Robust Instruction Following via Reinforcement Learning

Anonymous ACL submission

Abstract

Instruction following has become a cornerstone capability of Large Language Models (LLMs) since the advent of ChatGPT (Ouyang et al., 2022). Despite its significance, we observe that existing training paradigms often lead to a catastrophic degradation in multi-step reasoning performance. This phenomenon is counter-intuitive, as robust instruction adherence should theoretically augment, rather than hinder, general task performance. To bridge this gap, we propose **Ro**bust **I**nstruction **F**ollowing via **R**einforcement **L**earning (**`RoIFRL`**), a novel RL training framework. `RoIFRL` reframes instruction following as a composite task consisting of *constraint adherence* and *sequential execution*. Furthermore, it introduces a two-dimensional curriculum that dynamically orchestrates data sampling from two subtasks by aligning the data mixture and the instruction complexity with the policy model's evolving capabilities. We synthesize a comprehensive set of verifiable RL training data covering both subtasks. Extensive experiments and ablation studies demonstrate that `RoIFRL` not only preserves but further enhances multi-step reasoning while significantly improving standard instruction-following benchmarks, like IFEval[1].

1 Introduction & Pilot Study

Reinforcement Learning (RL) has emerged as a powerful paradigm for enhancing the instruction-following capabilities of LLMs. Following Pyatkin et al. (2025) and Peng et al. (2025), improving Instruction Following via RL (IFRL) could be formulated as optimizing a policy π_θ to generate a response y that adheres to a specific set of constraints $\mathcal{C}$ while fulfilling a core task x. This optimization problem is modeled as maximizing the expected return of a constraint-aware reward function $R(y, \mathcal{C})$:

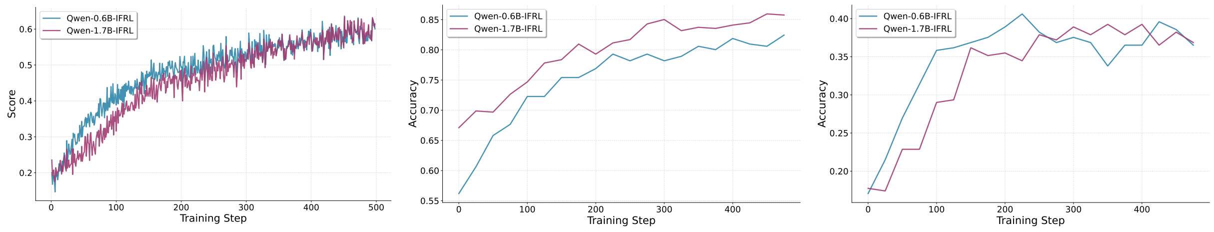
$$\max_{\theta} \mathbb{E}_{(x,\mathcal{C})\sim\mathcal{D}} \left[\mathbb{E}_{y\sim\pi_\theta(\cdot|x,\mathcal{C})}[R(y,\mathcal{C})]\right]$$

Here, $\mathcal{D}$ represents the distribution of instruction-constraint pairs, and $R(y, \mathcal{C})$ could be both deterministic programs or reward models.

Existing IFRL training recipes (Bercovich et al., 2025; Pyatkin et al., 2025; Peng et al., 2025) achieve substantial gains on standard instruction following benchmarks like IFEval (Zhou et al., 2023), IFBench (Pyatkin et al., 2025). However, we identify **a critical limitation of the IFRL-trained models: their multi-step reasoning abilities suffer significant degradation**. We train Qwen3 (Yang et al., 2025) models on the verifiable instruction-following dataset proposed by Pyatkin et al. (2025) using GPRO (Shao et al., 2024) and evaluate these models on standard instruction-following benchmarks, as well as multi-step reasoning benchmarks: GSM8K (Cobbe et al., 2021), MATH500 (Lightman et al., 2023), MMLU-Lite (Singh et al., 2025), BBH (Suzgun et al., 2023) and MBPP+ (Austin et al., 2021).

