

# Towards Enhanced Exploration Reasoning from Stream-of-Search & RLOO

GITA401 – Introduction to Reinforcement Learning

# Project Objectives

## Overview

### Countdown numbers game

- [https://en.wikipedia.org/wiki/Countdown\\_\(game\\_show\)](https://en.wikipedia.org/wiki/Countdown_(game_show))
- 영국에서 방영한 Countdown Game Show 의 Numbers Round
- 제시된 숫자 집합을 산술 연산과 결합하여 목표 숫자에 도달하는 게임
- Katz et al.[1] 에 따르면 해당 과정은 NP-Complete Problem
- 이 문제를 풀기 위해서는 계획, 탐색, 백트래킹 과정을 종합적으로 고려해야 함

#### Example:

Contestant One requests two large numbers and four small numbers.

Selection is:

75 50 2 3 8 7

Randomly generated target is:

812

Contestant One declares 813, while Contestant Two declares 815.

Contestant One is closer and so reveals:  $75 + 50 - 8 = 117$ , and  $117 \times 7 - (3 \times 2) = 813$ , which scores 7 points for being 1 away. Contestant Two does not score.

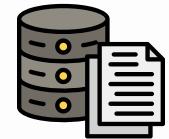
Assistant notes:  $50 + 8 = 58$ , and  $7 \times 2 \times 58 = 812$ , which would have scored 10 points.

## 구현 전략

### 주요 가설

1. Stream of Search 형태의 추론과정을 서술하는 데이터셋을 활용한 SFT.로 모델에게 생각하는 방법을 가르킨다!
2. RLHF 를 이용하여 수학 능력을 강화시켜 정확성을 상승 시킬수 있다.

# Dataset



## DataSet Description

### Countdown numbers 4 Dataset

#### [SFT Training Set]

- Stream of Search[2] Github[3] 의 코드를 이용하여 10만개의 데이터를 생성하였습니다.
- 랜덤으로 생성한 4개의 숫자와 연산자를 이용하여 문제와 Optimal Path 를 생성합니다.
- DFS, BFS 탐색을 이용하여 최대 4K token 길이의 Search Path 를 생성합니다.

#### [RLHF Training Set]

- HuggingFace, "Jiayi-Pan/Countdown-Tasks-3to4"[4] Dataset의 4개 숫자 데이터를 이용합니다.
- nums: [76, 82, 70, 51], target: 86 Format 의 데이터

#### [Evaluation Set]

- "Jiayi-Pan/Countdown-Tasks-3to4" 의 일부 데이터를 이용합니다.

[2] Gandhi, K., Lee, D., Grand, G., Liu, M., Cheng, W., Sharma, A., & Goodman, N. D. (2024). Stream of search (sos): Learning to search in language. arXiv preprint arXiv:2404.03683.

[3] <https://github.com/kanishkg/stream-of-search>

[4] <https://huggingface.co/datasets/Jiayi-Pan/Countdown-Tasks-3to4>

# Preprocessing



## Preprocessing

### SFT Traning Data Preprocessing

- Optimal 경로를 이용하여 올바른 수식을 만듭니다.
- ((63-17)-(27-5)) 형태로 변환하여 최종 정답으로 사용

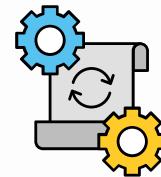
```
def _convert_solution_to_expression(self, solution_list):
    expressions = {}
    last_result = ""
    for step in solution_list:
        match = re.match(r"(\d+)([ +\-\/*\//])(\d+)=(\d+)", step)
        if match:
            left, op, right, res = match.groups()
            left_expr = expressions.get(left, left)
            right_expr = expressions.get(right, right)
            current_expr = f"({left_expr}{op}{right_expr})"
            expressions[res] = current_expr
            last_result = res
    return expressions.get(last_result, "")
```

