Empowering Multi-Turn Tool-Integrated Reasoning with Group Turn Policy Optimization

Anonymous ACL submission

Abstract

Training Large Language Models (LLMs) for multi-turn Tool-Integrated Reasoning (TIR) – where models iteratively reason, generate code, and verify through execution – remains challenging for existing reinforcement learning (RL) approaches. Current RL methods, exemplified by Group Relative Policy Optimization (GRPO), suffer from coarse-grained, trajectory-level rewards that provide insufficient learning signals for complex multi-turn interactions, leading to training stagnation. To address this issue, we propose **Group Turn Policy Optimization** (**GTPO**), a novel RL algorithm specifically designed for training LLMs on multi-turn TIR tasks. GTPO introduces three key innovations: (1) *turn-level reward assignment* that provides fine-grained feedback for individual turns, (2) *return-based advantage estimation* where normalized discounted returns are calculated as advantages, and (3) *self-supervised reward shaping* that exploits self-supervision signals from generated code to densify sparse binary outcome-based rewards. Our comprehensive evaluation demonstrates that GTPO outperforms GRPO by 3.0% across diverse math reasoning benchmarks, establishing its effectiveness. GTPO also improves GRPO by 3.9% on commonsense reasoning and program synthesis tasks, demonstrating its generalizability to non-math domains. Importantly, GTPO incurs negligible overhead, ensuring its practicality for real-world scenarios.

1 Introduction

Reinforcement learning (RL) has become a powerful training technique to improve language model reasoning capabilities, enabling these models to generate long and complex chains of thoughts (Jaech et al., 2024; Team, 2025; DeepSeek-AI et al., 2025). To improve model reasoning beyond its natural language form, recently Jin et al. (2025); Feng et al. (2025) adopted tool-using strategies (Chen et al., 2023; Gao et al., 2023) and optimized language models for tool-integrated reasoning (TIR). In domains that require intense and symbolic reasoning, TIR can facilitate precise and numerical validations between intermediate reasoning steps (Hendrycks et al., 2021; He et al., 2024). Figure 1 provides a motivating example task with TIR.

In TIR, the integration of tools provides an executable interface a model can interact with across multiple turns. In each turn, the model can iteratively evoke tools, receive tool output results, and revise its reasoning accordingly. As this multi-turn extension inherently complicates LLM reasoning trajectories, we observed severe issues when applying advanced RL algorithms such as Group Relative Policy Optimization (GRPO) and its variants (Shao et al., 2024; Liu et al., 2025; Yu et al., 2025) for TIR. Specifically, we observed empirically that models' performance often stops improving effectively when training with GRPO for multi-turn TIR, even with continued learning iterations.

We identify two major challenges in existing RL approaches for TIR tasks. Firstly, current RL algorithms such as GRPO (Shao et al., 2024) adopt a simple *sequence-level* reward assignment, and represent the advantage of each token using the normalized sequence-level *reward*. While scalable, this assignment strategy introduces arbitrary and noisy reward feedback in multi-turn TIR. Specifically, depending on the tool execution outputs in each turn, the model can reflect and revise its reasoning steps significantly in the subsequent turns. Potentially, this dynamic model behavior can drastically shift the underlying reward contribution across all reasoning turns. In this case, we found that a *turn-wise* reward assignment strategy along with *return*-based advantage is more appropriate.

Furthermore, we observe that existing work often leverages a simple binary outcome reward based on the accuracy of the final response (DeepSeek-AI et al., 2025; Feng et al., 2025).

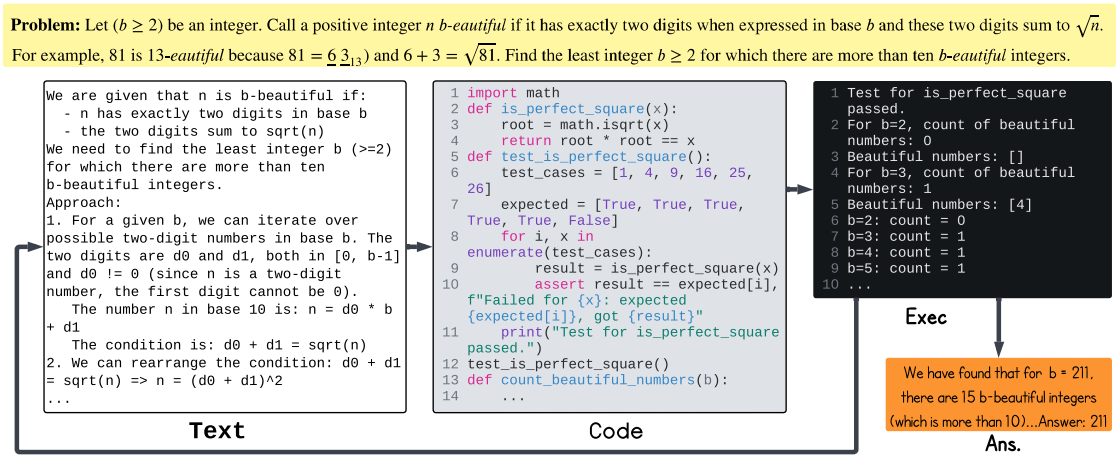

Figure 1: **Tool-integrated reasoning (TIR)**: Given a problem, the model progresses over multiple turns, where each turn consists of: (1) generating textual reasoning, (2) invoking tools (e.g., code), and (3) incorporating tool execution results to refine its understanding. The model repeats this cycle until a termination condition is met, either by producing a final answer or by reaching a predefined stopping criterion.

While accurate and efficient, such a binary sparse reward quickly becomes insufficient to bring enough learning signals in RL training for multi-turn TIR tasks. Specifically, simply assigning zero to incorrect trajectories neglects the fact that trajectories with wrong final answers may still be partially correct and contain valuable learning signals, thus making RL training signals too sparse for LLMs to learn multi-turn TIR well. Clearly, it's important to explore new reward shaping techniques that can densify the sparse binary outcome reward while maintaining its accuracy and efficiency.

To address the above challenges, we propose **Group Turn Policy Optimization** (**GTPO**). Unlike existing approaches that rely on trajectory-level rewards (Feng et al., 2025), we introduce a fine-grained turn-level reward function that assigns diverse distinct rewards for individual turns within each trajectory. Building upon research in conventional multi-turn RL (Shani et al., 2024; Gao et al., 2025), we enable turn-level discounting to calculate return-based advantages. Essentially, this reward assignment strategy and return-based advantage addressed not only the non-uniformity of reward distributions in TIR but also the temporal shift of rewards throughout the reasoning process.

Additionally, GTPO introduces a novel reward shaping technique that leverages self-supervision signals based on tool-calling contents in each turn. Specifically, different from directly assigning zero reward to negative trajectories in outcome-based reward, we leverage the accumulated code contents in TIR trajectories to compute mean similarity scores between negative and positive trajectories during training, and use such similarity scores as the partial rewards for negative trajectories. We found this reward modeling strategy simple yet effective in augmenting conventional sparse binary outcome-based rewards and maintaining their original accuracy, without losing any noticeable efficiency.