Figure 1 illustrates the training rewards and instruction following performance of two IFRL trained Qwen3 models. We observe that training remains stable and accuracy on both IFEval and IFBench consistently improves, confirming IFRL as an effective method for instruction following. However, as detailed in Table 1, IFRL trained models exhibit notable performance drops across most multi-step reasoning benchmarks. Furthermore, IFRL training substantially reduces the number of output tokens across all benchmarks; this reduction indicates a degradation in multi-step reasoning ability, as models tend to produce shorter responses that lack necessary intermediate steps. This raises a fundamental question for existing IFRL training methods: *how can we prevent the degradation in multi-step reasoning while improving instruction following?* Given that instruction following is a core capability of LLMs, it's improvement should ideally synergize with, rather than compromise, other essential capabilities.

[1] Our codebase and data will be released.



Critic Score | IFEval | IFBench

Figure 1: Standard Instruction Following Performance

Model	GSM8K	MATH500	MMLU-Lite	BBH	MBPP+	Avg.	Avg. Tokens
Qwen3-0.6B	38.9	36.5	51.6	32.8	34.6	38.9	248
Qwen3-0.6B-IFRL	2.4	4.2	40.8	18.6	18.5	16.9	21
Δ (Base $\rightarrow$ IFRL)	-36.5	-32.3	-10.8	-14.2	-16.1	-22.0	-227
Qwen3-1.7B	46.7	47.4	65.0	42.7	53.7	51.1	399
Qwen3-1.7B-IFRL	19.1	10.6	58.0	33.6	52.3	34.7	50
Δ (Base $\rightarrow$ IFRL)	-27.6	-36.8	-7.0	-9.1	-1.4	-16.4	-349

Table 1: Performance degradation of IFRL-trained models on multi-step reasoning benchmarks.

2 Robust Instruction Following from RL

To address the limitations of standard IFRL in multi-step reasoning, we propose **Ro**bust **I**nstruction **F**ollowing via **R**einforcement **L**earning (**RoIFRL**). Unlike the standard formulation which focuses solely on constraint adherence, RoIFRL further encourages **sequential execution** and treats instruction following as a dual-objective optimization problem. We draw from two distinct data sources: a constraint adherence dataset $\mathcal{D}_{\text{ca}}$ (where instructions x_{ca} contains a set of constraints $\mathcal{C}$ to adhere) and a sequential execution dataset $\mathcal{D}_{\text{se}}$ (where instructions x_{se} consists of an ordered sequence of sub-steps $\mathcal{S} = (s_1, s_2, \ldots, s_k)$). The optimization objective is to maximize the expected return over a mixed distribution, using specialized reward functions for each subtask:

$$\max_{\theta} \Bigg(\underbrace{\mathbb{E}_{(x_{\text{ca}},\mathcal{C})\sim\mathcal{D}_{\text{ca}}} \left[R_{\text{ca}}(y_{\text{ca}}, \mathcal{C})\right]}_{\text{Constraint Adherence}} + \alpha \cdot \underbrace{\mathbb{E}_{(x_{\text{se}},\mathcal{S})\sim\mathcal{D}_{\text{se}}} \left[R_{\text{se}}(y_{\text{se}}, \mathcal{S})\right]}_{\text{Sequential Execution}} \Bigg). \tag{1}$$

$R_{\text{ca}}(y_{\text{ca}}, \mathcal{C})$ is used to assess whether the response satisfies constraints $\mathcal{C}$. $R_{\text{se}}(y_{\text{se}}, \mathcal{S})$ is a reward function that evaluates whether the policy correctly executes the sub-steps $\mathcal{S}$ sequentially. By optimizing this joint objective, models can not only gain a "local focus" of adhering constraints in a single instruction but also a "global view" of correctly executing the given instructions one by one.