### Chain-of-Thought Instruction Data

- user, assistant 형태 Instruction 쌍을 생성
- <think></think> Tag 를 사용하여 추론 과정을 강화합니다.

```
def _formatting_func(self, example):
    final_equation = self._convert_solution_to_expression(example['solution'])
    text = (
        f"<|im_start|>system\n당신은 숫자를 조합하여 목표값을 만드는 수학 전문가입니다.<| im_end|>\n"
        f"<|im_start|>user\nnums: {example['nums']}, target: {example['target']}<| im_end|>\n"
        f"<|im_start|>assistant\n<think>\n{example['search_path']}\n</think>\n\n"
        f"{final_equation}<|im_end|>"
    )
    return {"text": text}
```

# Preprocessing



## Preprocessing – Stream of Search

### Stream of search

- 여러 상징적 검색 전략(예: Breadth First Search, Depth First Search)을 기반으로 100,000개의 검색 경로를 생성하여 훈련 데이터셋을 구축합니다
- 이로써 모델은 최적 경로만을 학습하는 대신에 다양한 경로에서의 탐색과 백트래킹을 경험하게 됩니다.

```
"nums": [ 64, 50, 12, 56 ],  
"target": 70,  
"search_path": "Current State: 70:[64, 50, 12, 56], Operations: []\nExploring Operation: 50+12=62, Resulting Numbers: [64, 56, 62]\nGenerated Node #0,0: 70:[64, 56, 62] Operation: 50+12=62\nMoving to Node #0,0\nCurrent State: 70:[64, 56, 62], Operations: ['50+12=62']\nExploring Operation: 56+62=118, Resulting Numbers: [64, 118]\nGenerated Node #0,0,0: 70:[64, 118] Operation: 56+62=118....생략\nExploring Operation: 62+8=70, Resulting Numbers: [70]\n70,70 equal: Goal Reached\n",
```

# Reinforcement algorithm



## REINFORCE Leave-One-Out (RLOO)

### PPO 의 문제점

- 계산 비용: PPO는 일반적으로 생성자, 참조 모델(KL 추정을 위한 모델), 비평 모델, 보상 모델 등 최대 4개의 모델을 동시에 학습해야 하며, 생성자와 비평 모델의 학습은 번갈아 가며 진행됩니다(Schulman et al., 2017)
- Optimization Challenge: 온라인 RL 최적화는 불안정하고 민감한 특성을 가지며, PPO 알고리즘은 상대적으로 복잡하기 때문에 이를 효과적으로 조정하려면 전문적인 지식이 필요합니다 (Engstrom et al., 2020). [5]

### Why RLOO?

- 가장 기본적인 정책 경사 알고리즘인 바닐라 정책 경사 REINFORCE(Sutton and Barto, 2020)는 모든 데이터셋과 기본 모델 조합에서 승률 기준으로 PPO보다 3.2%에서 20.3% 까지 꾸준히 더 높은 성능을 보였습니다.
- modeling partial sequences 은 LLM 선호도 학습에 불필요 한 작업임을 효과적으로 입증하였으며 대신 전체 세대를 모델링 하면 RL 단계의 복잡성을 줄이면서 성능을 유지할 수 있고, 학습 속도도 크게 향상됩니다.
- RLOO는 노이즈와 KL 페널티에 대한 민감도가 상대적으로 낮습니다

# State, action, reward Design

## State

RLHF 의 일반적인 State

- $x=(x_1, x_2, \dots, x_n)$  는 토크나이즈된 prompt sequence

State at step  $t = s_t = x_t$

→ State는 고정된 컨텍스트,  
즉 Prompt 자체.

## Action

정책 모델이 prompt  $x$ 를 보고 생성하는 토큰 시퀀스 전체가 Action이다.

정책확률은 다음과 같다.

$$\pi_\theta(y \mid x) = \prod_{t=1}^m \pi_\theta(y_t \mid x, y_{<t})$$

토큰 레벨이 아니라 sample 단위 action 을 사용한다는 것이 RLOO의 중요한 특징.

## Reward

RLOO는 reward model 또는 rule-based function을 이용해 답변 전체에 대한 scalar reward를 부여한다.

여기서는 rule-base function 을 이용하여 리워드를 생성함.