Our comprehensive evaluation demonstrates that GTPO achieves 3.0% relative improvement over GRPO on five diverse math reasoning benchmarks . GTPO also improves GRPO by 3.9% on commonsense reasoning and program synthesis tasks, demonstrating its generalizability to non-math domains. Furthermore, GTPO incurs negligible overhead, establishing GTPO as a promising solution for the real world.

2 Related Work

Related to our work is the research of reasoning abilities in Large Language Models (LLMs). Early research work have demonstrated critical model behaviors such as reflection, deliberation, and correction (Wei et al., 2022; Shinn et al., 2023). Yao et al. (2023); Snell et al. (2024) extended this line of research with inference-time scaling, in which sophisticated search techniques or nontrivial computational resources are deployed to control and manipulate model behaviors during reasoning processes. More related to our work is the research for model training techniques to improve model reasoning (Zelikman et al., 2022; Wang et al., 2024; Trung et al., 2024; Xie et al., 2024; Kumar et al., 2025). Recently, Team (2025); DeepSeek-AI et al.

Feature / Dimension	GTPO (Ours)	GRPO / DAPO / GSPO	ReTool / ToRL / Search-R1
MDP Formulation	✔ Turn-level	✗ Trajectory-level	✗ Trajectory-level
Reward Granularity	✔ Dense (Self-Supervised Shaping)	✗ Sparse (Binary)	✗ Sparse (Binary)
Advantage Estimate	✔ Discounted Return	✗ Broadcast Uniform Reward	✗ Broadcast Uniform Reward

Table 1: **A comparison between GTPO and prior works:** different from prior works, GTPO improves multi-turn RL training by introducing turn-level MDP formulation (§4.1), return-based advantage estimates (§4.2), and denser non-binary rewards based on self-supervised reward shaping (§4.3).

(2025); Team et al. (2025) showed that with appropriate reward functions and scalable RL training, LLMs can learn from its past reasoning, improving the reasoning qualities with naturally emerging behaviors and solving very complex tasks.

Also related to our work is the research on tool-integrated reasoning (TIR). Chen et al. (2023); Gao et al. (2023) proposed test-time strategies to integrate tools to solve mathematical problems. These tools can provide precise numerical validation at each step. More related to our work is the research on RL training strategies for TIR. Retool (Feng et al., 2025), ToRL (Li et al., 2025), and Search-R1 (Jin et al., 2025) used RL with a binary and trajectory-level reward function based on the accuracy of final answers. Extending from these approaches, we introduce GTPO with carefully designed turn-level reward assignment, discounted return-based advantage, and self-supervised reward shaping strategies to address the multi-turn nature and sparse reward feedback in TIR tasks. We conduct a systematic comparison of GTPO and related works in Table 1 and Appendix A.1.

3 Background

Preliminaries. We define a language model parameterized by θ as a policy π_θ. We denote x as input to the model. The likelihood under π_θ to obtain a response y from x is: $\pi_\theta(y|x) = \prod_{t=1}^{|y|} \pi_\theta(y_t|x, y_{<t})$ where $|y|$ is the number of tokens in y, y_t denotes the t-th token in y, and $y_{<t}$ represents the part of y before the t-th token. Typically, in reasoning tasks, a verifier v is available to assess the accuracy of the generated answer, resulting in a reward r. A simple verifier provides binary rewards $r \in \{0, 1\}$ where 1 is for an answer that exactly matches the ground truth, and 0 otherwise.

Tool-integrated reasoning (TIR). TIR (Chen et al., 2023; Gao et al., 2023) enhances language models with external tools to improve reasoning capabilities. TIR naturally decomposes output y as a sequence of n "turns": $y = \{y_1, b_1, y_2, b_2, \cdots, y_n\}$ and each turn y_j $(j < n)$ can be represented by $y_j = \{t_j, c_j\}$, where t_j is the natural language reasoning, c_j as the tool invocation, and b_j as the feedback of tools execution. The last turn y_n will consist of natural language only: $y_n = t_n$, in which the final answer is extracted.

Group Relative Policy Optimization (GRPO). Recently, Shao et al. (2024) proposed Group Relative Policy Optimization (GRPO) to scale RL training by removing the need to train and maintain a value model to estimate the rewards. Specifically, GRPO adopts group-based advantage estimation with the following objective [1]:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{x,\{y_i\}}[\frac{1}{\sum_{i=1}^{G}|y_i|}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\min \left(w_{i,t}(\theta)A_{i,t},\ \text{clip}\left(w_{i,t}(\theta), 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}}\right)A_{i,t}\right)] \tag{1}$$

where G is the number of responses generated from LLM for an input query x, ε is a clipping range, and $w_{i,t}(\theta)$ is the importance ratio of the token $y_{i,t}$:

$$w_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})} \tag{2}$$

Unlike PPO (Ouyang et al., 2022), $A_{i,t}$ in GRPO is measured as the group-based relative advantage:

$$A_{i,t} = A_i = \frac{r(x, y_i) - \text{mean}\left(\{r(x, y_i)\}_{i=1}^{G}\right)}{\text{std}\left(\{r(x, y_i)\}_{i=1}^{G}\right)}, \tag{3}$$

where each token $y_{i,t}$ within the same output i shares the same advantage A_i. Following existing work (Yu et al., 2025), we remove KL regularization from the training objective in our experiments.

4 Group Turn Policy Optimization

Motivated by GRPO 's oversight of the multi-turn nature of TIR tasks, we propose Group Turn Policy Optimization (GTPO) to better fine-tune LLMs through the following strategies: (i) *turn-level reward assignment* (§4.1) assigns individual reward

[1]For notation, we simplify the details of data distributions under $\mathbb{E}$: $x \sim \mathcal{D}$, $\{y_i\} \sim \pi_{\theta_{\text{old}}}$ where $\mathcal{D}$ is the training dataset. Different from vanilla GRPO, we include several improvements DAPO (Yu et al., 2025) introduces in the objective.

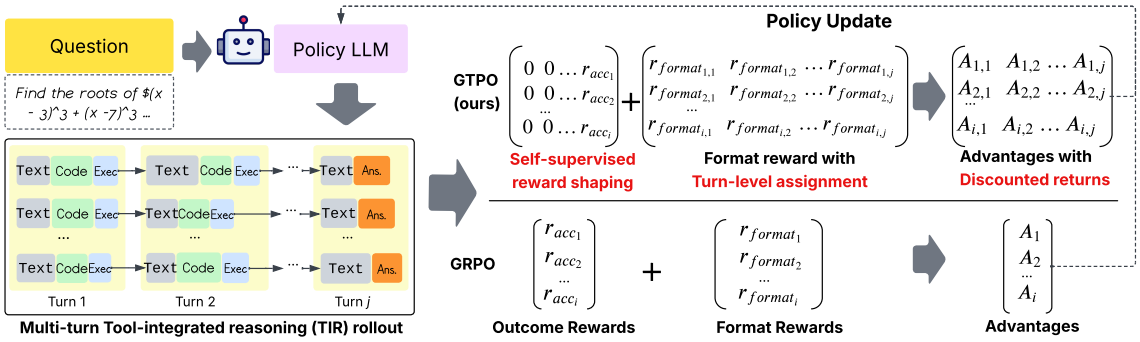

Figure 2: **An overview of GTPO**: Unlike existing approaches that rely on trajectory-level rewards, GTPO introduces a turn-level reward function that assigns diverse, rule-based rewards for individual turns within each trajectory and performs turn-level return-based discounting for advantage calculation.

r_i to individual turn y_i; (ii) *advantage with discounted return* (§4.2) is measured by a discount factor γ based on the corresponding turn-wise position; and (iii) *self-supervised reward shaping* (§4.3) augments conventional binary reward models with a self-supervised scoring method. Figure 2 illustrates the overall framework, and Figure 3 details the reward shaping mechanism. In all the figures and equations, we use red to indicate the specific terms that differ between GRPO and GTPO. We provide pseudocode (Algorithm 1 in the Appendix) to further clarify the workflow of GTPO.