Two-Dimensional Curriculum for Balanced Training. Instructions for sequential execution encompass many intermediate sub-steps $\mathcal{S} = (s_1, s_2, ..., s_k)$, making them inherently more extensive than instructions for constraint adherence. Consequently, models are required to generate much longer to fulfill all the substeps. How to balance the optimization of the subtasks in Eq. 1 is the key for effective training. To address this, we introduce a *Two-Dimensional Curriculum* that dynamically orchestrates data sampling from two subtasks by aligning both the inter-source mixture ratio and intra-source instruction complexity with the policy model's evolving capabilities.

Inter-Source Balance via Online Rewards The online training reward serves as an informative signal of the policy's evolving ability. To balance the mixing ratio in a training batch between instruction sources $d \in \{\mathcal{D}_{ca}, \mathcal{D}_{se}\}$, we employ an inverse-reward sampling strategy. We maintain an exponential moving average (EMA_d) of the rewards of model responses for each source. Sampling weights w_d are dynamically adjusted such that sources with lower performance are sampled more frequently:

$$w_d \propto \exp\left(-\text{EMA}_d/\tau\right), \quad \text{s.t.} \quad \frac{\max_d w_d}{\min_d w_d} \leq R,$$

where τ controls the sensitivity to reward differences and R is a clipping ratio to prevent one data source dominates the training batch. This creates a *self-correcting* loop: if the policy struggles with one task, the corresponding sampling weight increases; as proficiency is regained, the curriculum naturally shifts back to the other task.

Intra-Source Balance for Sequential Execution Training on samples with complexity closest to the model's current capability is intuitive and effective (Bae et al., 2025; Shi et al., 2025), where a reliable complexity measure is crucial. To achieve this, we employ LogicIFGen (Zhang et al., 2025) to generate **verifiable** instructions for sequential execution. LogicIFGen transforms a code function and its associated test cases into a multi-step natural language instruction that guides the model through the input processing steps required to obtain the correct output. The complexity of the generated instructions can be naturally measured by Abstract Syntax Tree and are verified to be reliable. We then partition the instructions of sequential execution into complexity intervals $i \in \{0, \ldots, N\}$. Higher complexity intervals are *unlocked* only when the policy's $\text{EMA}_{\mathcal{D}_{se}}$ exceeds a threshold. To accelerate adaptation to newly unlocked difficulties while preventing forgetting of simpler skills, we apply *interval decaying*. Given unlocked intervals $0 \ldots k$, the probability of sampling interval i is:

$$P(i) = \frac{\lambda^{k-i}}{\sum_{j=0}^{k} \lambda^{k-j}},$$

where $\lambda \in (0, 1]$ is a decay factor. This mechanism ensures the model focuses on the frontier of its capabilities (highest weight on interval k) while maintaining previously learned skills.

3 Experiments

Data for Instruction Following 1) For constraint adherence (CA), we incorporate the IFTrain dataset proposed by (Pyatkin et al., 2025), which contains 95k instructions with diverse verifiable constraints. 2) For sequential execution (SE), we collect the solutions from the `code_contests` dataset (Li et al., 2022) as seed functions and feed them to LogicIFGen (Zhang et al., 2025). As a result, we generate 441,564 verifiable SE instructions corresponding to 27,324 functions, with an average length of 1,030.5 words. Following Zhang et al. (2025), we use Abstract Syntax Tree to get the complexity scores of the instructions, whose distribution is shown in Figure 3. Please see Appendix Section A for more details of the training data.

Experimental Setting We use the instruct version of Qwen3-0.6B as the base model for the training. The clipping ratio for the two types of instruction following data R is set to 3 and the temperature τ is set to 0.3 for inter-source balance. For intra-source balance of SE, we divide the instructions into 20 intervals based on their complexity scores and unlock the easiest 4 intervals before training. The unlocking threshold is set to 0.5, which corresponds to a model success rate of roughly 50% and the decay factor λ is set to 0.8. We use a training batch size of 512, a learning rate of 1e-6, and GRPO with a group size of 8. For evaluation, we evaluate the models on the same benchmarks as the preliminary experiments in Section 1 and use the recommended decoding temperature of 0.7 (Yang et al., 2025). More detailed experimental setting could be found at Appendix Section B.