RLOO 는 한 프롬프트에 K 개의 리워드를 생성해 평균을 내 사용.

$$b_i = \frac{1}{K-1} \sum_{\substack{j=1 \\ j \neq i}}^K r_j$$

# Reward design



## Rule-based Reward

- 수학 문제 등의 정답이 정해져 있는 문제에서는 리워드 모델을 굳이 사용할 필요 없이 Rule-Based Function 이 잘 동작함
- 포매팅 Reward, 계산과정 Reward, 최종 정답여부 Reward 를 주는 형식으로 함수를 디자인.

### 1. Fomattting Reward

프롬프트에서 Target 파싱이 정확한지?

```
re.search(r"nums: \s*\[(\d,\s]+)\],\s*target:\s*(\d+)", prompt)
```

Completions 에서 <Think>...</think>수식 포맷이 정확한지?

```
pattern = r"^(?P<think>(\s\S)*?)</think>\s*(\s\S)*?"  
completion_content = completion[0]["content"]  
  
has_think = re.match(pattern, completion_content)
```

정확하면 +10, 틀리면 -5

### 2. Accuracy Reward

계산 결과가 Target 과 일치하는지?

```
try:  
    final_val = eval(clean)  
    if abs(final_val - target_val) < 1e-5:  
        reward += 20.0 # 정답 맞히면 큰 점수  
    else:  
        reward -= 0.5 # 틀리면 감점  
except:  
    # 수식 파싱 에러 (괄호 짹 안맞음 등)  
    reward -= 3.0
```

# Experiments Setup



## Hardware 요구사항

CPU: Intel 12th Gen i5 12 core  
RAM: 32GB  
GPU: NVIDIA RTX 5090 32GB  
cuda driver: 12.8

## Software 요구사항

OS: Ubuntu 24.04  
cuda-toolkit: 12.8  
python 3.12  
torch: 2.8.0  
flash-attention2: 2.8.3

## Base Model

Qwen3-0.6B-Base[6] 모델을 베이스 모델로 사용한다.

## Inference Engine

vllm 을 이용하여 Test 진행



## Metric

### Pass@1 Accuracy

- 최종 답변의 계산결과가 실제로 Target 과 일치하는 비율

### Format Compliance

- “<Think>추론과정</Think>/₩수식“ 형식과 생성된 형식이 일치하는 비율

### Process Validity

- 탐색 중 중간 계산 과정이 수학적으로 오류가 발생하는 비율

# Experiments Setup



## Hyperparameter Setting

### RLOO 하이퍼파라미터

```
num_train_epochs=1  
learning_rate=1e-6,  
num_generations=4, # RLOO 리워드 계산을 위한 Sample K 개수
```

KL Coefficient: 0.05 # Default

```
per_device_train_batch_size=1,  
gradient_accumulation_steps=8, # 스텝 당 8개 배치로 학습  
gradient_checkpointing=True,
```

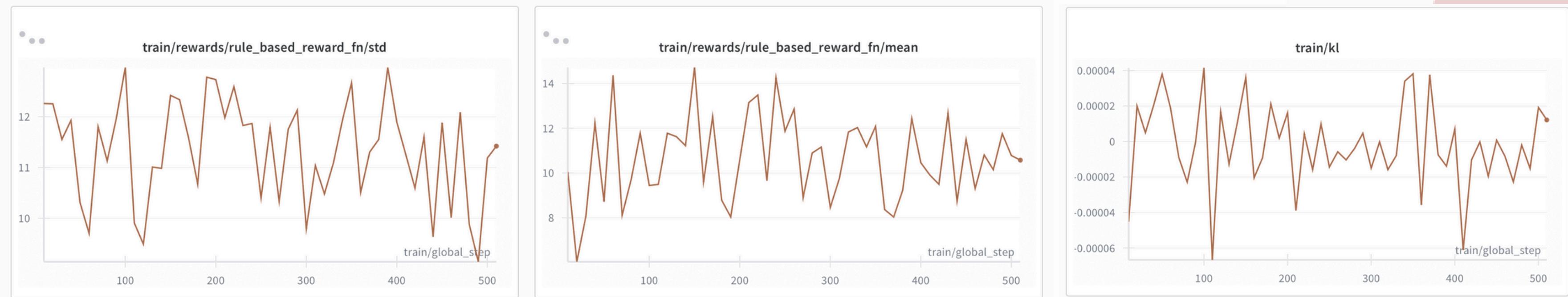
```
max_prompt_length=512, # 입력(질문) 길이 제한  
max_completion_length=4096, # 생성(답변) 길이 제한
```

```
use_vllm=True,  
vllm_mode="colocate", # vllm 추론과 학습을 하나의 GPU에서 수행
```