4.1 Turn-level Reward Assignment

In the common practice of GRPO (DeepSeek-AI et al., 2025; Feng et al., 2025), a single terminal reward r_i is assigned to a whole TIR trajectory i:

$$r_i = \min(r_{\text{acc}_i}, r_{\text{format}_i}) \tag{4}$$

$$r_{\text{acc}_i} = \begin{cases} 0 & \text{if final answer is wrong} \\ 1 & \text{otherwise} \end{cases},$$

$$r_{\text{format}_i} = \begin{cases} 0 & \text{if trajectory has a format error} \\ 1 & \text{otherwise} \end{cases}$$

GRPO defines TIR with the following MDP:

- **State:** the initial input prompt x as the state s
- **Action:** the whole TIR trajectory $y_i = \{y_{i,1}, b_{i,1}, y_{i,2}, b_{i,2}, \cdots, y_{i,T_i}\}$ as one action a_i (T_i refers to the number of turns in y_i)

While straightforward, this sequence-level reward assignment may introduce arbitrary and noisy feedback in the long multi-turn trajectory in TIR. Based on this observation, we propose a turn-level reward assignment that measures individual reward r_j at turn j. Correspondingly, instead of treating the TIR task as a simple bandit problem, we construct a new MDP as follows, where we treat each turn y_j in the trajectory as a separate action:

- **State:** the input prompt of the j-th turn $x_j = \{x, y_{i,1}, b_{i,1}, \cdots, y_{i,j-1}, b_{i,j-1}\}$ as the state $s_{i,j}$ in the i-th trajectory
- **Action:** the j-th turn of the i-th trajectory $y_{i,j} = \{t_{i,j}, c_{i,j}\}$ as the action $a_{i,j}$

Specifically, extending from the reward Eq.4 from GRPO, we can define $r_{i,j}$ for each turn j in one TIR trajectory i as follows, and more details have been included in Appendix A.2:

$$r_{i,j} = r_{\text{acc}_{i,j}} + r_{\text{format}_{i,j}} \tag{5}$$

$$r_{\text{acc}_{i,j}} = \begin{cases} 0 & \text{if } y_{i,j} \text{ is not the last turn} \\ 0 & \text{if } y_{i,j} \text{ is the last turn and final answer is wrong} \\ 1 & \text{otherwise} \end{cases},$$

$$r_{\text{format}_{i,j}} = \begin{cases} -0.1 & \text{if } y_{i,j} \text{ contains format errors} \\ 0 & \text{otherwise} \end{cases}$$

4.2 Advantage with Discounted Return

While the turn-level reward assignment strategy accounts for more fine-grained rewards in individual turns, it does not account for the sequential order of turns. Motivated by traditional RL practice (Shani et al., 2024; Gao et al., 2025), we propose to incorporate the temporal effect between turns through a discounting factor γ in the reward formula:

$$R_{i,j} = \sum_{m=j}^{T_i} \gamma^{m-j} r_{i,m}, \tag{6}$$

where $r_{i,m}$ refers to the reward of turn y_m in trajectory i. Essentially, the discounting factor γ can systematically discount the value of future rewards with a diminishing values turn-by-turn. At each turn j, a return (*i.e.,* reward-to-go) R_j is the sum of individual discounted rewards from the current turn until the terminal turn. From Eq.3, the GTPO

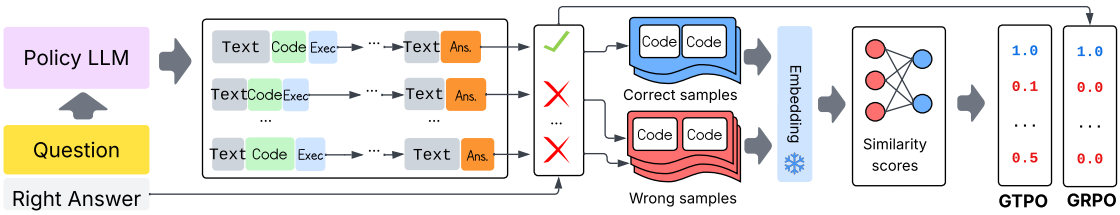

Figure 3: **GTPO reward shaping strategy**: In GTPO, each rollout trajectory is partitioned by final outcome (correct vs. incorrect), and the code content is extracted. For each trajectory in the incorrect group, we compute its average similarity against all samples in the correct group and use the similarity score as its partial reward, so that wrong trajectories can still be properly utilized during training for more learning signals.

advantages are then updated as:

$$A_{i,j,t} = A_{i,j} = \frac{R_{i,j} - \text{mean}\left(\{R_{i,j}\}_{\forall i,j}\right)}{\text{std}\left(\{R_{i,j}\}_{\forall i,j}\right)}, \quad (7)$$

where the advantage normalization is performed globally: we aggregate the advantages from all turns across all sampled trajectories and compute a single mean and standard deviation over this pooled set. This global normalization scheme naturally handles trajectories of varying lengths, as each turn-level advantage is treated as an independent sample in the shared normalization pool.

This turn-wise reward assignment with the discounting strategy addresses not only the non-uniformity of reward distributions in multi-turn TIR but also the temporal shift of rewards across reasoning steps throughout the reasoning process. We further study the relationship between the values of γ and the model's performance after RL training. As shown in §5.2, choosing the right γ is crucial to the final success of GTPO training.

4.3 Self-supervised Reward Shaping

GRPO estimates advantages from the binary outcome reward (DeepSeek-AI et al., 2025). While straightforward, such a sparse reward cannot bring enough learning signals for effective RL training. Specifically, simply assigning $r_{\text{acc}} \in \{0, 1\}$ neglects the fact that trajectories with wrong final answers may still be partially correct and contain good supervision data (e.g., partially correct code). Assigning strict $r_{\text{acc}} = 0$ to failed trajectories makes RL training signals too sparse for LLMs to learn well. As such, we propose a simple yet effective strategy to assign partial rewards for r_{acc} of the failed trajectories. See Figure 3 for an overview.