Main Results The performance comparison of the base model and the models trained with IFRL and `RoIFRL` are shown in Table 2. We found that: **1) Redefining instruction following as a composite task consisting of both constraint adherence (CA) and sequential execution (SE) is not only more intuitive but also empirically more effective.** This is clearly demonstrated in the final row of Table 2 ("- Inter-source Balance (Fixed 1:1)"), where we implement a fixed 1:1 training mixture of CA and SE data (no two-dimensional curriculum), the model's multi-step reasoning capability is successfully maintained, with an average of 35.1 compared to the IFRL trained model (16.9), while suffering only a minor reduction in standard instruction-following. **2) Our proposed Two-Dimensional Curriculum effectively optimizes the synergy between two disparate instruction following training data sources.** As shown in Table 2, the final Qwen3-0.6B-`RoIFRL` model achieves the highest performance across both type of benchmarks, reaching an average of 41.5 in multi-step reasoning and 60.3 in instruction following. Notably, `RoIFRL` *not only prevents the degradation seen in standard IFRL but improves the base model's multi-step reasoning from 38.9 to 41.5 while simultaneously achieve better results than IFRL-trained model in instruction following (60.3 vs. 59.5).* It achieves a significant leap in GSM8K performance to 58.9 (compared to the base model's 38.9) and maintains a superior IFBench score of 43.7. These results confirm the effectiveness of the proposed two-dimensional curriculum

Model	Multi-Step Reasoning						Instruction Following		
	GSM8K	MATH500	MMLU-Lite	BBH	MBPP+	Avg.	IFEval	IFBench	Avg.
Qwen3-0.6B	38.9	36.5	**51.6**	**32.8**	**34.6**	38.9	56.1	17.0	36.6
Qwen3-0.6B-IFRL	2.4	4.2	40.8	18.6	18.5	16.9	**82.4**	36.5	59.5
Qwen3-0.6B-RoIFRL	58.9	42.0	46.4	26.7	33.3	**41.5**	76.8	**43.7**	**60.3**
– Interval Decaying	48.9	**44.4**	44.0	23.5	34.4	39.0	80.5	36.5	58.5
– Intra-source Balance for SE	**61.6**	43.8	34.5	23.1	15.6	35.7	75.0	33.1	54.1
– Inter-source Balance (Fixed 1:1)	49.8	35.4	42.1	27.7	20.6	35.1	77.8	35.4	56.6

Table 2: Model Performance on Multi-step Reasoning and Standard Instruction Following Benchmarks

to dynamically orchestrate data sampling based on evolving policy capabilities.

Ablation on the Components of Two-Dimensional Curriculum We remove the components of the two-dimensional curriculum one by one to study their effect on the model training, specifically, the interval decaying for intra-source balance, the whole intra-source balance for SE and the inter-source balance mechanism. Removing all components reverts the training to a "Fixed 1:1" baseline, where the sampling ratio between data sources remains 1:1 throughout training. As shown in Table 2, *removing the intra-source balance for SE results in the most significant performance degradation*. Specifically, the average multi-step reasoning score drops from 39.0 to 35.7, while the average instruction-following decreases from 58.5 to 54.1. This indicates that progressive learning on SE is essential for balancing the two types of model capabilities. Furthermore, we observe that *applying inter-source balance alone yields negligible gains in reasoning and may even deteriorate instruction following*. By analyzing the training ratio of CE samples in Figure 4, we note a precipitous drop at the beginning of training. This is attributed to the model being overwhelmed by high-complexity SE data before mastering basic skills. These findings underscore that inter-source balance is better coupled with intra-source complexity control to maintain training stability when the dataset contains challenging SE instructions.

Comparison with Math RL Data To investigate whether the performance recovery observed in `RoIFRL` is simply a result of introducing additional data requiring multi-step inference or specifically due to the SE data, we compare our approach against a model trained on a 1:1 mixture of CE instructions and pure mathematical reasoning data (Math RL Data). We use DeepScalarR (Luo et al., 2025), which contains 40k unique mathematics problem-answer pairs from various math competitions.