# Experiments Result



## 훈련과정



- 리워드의 Mean이 조금씩 올라가는 것으로 볼 수 있지만 std 표준편차는 크게 좁혀지지 않음.
- 샘플간의 리워드 차이가 매우 큰걸로 보임
- KL은 크게 변화 없이 기존 SFT 모델에서 크게 변화하지는 않고 안정적임

# Experiments Result



## Model Evaluation

### Metric 평가

550 개의 Countdown 4 문제를 이용해서 평가를 진행

#### 포맷 에러 개수

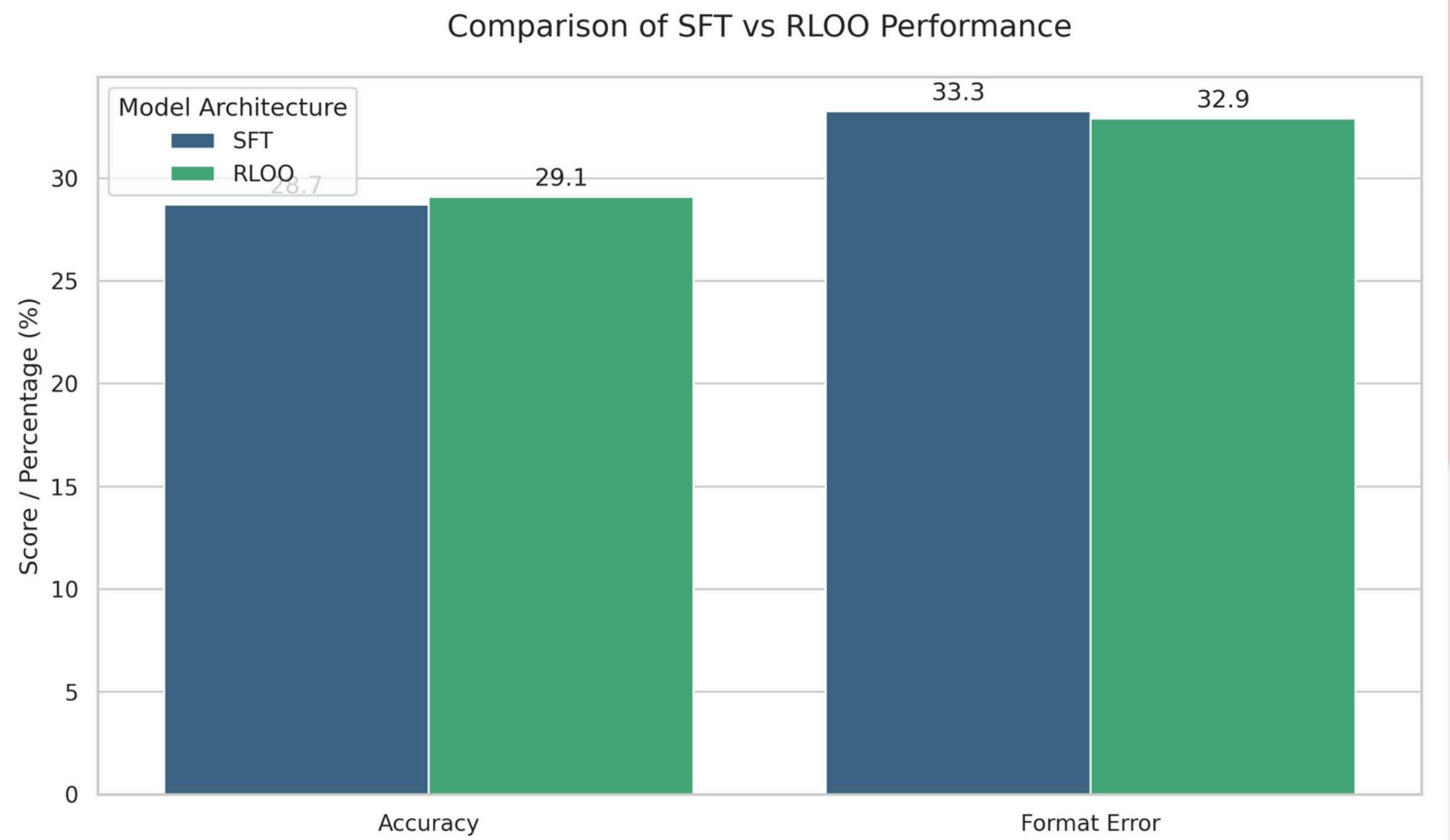
- SFT: 183
- RLOO: 181

으로 2건이 감소하였다

#### 정답을 맞춘 개수

- SFT: 158
- RLOO: 160

으로 역시 2건이 증가하다.



# Experiments Result



## Generate Example: Base vs SFT vs RLOO

Base 와는 크게 차이나지만 SFT-RLOO 차이가 크게 나지는 않는 것을 볼 수 있다.  
대부분의 경우는 동일한 답변을 생성

### Base

$$45 + 53 + 40 + 27 = 165 \quad \text{⚙️}$$

- ◆ 数 조합을 최소화하여 목표값을 만드는 수학 전문가입니다. ⚙️
- ◆ 数 조합을 최소화하여 목표값을 만드는 수학 전문가입니다. ⚙️
- ◆ 数 조합을 최소화하여 목표값을 만드는 수학 전문가입니다. ⚙️

### SFT

```
Current State: 75:[45, 53, 67], Operations:  
['20+47=67']  
Exploring Operation: 53-45=8, Resulting  
Numbers: [67, 8]  
Generated Node #0,1,2: 75:[67, 8] Operation:  
53-45=8  
Moving to Node #0,1,2  
Current State: 75:[67, 8], Operations:  
['20+47=67', '53-45=8']  
Exploring Operation: 67+8=75, Resulting  