Specifically, given a rollout sample i, we first extract and concatenate the tool invocation (code) contents: $c_{i,0} \oplus c_{i,1} \oplus \cdots \oplus c_{i,T_i-1}$. From a group of G samples, we filter for samples with the final predicted answer matching the ground truth answer and denote this set of positive samples as $\mathcal{P}$. We then compute the partial reward as the mean similarity score between a negative code sample against all positive code samples. Essentially, extending $r_{\text{acc}i,j}$ when $y_{i,j}$ is the last turn from Eq.5:

$$r_{\text{acc}i,j} = \frac{\alpha}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \text{sim}\Big(\bigoplus_{m<j} c_{i,m}, \bigoplus_{m<j} c_{p,m}\Big) \text{ if } i \notin \mathcal{P} \quad (8)$$

Note that $r_{\text{acc}i,j} = 1.0$ if $i \in \mathcal{P}$, and α is the upper bound hyperparameter of the partial rewards. $\text{sim}(.,.)$ returns a similarity score between two input components, which can be any efficient off-the-shelf embedding model. When $|\mathcal{P}| = 0$ (*i.e.,* the rollout group has no correct samples), no partial rewards will be assigned. Apart from the final result of the ground truth to obtain the positive set $\mathcal{P}$, our reward shaping strategy does not need additional external supervision to measure the turn-wise reward, demonstrating its simplicity and efficiency to apply. In practice, we set $\alpha = 0.5$, which achieves good results across benchmarks empirically.

4.4 Group Turn Policy Optimization

Adopting turn-level reward assignment, advantage with discounted return, and self-supervised reward shaping, we can obtain the final training objective of GTPO by extending from Eq.1 as follows:

$$\mathcal{J}_{\text{GTPO}}(\theta) = \mathbb{E}_{x,\{y_i\}}\Big[\frac{1}{\sum_{i=1}^{G}|y_i|} \sum_{i=1}^{G} \sum_{j=1}^{T_i} \sum_{t=1}^{|y_{i,j}|} \min \Big(w_{i,j,t} A_{i,j},\ \text{clip}(w_{i,j,t}, 1-\varepsilon_{\text{low}}, 1+\varepsilon_{\text{high}}) A_{i,j}\Big)\Big] \quad (9)$$

where $y_{i,j}$ denotes the turn j in sample i: $y_{i,j} = t_{i,j} c_{i,j}$, and T_i refers to the total number of turns in the sample i. Note that in Eq.9, we still adopt the original formula Eq.2 for the importance ratio $w_{i,j,t}$ and normalize the objective loss by the total number of tokens $\sum_{i=1}^{G}|y_i|$. We also conduct an overhead analysis of GTPO in Appendix A.3.

Models	AIME 2024 (*avg@16*)	AIME 2025 (*avg@16*)	MATH 500 (*pass@1*)	AMC 2023 (*avg@16*)	SVAMP (*pass@1*)	Average
Qwen2.5-7B-Instruct	6.70	10.00	73.49	50.00	93.80	46.80
+ TIR Prompting	16.04	14.38	67.41	47.66	92.80	47.66
+ GRPO	20.63	16.25	70.25	54.38	87.40	49.78
+ GTPO (ours)	**22.29**	**16.88**	**72.78**	**54.53**	**89.80**	**51.26**

Table 2: **Main experimental results:** we report the passing rate results of TIR Prompting, GRPO, and GTPO on five diverse mathematical reasoning benchmarks. We report either the *avg@k* or *pass@k* and $k = \{1, 16\}$.

5 Experiments

5.1 Experimental Setup

Cold-Start SFT Training. Following existing work, we first constructed our cold-start dataset based on ReTool-SFT (Feng et al., 2025). Specifically, we extracted problems from ReTool-SFT, distilled TIR trajectories from DeepSeek-R1 (DeepSeek-AI et al., 2025) with OpenHands (Wang et al., 2025) as the scaffold, and conducted rejection sampling by filtering out trajectories whose final answers were incorrect. In the end, our cold-start dataset contains 1.2K problem-trajectory pairs, and we performed SFT on Qwen2.5-7B-Instruct (Qwen et al., 2025) for three epochs.

RL Training. We used DAPO-17K (Yu et al., 2025) as our RL training dataset and implemented GTPO by extending SkyRL (Cao et al., 2025). For self-supervised reward shaping in GTPO, we used Amazon Titan Text Embeddings V2[2] to embed the generated code and calculate the similarities between embedding vectors. During training, we utilized the AdamW optimizer with an initial learning rate of 1e-6. We defined the maximum sequence length for each turn as 8192 tokens, the maximum number of turns as 3, and the mini-batch size as 1024. Note that we set the KL coefficient to 0.0.

Evaluation. To evaluate the effectiveness of GTPO, our evaluation focuses on mathematical reasoning tasks in five diverse math reasoning benchmarks: AIME 2024, AIME 2025[3], MATH 500 (Lightman et al., 2023), AMC 23[4], and SVAMP (Patel et al., 2021). We provide the detailed evaluation settings in Appendix A.4. We considered the following three baselines in our experiments:

- **Simple Prompting:** Chain-of-thought reasoning without any tool calling.
- **TIR Prompting:** Directly using the OpenHands scaffold to generate TIR trajectories without any RL training.
- **GRPO Training for TIR:** Following existing work, to apply GRPO for TIR tasks, we performed cold-start SFT and then used GRPO to further finetune the model. To make a fair comparison, the hyperparameter settings and datasets are the same as fine-tuning GTPO.

5.2 Experimental Results

From Table 2, we observed the following: first, directly prompting a conventional language model with TIR is not sufficient, increasing the average passing rate only from 46.80% (*i.e.,* simple prompting) to 47.66%. This observation indicates the need of finetuning language models for TIR to solve complex reasoning tasks. Secondly, we found that finetuning with our GTPO consistently pushed the performance across all benchmarks, achieving 51.26% passing rate on average. This result outperformed GRPO (49.78%) by 3.0% relative improvement, with more significant performance gains in AIME 2024, MATH 500, and SVAMP. Note that we maintained the same training setup between GRPO and GTPO to ensure a fair comparison.

Qualitative analysis. In Figure 4, we showed an example task from AIME 2024 and the critical distinction between code generated by models trained with GRPO and with GTPO. This task requires finding the least integer $b \geq 2$ for which there are more than ten b-eautiful integers. An integer n is b-eautiful if it has exactly two digits in base b and they sum to $\sqrt{n}$. The GRPO-trained model demonstrates a fundamental algorithmic flaw: it attempts to validate test cases wrongly in a post-hoc manner after completing the count operation, resulting in an assertion error when $b = 13$ yields only 3 beautiful integers. In contrast, the GTPO-trained model implements more robust test case

[2] https://docs.aws.amazon.com/bedrock/latest/userguide/titan-embedding-models.html

[3] https://huggingface.co/datasets/AI-MO/aimo-validation-aime

[4] https://huggingface.co/datasets/zwhe99/amc23

Problem: Let ($b \geq 2$) be an integer. Call a positive integer n *b-eautiful* if it has exactly two digits when expressed in base b and these two digits sum to $\sqrt{n}$. For example, 81 is 13-*eautiful* because $81 = \underline{6}\,\underline{3}_{13}$ and $6 + 3 = \sqrt{81}$. Find the least integer $b \geq 2$ for which there are more than ten *b-eautiful* integers.

