXiv e-prints*, pages arXiv–2402, 2024.