NVIDIA

在RL背景下，如何将LLM + TOOL-USING 扩展到非数学/编程领域？

Nemotron-Research-Tool-N1: Exploring 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] Guilin Liu[1]

[1]NVIDIA [2]Pennsylvania State University [3]University of Washington

简介

代码：HTTPS://GITHUB.COM/NVLABS/TOOL-N1

本文提出了Tool-N1，探索在非数学/编程领域，延续RLVR做法，让llm学会使用tool(本文更像function calling)提升自己的能力，为了作者设计了一个简单的二值reward function：只有response的format和tool 调用参数正确就是1，否则是0

背景

我个人认为本文和ToolRL很像，因此直接复制：
目前LLM + Tool via RL的工作基本聚焦在数学领域，一方面是数学题很容易验证正确性，适合RLVR，另一方面相关的公开数据集非常丰富。本文则思考如何将LLM + Tool via RL扩展到通用领域，延续RLVR的做法，那么重点就是如何设计reward function?

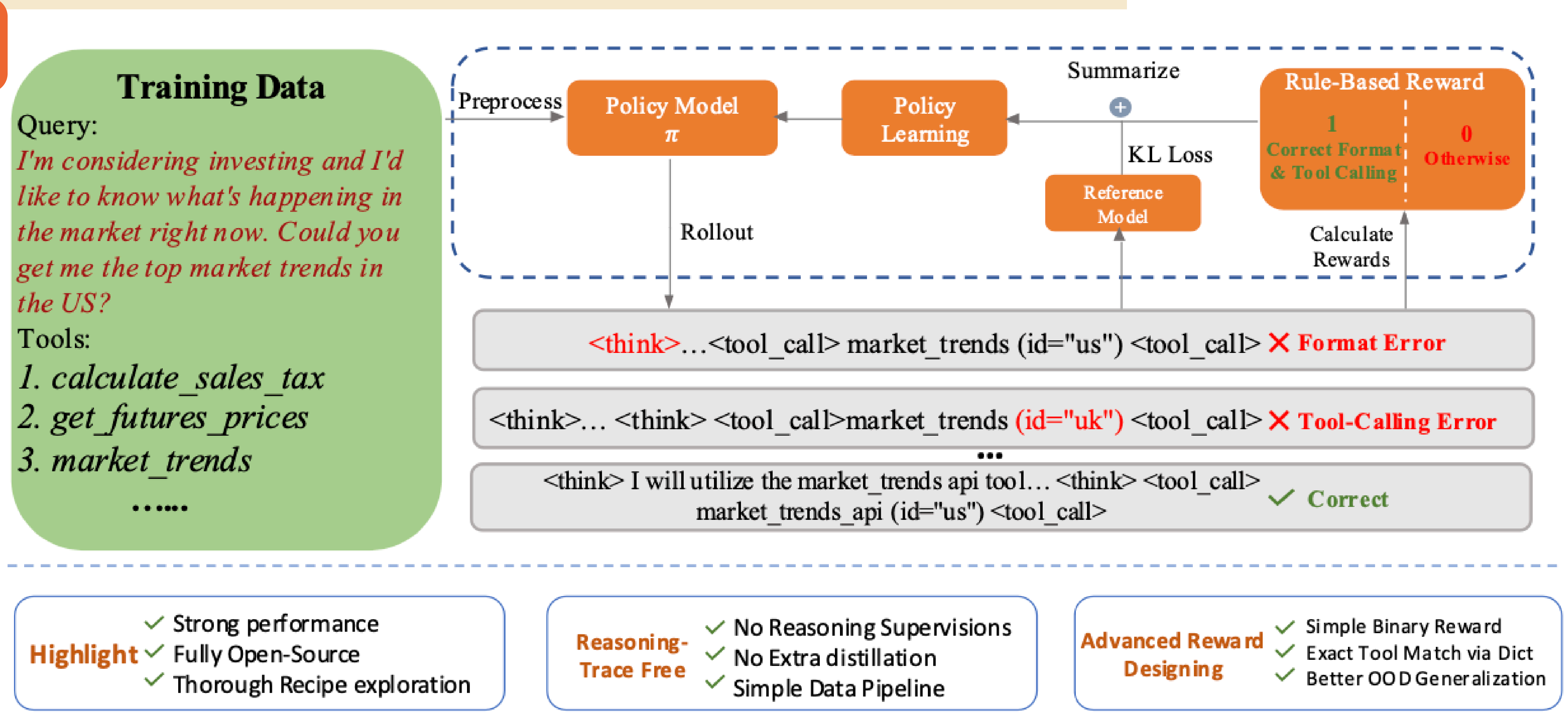
实验设置

- 框架：verl
- 实验对象：Qwen2.5-7B/14B-Instruct
- 强化学习算法：GRPO
- reward function：kl loss + 一个二值(0, 1) reward，二值指的是只有response的format和tool 调用正确就是1，否则是0

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

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{(c_t,\mathcal{Z})}\,\mathbb{E}_{O^i \sim \mathcal{O}}\Big[\min\big(\rho_i A_i,\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)A_i\big) - \beta\,\text{KL}(\pi_\theta \,\|\, \pi_{\text{old}})\Big], \text{where } \rho_i = \frac{\pi_\theta(O^i \mid c_t, \mathcal{Z})}{\pi_{\text{old}}(O^i \mid c_t, \mathcal{Z})}.$$

REWARD



部分实验结果

Models	Non-Live Simple	Non-Live Multiple	Non-Live Parallel	Non-Live Parallel Multiple	Live Simple	Live Multiple	Live Parallel	Live Parallel Multiple	Overall Non-live	Overall Live	Overall Overall
GPT-4o	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	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
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
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**

Table 2: Comparison on the BFCL (last updated on 2025-04-13). Average performance is calculated using the official script. The best results in each category are highlighted in **bold**, while the second-best are underlined.

@机器爱学习

思考

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.
A complete reply example is: <think>To address the query, I need to send the email to Bob and then buy the banana through walmart.</think> <tool_call> ["name":"email", "arguments":"receiver": "Bob", "content": "I will bug banana through walmart", "name": "walmart", "arguments": "input": "banana"]</tool_call>. Please make sure the type of the arguments is correct.

感觉本文和TOOLRL很像，因此同样的思考内容就不写了。单说一点，上面是论文给出的THINKING TEMPLATE，不清楚是不是我对“EACH ACTION STEP”理解的不对，看起来是指定让LLM的RESPONSE包含两部分，THINK和TOOL调用参数，为什么LLM一定要有这种固定的输出格式呢？难道不可以 <THINK> ... </THINK> <TOOL_CALL> ... </TOOL_CALL> <THINK> ... </THINK>？换句话说，LLM应该自己决定在何时插入TOOL调用，调用TOOL几次。我觉得这样的使用场景才更合理。当然了，看一下本文训练集(TOOLACE, XLAM)似乎就明白了，因为它们就是这样的格式。