GRPO Solution

```python
1 def count_beautiful_integers(b):
2     count = 0
3     for k in range(1, b):
4         numerator = k * b - k**2
5         if numerator % (b-1) == 0:
6             c_val = numerator // (b-1)
7             if 0 <= c_val < b:
8                 n = k**2
9                 digits = []
10                temp = n
11                while temp:
12                    digits.append(temp % b)
13                    temp //= b
14                if len(digits) == 2 and digits[0] >= 1:
15                    if sum(digits) == k:
16                        count += 1
17    return count
18 def test_example():
19    b = 13
20    print(f"Testing b=13: Expected 1 (for 81)")
21    result = count_beautiful_integers(b)
22    print(f"Result for b=13: {result}")
23    assert result == 1, f"Expected 1 for b=13, got
   {result}"
24 test_example()
25 ......
```

```
Testing b=13: Expected 1 (for 81)
Result for b=13: 3
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[1], line 43
     40     print(f"Result for b=13: {result}")
     41     assert result == 1, f"Expected 1 for b=13, got {result}"
---> 43 test_example()
     45 # Now we need to find the least b for which the count is greater than 10
     46 b = 2

Cell In[1], line 41, in test_example()
     39 result = count_beautiful_integers(b)
     40 print(f"Result for b=13: {result}")
---> 41 assert result == 1, f"Expected 1 for b=13, got {result}"

AssertionError: Expected 1 for b=13, got 3
```

GTPO Solution

```python
1 import math
2 def is_perfect_square(x):
3     root = math.isqrt(x)
4     return root * root == x
5 def test_is_perfect_square():
6     test_cases = [1, 4, 9, 16, 25, 26]
7     expected = [True, True, True, True, True,
   False]
8     for i, x in enumerate(test_cases):
9         result = is_perfect_square(x)
10        assert result == expected[i], f"Failed
   for {x}: expected {expected[i]}, got {result}"
11    print("Test for is_perfect_square passed.")
12 test_is_perfect_square()
13 def count_beautiful_numbers(b):
14    count = 0
15    beautiful_n = []
16    for d0 in range(1, b):
17        for d1 in range(0, b):
18            n = d0 * b + d1
19            if is_perfect_square(n):
20                root = math.isqrt(n)
21                if root == d0 + d1:
22                    count += 1
23                    beautiful_n.append(n)
24    return count, beautiful_n
25 ......
```

```
Test for is_perfect_square passed.
For b=2, count of beautiful numbers: 0
Beautiful numbers: []
For b=3, count of beautiful numbers: 1
Beautiful numbers: [4]
b=2: count = 0
b=3: count = 1
b=4: count = 1
b=5: count = 1
b=6: count = 1
b=7: count = 3
b=8: count = 1
b=9: count = 1
b=10: count = 1
b=11: count = 3
b=12: count = 1
b=13: count = 3
...
```

Figure 4: **Qualitative example:** We demonstrated an AIME24 example task to compare the distinct coding patterns of GRPO and GTPO. Qwen2.5-7B-Instruct trained with GTPO can write correct code along with accurate tests that thoroughly validate the code correctness, while Qwen2.5-7B-Instruct trained with GRPO fails to solve the problem.

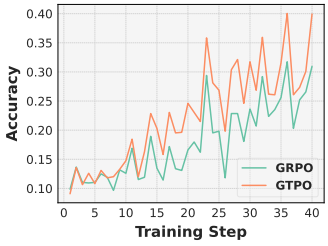

Figure 5: Training accuracy curves of GRPO and GTPO under the same experimental setup and training datasets.

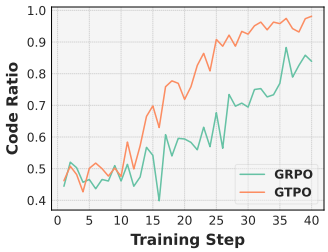

Figure 6: Code ratio curves of GRPO and GTPO during training: the code ratio refers to the percentage of rollout trajectories that contain some code content in reasoning.

Turn-level Reward	Adv. w/ Disct. Ret.	Reward Shaping	MATH 500 (*pass@1*)	AIME 2024 (*avg@16*)	Average
✗	✗	✗	67.09	20.63	43.86
✓	✗	✗	72.15	20.21	46.18
✓	✓	✗	69.94	20.63	45.29
✓	✓	✓	**72.47**	**21.25**	**46.86**

Table 3: **Ablation results of GTPO**: we reported the results with Qwen2.5-7B-Instruct when removing major components from GTPO: (i) turn-level reward assignment, (ii) advantage with discount return, and (iii) our reward shaping strategy.

Discounting factor (γ)	MATH 500 (*pass@1*)	AIME 2024 (*avg@16*)	Average
0.5	70.57	20.21	45.39
0.7	72.78	17.50	45.14
0.9	**72.47**	**21.25**	**46.86**
1.0	72.15	20.21	46.18

Table 4: **GTPO with different discounting factors** (γ): we reported the results on Qwen2.5-7B-Instruct with $\gamma \in [0.5, 1]$, where $\gamma = 1$ refers to no discounting in advantage estimate.

validation directly into the search loop (lines 8-10), allowing it to correctly verify the correctness of intermediate results. The GTPO model successfully identifies that $b = 2$ produces 11 beautiful integers (as shown in the test output), satisfying the problem's requirements. This example task underscores how finetuning with GTPO can correctly incorporate test-driven validation as an integral part of the solution process, indicating superior reasoning enhancement for language models than GRPO.

Training Curves. We investigated the training progress of GRPO and GTPO in Figure 5. Compared with GRPO, GTPO demonstrates both higher peak performance (reaching approximately 40% accuracy) as well as greater volatility. This observation suggests more aggressive optimization and exploration of GTPO for RL training. GRPO exhibits a more conservative learning pattern as we found its accuracy plateauing around 25% - 30% in the later stages. This persistent performance gap between GRPO and GTPO demonstrates the effective use of reward feedback from our method to improve the model reasoning capabilities.

Figure 6 compares the code ratio curves of GRPO and GTPO. GTPO demonstrates a more aggressive shift toward code-based reasoning, reaching nearly 98% code ratio by training step 40 with a relatively consistent upward trend. In contrast, GRPO exhibits a more conservative learning pattern, plateauing around 85%, suggesting potential instability or exploration-exploitation trade-offs in its optimization process. Compared with GRPO, training with GTPO significantly pushes the model to use code as an external tool for TIR tasks.

Ablation Studies. We performed ablation studies to investigate the design choices of GTPO. All the ablation experiments start from Qwen2.5-7B-Instruct after cold-start SFT, following the same settings described in §5.1. First, we conducted an experiment to study the effect of return discounting and self-supervised reward shaping in GTPO. As shown in Table 3, the model trained with both return discounting and self-supervised reward shaping achieves the best evaluation results on MATH 500 and AIME 2024, achieving 46.86% passing rate on average. When we removed these components gradually, we observed negative performance impacts with more significant performance drops when removing both turn-level reward and reward shaping strategies. These observations highlight the importance of different components in GTPO for multi-turn TIR tasks.

Impact of Discounting Factor We studied the effect of the discounting factor γ. In our experiment, we study four different values of the discounting factor: $\gamma = \{0.5, 0.7, 0.9, 1.0\}$. As shown in Table 4, $\gamma = 0.9$ achieves the optimal balance between distant rewards and immediate rewards, outperforming the conventional approach in GRPO (where $\gamma = 1.0$). When $\gamma < 0.9$, we observed negative performance impacts with a reduction of passing rate up to 1.72%. These observations demonstrate the importance of the discounting factor for any turn-based RL approaches like GTPO.

For more experimental results, please refer to Appendix A.5, A.6, A.7, and A.8.