Numbers: [75]  
75,75 equal: Goal Reached  
</think>
```

$$((47+53)-(45-20))$$

### RLOO

```
<Think>  
Current State: 75:[45, 53, 67], Operations:  
['20+47=67']  
Exploring Operation: 53-45=8, Resulting  
Numbers: [67, 8]  
Generated Node #0,1,2: 75:[67, 8] Operation:  
53-45=8  
Moving to Node #0,1,2  
Current State: 75:[67, 8], Operations:  
['20+47=67', '53-45=8']  
Exploring Operation: 67+8=75, Resulting  
Numbers: [75]  
75,75 equal: Goal Reached  
</think>
```

$$((47+53)-(45-20))$$

# 토의 및 결과



## Discussion

01

### RLOO 의 장점

PPO, GRPO 등의 actor-critic 기반 알고리즘에 비해 저렴한 비용과 훈련 세팅이 간편하고, 리워드 모델 대신 간단한 함수로만으로도 훈련이 된다는 점

해당 장점은 수학, 과학 등의 태스크에서 유리함

02

### 실험의 한계점

초기 SFT 부터 정확하지 않은 지시문으로 학습이 진행되어서 실제 Think 과정과 정답이 다른 경우가 발생

Base 모델인 Qwen3-0.6B 모델의 한계

시간과 자원의 한계로 RLHF 를 더 오랜 시간을 들여 훈련하였으면 효과가 있었을 것 같고, 다양한 하이퍼 파라미터를 세팅하여 비교하지 못하여 알고리즘의 효과성을 충분히 입증하지 못함

03

### 후속 연구

CoT, 추론과 관련된 다른 연구를 찾아 보면 더 효율적인 Alignment 알고리즘도 존재함(APA, STAR 등)

또한 에이전틱 전략을 사용하는 Tool Calling, Multi Agent 시스템 등이 높은 성능을 보여 해당 전략들의 혼합 필요