6 Discussion

6.1 Generalization across Model Families

To assess the generality of GTPO across architectures and scales, we additionally trained Llama-3.2-3B-Instruct with both GRPO and GTPO and evaluated on AIME 2024 (avg@16) and MATH 500 (pass@1). As shown in Table 5, GTPO consistently outperforms GRPO, improving accuracy from 21.18% to 23.01% on average, supporting our claim that GTPO generalizes across diverse LLM families and model sizes.

6.2 Generalization to Non-Math Domains

We extended our evaluation beyond mathematical reasoning to assess GTPO on two additional task families: commonsense reasoning and program synthesis. For commonsense reasoning, we evaluated on GPQA Diamond (Rein et al., 2023) and BBEH-mini (Kazemi et al., 2025), using tool-integrated reasoning for each problem. For program synthesis, we use EvalPlus (Liu et al., 2023), including HumanEval and MBPP, which is a natural fit since TIR-trained models tend to develop stronger Python-based problem-solving skills.

Llama3.2-3B-Instruct	**MATH 500** (*pass@1*)	**AIME 2024** (*avg@16*)	**Average**
+ GRPO	37.97	4.38	21.18
+ GTPO (ours)	**39.56**	**6.46**	**23.01**

Table 5: **Generalization across model families:** we validated the generalizability of GTPO across model families and sizes. On Llama-3.2-3B-Instruct, GTPO consistently outperforms GRPO by 8.6% on average.

As shown in Table 6, across all four benchmarks, GTPO consistently outperforms GRPO with 3.9% relative improvement on average. This shows that GTPO can generalize beyond mathematical reasoning and provide gains for non-math tasks.

Qwen2.5-7B	**GPQA** (*pass@1*)	**BBEH** (*pass@1*)	**HumanEval** (*pass@1*)	**MBPP** (*pass@1*)	**Avg.**
+ GRPO	21.7	4.1	77.4	81.0	46.1
+ GTPO (ours)	**24.2**	**4.4**	**81.7**	**81.2**	**47.9**

Table 6: **Generalization to non-math domains:** we evaluate the generalizability of GTPO to commonsense reasoning and program synthesis tasks. GTPO maintains stable improvements over GRPO with 3.9% relative improvement on average.

7 Conclusion

In this work, we addressed the challenge of training language models for multi-turn Tool-Integrated Reasoning through RL. Our solution, **Group Turn Policy Optimization (GTPO)**, introduces turn-level reward functions with rule-based rewards for individual turns and turn-level reward discounting for advantage calculation, overcoming trajectory-level reward limitations. Additionally, our reward shaping technique uses self-supervision signals from generated code to densify sparse binary rewards, improving learning efficiency. Empirical results demonstrate that GTPO achieves 3.0% relative improvement on average over GRPO on five diverse math reasoning benchmarks. GTPO also improves GRPO by 3.9% on commonsense reasoning and program synthesis tasks, demonstrating its generalizability to non-math domains. Furthermore, GTPO incurs negligible overhead, setting it as a new advanced RL technique to improve model reasoning in the real world.

Limitations

While GTPO has proven effective through extensive evaluation in the paper, our experiments are restricted to models no larger than 7B parameters due to the computation budget. It is prohibitively expensive to perform large-scale RL experiments for LLMs, and unfortunately, we do not have enough resources to demonstrate the impact of GTPO on larger models. In addition, this work has mainly focused on Tool-Integrated Reasoning tasks, but the idea of GTPO can be broadly applicable for improving models' reasoning capability in general multi-turn scenarios such as real-world software engineering tasks, which we leave to future work.

References

Shiyi Cao, Sumanth Hegde, Dacheng Li, Tyler Griggs, Shu Liu, Eric Tang, Jiayi Pan, Xingyao Wang, Akshay Malik, Graham Neubig, Kourosh Hakhamaneshi, Richard Liaw, Philipp Moritz, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Skyrl-v0: Train real-world long-horizon agents via reinforcement learning.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, and 81 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *CoRR*, abs/2501.12948.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *Preprint*, arXiv:2504.11536.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10764–10799. PMLR.

Zhaolin Gao, Wenhao Zhan, Jonathan Daniel Chang, Gokul Swamy, Kianté Brantley, Jason D. Lee, and Wen Sun. 2025. Regressing the relative future: Efficient policy optimization for multi-turn RLHF. In *The Thirteenth International Conference on Learning Representations*.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. 2024. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3828–3850, Bangkok, Thailand. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, and 1 others. 2024. Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, 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*.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K. Jain, Virginia Aglietti, Disha Jindal, Peter Chen, Nishanth Dikkala, Gladys Tyen, Xin Liu, Uri Shalit, Silvia Chiappa, Kate Olszewska, Yi Tay, Vinh Q. Tran, Quoc V. Le, and Orhan Firat. 2025. Big-bench extra hard. *Preprint*, arXiv:2502.19187.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2025. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations*.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. 2025. Torl: Scaling tool-integrated rl. *Preprint*, arXiv:2503.23383.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. In *2nd AI for Math Workshop @ ICML 2025*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, and 1 others. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *Preprint*, arXiv:2103.07191.

Eduardo Pignatelli, Johan Ferret, Matthieu Geist, Thomas Mesnard, Hado van Hasselt, Olivier Pietquin, and Laura Toni. 2024. A survey of temporal credit assignment in deep reinforcement learning. *Preprint*, arXiv:2312.01072.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *Preprint*, arXiv:2311.12022.

Lior Shani, Aviv Rosenberg, Asaf Cassel, Oran Lang, Daniele Calandriello, Avital Zipori, Hila Noga, Orgad Keller, Bilal Piot, Idan Szpektor, Avinatan Hassidim, Yossi Matias, and Remi Munos. 2024. Multi-turn reinforcement learning with preference human feedback. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, and 1 others. 2025. Kimi k1.5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

Qwen Team. 2025. Qwq-32b: Embracing the power of reinforcement learning.

Luong Trung, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. ReFT: Reasoning with reinforced fine-tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7601–7614, Bangkok, Thailand. Association for Computational Linguistics.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024. Mathcoder: Seamless code integration in LLMs for enhanced mathematical reasoning. In *The Twelfth International Conference on Learning Representations*.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025. Openhands: An open platform for AI software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *Preprint*, arXiv:2502.18449.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P Lillicrap, Kenji Kawaguchi, and Michael Shieh. 2024. Monte carlo tree search boosts reasoning via iterative preference learning. In *The First Workshop on System-2 Reasoning at Scale, NeurIPS'24*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, and 1 others. 2025. Dapo: An open-source llm reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STar: Bootstrapping reasoning with

reasoning. In *Advances in Neural Information Processing Systems*.

Dejiao Zhang, Wasi Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. 2024. Code representation learning at scale. *arXiv preprint arXiv:2402.01935*.

A Appendix

A.1 Comparison of GTPO and related works

Different from prior works, we introduced GTPO by **reformulating multi-turn TIR as a turn-level MDP with a specific, tool-aware reward structure** and integrating this with group-based policy optimization. This reformulation is simple yet critical to address a fundamental modeling issue in prior TIR works: these methods continue to treat the entire tool-using trajectory as a single action with a sparse outcome reward. Below, we discussed in detail the differences between GTPO and prior work along the following dimensions: turn-level reward assignment, discounted return for advantage calculation, and self-supervised reward shaping for negative trajectories.

A.1.1 Turn-level reward assignment

GRPO-style methods in TIR treat the entire multi-turn trace as one action with a single trajectory-level reward (e.g., final correctness), broadcasting the same scalar advantage to all tokens. Our approach (GTPO) explicitly redefines the decision process at the turn level: each turn is an action whose reward is computed from tool feedback and signals (e.g., format correctness, presence, and quality of tool calls). This is not just a cosmetic tweak of GRPO's reward; **it changes the underlying state–action structure from a bandit over full trajectories to a sequential decision process over turns, which is tailored to multi-turn TIR**. Our ablations (Table 3) show that this reformulation alone yields clear gains over trajectory-level GRPO under the same data and infrastructure.

A.1.2 Discounted return for advantage calculation

GRPO/GSPO/DAPO typically use group-normalized, trajectory-level rewards that ignore temporal structure: every token in a trajectory receives the same normalized advantage scalar. Our approach (GTPO) instead computes **discounted reward-to-go at the turn level**, so terminal success or failure propagates backward in a degraded way. To our knowledge, prior GRPO-style methods for TIR do not (i) define a turn-level MDP and simultaneously (ii) use turn-wise discounted returns within the group objective. Our ablations (Tables 3–4) show that removing discounted turn-level return degrades performance and that the discount factor γ has a

Problem: Let K be the number of sequences $A_1, A_2, \ldots, A_n$ such that n is a positive integer less than or equal to 10, each A_i is a subset of $\{1, 2, 3, \ldots, 10\}$, and A_{i-1} is a subset of A_i for each i between 2 and n, inclusive. For example, $\{\}, \{5, 7\}, \{2, 5, 7\}, \{2, 5, 7\}, \{2, 5, 6, 7, 9\}$ is one such sequence, with $n = 5$. What is the remainder when K is divided by 10?

GRPO Solution

```python
def test_last_digit_sum():
    total = 0
    for n in range(1, 11):
        # Compute 4^n mod 10
        term = pow(4, n, 10)
        total = (total + term) % 10
        print(f"n={n}: 4^{n} mod 10 = {term}")
    return total
remainder = test_last_digit_sum()
print(f"The remainder when K is divided by 10 is: {remainder}")
```

```
EXECUTION:
n=1: 4^1 mod 10 = 4
n=2: 4^2 mod 10 = 6
n=3: 4^3 mod 10 = 4
n=4: 4^4 mod 10 = 6
n=5: 4^5 mod 10 = 4
n=6: 4^6 mod 10 = 6
n=7: 4^7 mod 10 = 4
n=8: 4^8 mod 10 = 6
n=9: 4^9 mod 10 = 4
n=10: 4^10 mod 10 = 6
The remainder when K is divided by 10 is: 0
```

GTPO Solution

```python
def test_dp():
    n_max = 10
    total_subsets = 2**10
    dp = [[0] * (n_max+1) for _ in range(total_subsets)]
    for s in range(total_subsets):
        dp[s][1] = 1
    for n in range(2, n_max+1):
        for s in range(total_subsets):
            for t in range(total_subsets):
                if (t & s) == t:
                    dp[s][n] = (dp[s][n] + dp[t][n-1] % 10) % 10
    K = 0
    for s in range(total_subsets):
        for n in range(1, n_max+1):
            K += dp[s][n]
    K_mod10 = K % 10
    print(f"Test: K mod 10 = {K_mod10}")
    return K_mod10
result = test_dp()
print(f"Result: {result}")
```

```
EXECUTION:
Test: K mod 10 = 5
Result: 5
```

Figure 7: An AMC23 example to compare the distinction in generation samples between GRPO and GTPO.

clear "good" range.

A.1.3 Self-supervised reward shaping for negative trajectories

Prior TIR RL work (e.g., ReTool, ToRL, Search-R1) typically uses binary or near-binary outcome rewards, treating all incorrect trajectories as equally bad and providing little signal when all samples in a group fail. GTPO introduces self-supervised shaping for failed trajectories: we assign partial rewards based on the similarity between their code and code from successful trajectories. **This recovers informative signal from "almost correct" code and exploits the fact that code is a concise, verifiable proxy for intermediate reasoning quality.** Table 7 shows that this shaping outperforms both purely binary rewards and simpler string-matching baselines.

The combined three designs target the core RL challenge: sparse, delayed rewards and poor temporal credit assignment (Pignatelli et al., 2024). GTPO simultaneously (1) changes the MDP granularity to turns, (2) propagates credit via discounted returns over turns, and (3) densifies supervision inside failed trajectories using code-based self-supervision. To our knowledge, GTPO is the first method to instantiate this specific turn-level, discounted, self-supervised RL formulation for multi-turn TIR, and our experiments and ablations consistently show empirical gains over standard GRPO-style baselines.

A.2 Turn-level Format Reward Design

In practice, considering the nature of TIR tasks, we focus on two major format requirements: (1) the format of tool calling must be correct, and (2) there must exist at least one tool call throughout the trajectory. Specifically, we assign $r_{\text{format}_{i,j}} = -0.1$ when y_j contains any invalid tool calls. To further ensure that at least one tool call happens throughout the trajectory, we directly demand the first turn y_1 to contain tool calls: we assign $r_{\text{format}_{i,1}} = -0.1$ if y_1 does not include any tool calls. The reason is that, based on our observation, for all the trajectories that contain tool calls, models will always make tool calls in the first turn. Following DAPO (Yu et al., 2025), we use the answer format "Answer:" throughout all the evaluations.

A.3 Overhead Analysis

GTPO introduces minimal and negligible computational overhead compared to GRPO. Firstly, the overhead of turn-level reward calculation is minimal. Both the format and outcome rewards rely solely on lightweight string-matching operations; thus, extending them from the trajectory level to the turn level does not introduce any meaningful additional cost. Likewise, the overhead of our self-supervised reward shaping is negligible: it only requires computing a single embedding per sam-

Algorithm 1 Group Turn Policy Optimization (GTPO)

Require: Policy π_θ, reference policy $\pi_{\theta_{\text{old}}}$, dataset $\mathcal{D}$, group size G, discount γ, clip $\epsilon_{\text{low}}, \epsilon_{\text{high}}$, shaping cap α, learning rate η

for each update step **do**
  Sample prompt $x \sim \mathcal{D}$
  Roll out G trajectories $\{y_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$, where $y_i = \{y_{i,1}, \ldots, y_{i,T_i}\}$ and $y_{i,j} = (t_{i,j}, c_{i,j})$
  Let $\mathcal{P} \leftarrow \{i \mid \text{final answer of } y_i \text{ is correct}\}$
  for $i = 1, \ldots, G$ **do**
    for $j = 1, \ldots, T_i$ **do**

$$r_{\text{format}_{i,j}} \leftarrow \begin{cases} -0.1 & \text{if } y_{i,j} \text{ has any format error} \\ 0 & \text{otherwise} \end{cases}$$

$$r_{\text{acc}_{i,j}} \leftarrow \begin{cases} 0 & j < T_i \\ 1 & (j = T_i) \wedge (i \in \mathcal{P}) \\ \frac{\alpha}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \text{sim}\Big(\bigoplus_{m<j} c_{i,m}, \bigoplus_{m<j} c_{p,m}\Big) & (j = T_i) \wedge (i \notin \mathcal{P}) \end{cases}$$

$$r_{i,j} \leftarrow r_{\text{acc}_{i,j}} + r_{\text{format}_{i,j}}$$

$$R_{i,j} \leftarrow \sum_{m=j}^{T_i} \gamma^{m-j} r_{i,m}$$

    end for
  end for
  Compute global mean μ and std σ over all $\{R_{i,j}\}$ (all i, j)
  $A_{i,j} \leftarrow (R_{i,j} - \mu)/(\sigma + \delta)$

$$\mathcal{L}(\theta) \leftarrow -\frac{1}{\sum_i |y_i|} \sum_{i=1}^{G} \sum_{j=1}^{T_i} \sum_{t \in y_{i,j}} \min\Big(w_{i,j,t} A_{i,j},\ \text{clip}(w_{i,j,t}, 1-\epsilon_{\text{low}}, 1+\epsilon_{\text{high}}) A_{i,j}\Big)$$

  where $w_{i,j,t} = \dfrac{\pi_\theta(y_{i,j,t} \mid x, \text{history})}{\pi_{\theta_{\text{old}}}(y_{i,j,t} \mid x, \text{history})}$
  Update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}(\theta)$
end for

pled trajectory, which takes on the order of seconds and is insignificant compared to the cost of generating full trajectories.

A.4 Additional Evaluation Settings

For benchmarks where we report *avg@16* performance, including AIME 2024, AIME 2025, and AMC 2023, we set the sampling temperature to be 0.6. For other benchmarks where we report *pass@1* performance, including MATH500 and SVAMP, we set a lower sampling temperature of 0.2 for more stable evaluation results. In the evaluation, we define the maximum sequence length for each turn as 8192 tokens and the maximum number of turns as 10, allowing models for more exploration. Because the RL training set we use (*i.e.,* DAPO-17K (Yu et al., 2025)) only includes problems whose answer is a single integer, we filter out all the problems from these benchmarks whose ground truth is not a single integer. In the main experiments in Table 2, both GRPO and GTPO checkpoints we evaluate have been trained for 40 steps. Limited by computational resources, for all the other experiments, we evaluate checkpoints that have been trained for 30 steps.

A.5 Impact of Reward Shaping Strategy

We evaluated different design choices for GTPO reward shaping. Specifically, we compared character-based sequence matching via `Difflib` [5] (Wei et al., 2025) and embedding-based similarity (Zhang et al., 2024). We also varied the content scope used to compute similarity, contrasting entire trajectories (natural language + code) with code-only inputs. As shown in Table 7, representing trajectory correctness using only code components yields the largest gains. In particular, embedding-based similarity computed on code-only content achieves

[5] https://docs.python.org/3/library/difflib.html

Scoring method	Sample content	MATH 500 (*pass@1*)	AIME 2024 (*avg@16*)	Avg.
`Difflib`	Code	71.52	21.25	46.39
`Difflib`	Trajectory	72.15	18.75	45.45
Embedding	Code	**72.47**	**21.25**	**46.86**

Table 7: **GTPO by reward shaping strategies**: we change the reward shaping strategy by the scoring method (using embedding model or `Difflib`) and the data sample content (code only or the whole trajectory).

the best performance. This finding suggests that, for TIR tasks, code provides a concise and reliable feedback signal for steering model reasoning, replacing the conventional natural language data.

A.6 Comparison with Additional Baseline

To compare GTPO with more baselines, we have conducted an additional experiment comparing GTPO against DAPO on Qwen2.5-7B-Instruct, using the same training setup and computational budget in the ablation experiments (§A.4). Following standard practice, we evaluate using MATH 500 (pass@1) and AIME 2024 (avg@16). As shown in Table 8, while DAPO provides a small improvement over GRPO, GTPO still achieves the best performance, supporting our claim that GTPO offers benefits beyond standard RL objectives.

Qwen2.5-7B-Instruct	MATH 500 (*pass@1*)	AIME 2024 (*avg@16*)	Average
+ GRPO	67.09	20.63	43.86
+ DAPO	67.72	20.77	44.25
+ GTPO (ours)	**72.47**	**21.25**	**46.86**

Table 8: **Comparison with DAPO:** we add DAPO in our evaluation as an additional baseline. Results show that GTPO still consistently outperforms DAPO, indicating that our turn-level reward formulation provides complementary advantages over existing RL methods.

A.7 Additional Case Studies

In Figure 7, we show an example from AMC23, illustrating the distinct problem-solving approaches between GRPO and GTPO models in tackling combinatorial counting problems. The task requires counting sequences of subsets with specific containment properties modulo 10 - a problem that demands careful handling of the exponential growth in possibilities. The GRPO solution attempts a direct computational approach using dynamic programming with memoization, but critically fails to properly manage the modular arithmetic. Specifically, it computes the full count first and only applies the modulo operation at the end, leading to integer overflow issues that produce an incorrect result of 0. In contrast, the GTPO solution demonstrates superior algorithmic insight by maintaining the modulo 10 constraint throughout the computation within its dynamic programming table, preventing overflow and correctly identifying the answer as 5.

A.8 Training Curves

Figure 8 demonstrates the format correctness curves of GRPO and GTPO. GTPO exhibits superior performance throughout the training process, achieving a robust improvement to around 99% by training step 40. In contrast, GRPO shows more volatile behavior, particularly evident in the dramatic spike and subsequent drop around training steps 20-25. While GRPO eventually recovers and stabilizes around 97% by the end of training, it consistently underperforms GTPO by approximately 2-3 percentage points in the later stages.

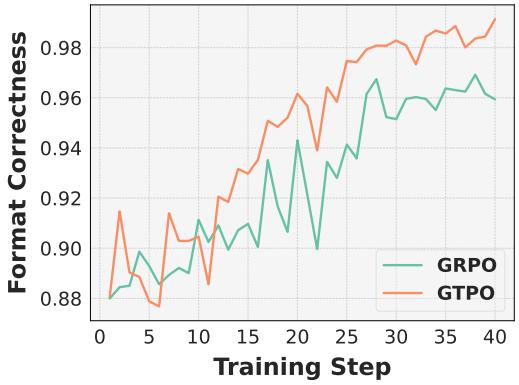

Figure 8: Format correctness curves of GRPO and GTPO: the format correctness metric refers to the percentage of rollout trajectories that do not include any format-based errors (e.g. generated code blocks).

A.9 Additional Results

Scaling with different trajectory turns . We further conducted ablation study by changing the maximum number of turns in generated trajectories. As shown in Table 9, the performance of GTPO improves steadily when increasing the number of maximum turns. This observation shows that GTPO scales well with the trajectory length in multi-turn TIR reasoning.

Max turns $\mathcal{T}$	**MATH 500** (*pass@1*)	**AIME 2024** (*avg@16*)	**Average**
1	71.20	19.38	45.29
2	69.62	20.21	44.92
3	**72.47**	**21.25**	**46.86**

Table 9: **GTPO by different maximum trajectory turns:** We conducted experiments with GTPO where we changed the maximum number of turns in rollout trajectories $\mathcal{T} = \{1, 2, 3\}$ during training.