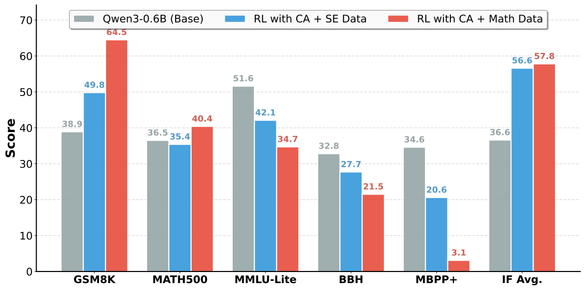


Figure 2: Sequential Execution Data vs Math RL Data

As illustrated in Figure 2, *while incorporating Math RL data provides a significant boost to domain-specific performance, it results in a lack of general robustness compared to our approach*: 1) The Math RL model achieves peak performance on math benchmarks, reaching 64.5 on GSM8K and 40.4 on MATH500. However, this specialized gain comes at the cost of performance degradation in other domains. 2) In contrast, our proposed SE data are far more balanced and generalized. While its scores on GSM8K and MATH500 are lower, it maintains significantly higher performance on MMLU-Lite, BBH, and MBPP compared to CA + Math data and the average multi-step reasoning is better (35.1 vs 32.8). 3) According to Table 2, by further adopting the two-dimensional curriculum, the model trained with CA + SE data could achieve far more better performance than CA + Math data.

4 Conclusion

We propose `RoIFRL`, a novel RL framework that addresses the multi-step reasoning degradation typically observed in instruction-following training via RL. By reframing instruction following as a composite task of constraint adherence and sequential execution, and employing a Two-Dimensional Curriculum to balance disparate data sources during training, RoIFRL effectively preserves and even enhances multi-step reasoning ability while improving the standard instruction following.

5 Limitation

Despite the effectiveness of `RoIFRL`, our work has several limitations that provide directions for future research.

- Due to the substantial computational resources required for the long generations characteristic of sequential execution, we primarily validated our framework on the Qwen-0.6B model. While the results are promising, the scalability of RoIFRL to larger models remains to be fully explored. Furthermore, given our hardware constraints, we utilized intuitive or standard hyperparameter configurations rather than conducting an exhaustive search, which may leave room for further performance optimization.

- Our current intra-source curriculum is exclusively applied to sequential execution data. This decision was driven by the availability of reliable and accurate complexity metrics for SE instructions—specifically those derived from Abstract Syntax Trees via LogicIFGen. In contrast, measuring the difficulty of instructions for constraint adherence remains an open challenge. Our findings suggest that the Two-Dimensional Curriculum can effectively maintain training stability and balance data sampling through the SE curriculum. Consequently, extending intra-source curriculum learning to constraint adherence is deferred to future work.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program synthesis with large language models. *arXiv [cs.PL].*

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, Jeongyeon Nam, and Donghyun Kwak. 2025. Online difficulty filtering for reasoning oriented reinforcement learning. *arXiv [cs.CL].*

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, Ido Shahaf, Oren Tropp, Ehud Karpas, Ran Zilberstein, Jiaqi Zeng, Soumye Singhal, Alexander Bukharin, Yian Zhang, Tugrul Konuk, and 117 others. 2025. Llama-nemotron: Efficient reasoning models. *arXiv [cs.CL].*

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv [cs.LG].*

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, and 7 others. 2022. Competition-level code generation with AlphaCode. *arXiv [cs.PL].*

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 [cs.LG].*

Michael Luo, Sijun Tan, Justin Wong, Xiaoxiang Shi, William Tang, Manan Roongta, Colin Cai, Jeffrey Luo, Tianjun Zhang, Erran Li, Raluca Ada Popa, and Ion Stoica. 2025. Deepscaler: Surpassing o1-preview with a 1.5b model by scaling rl.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E Miller, Maddie Simens, Amanda Askell, P Welinder, P Christiano, J Leike, and Ryan J Lowe. 2022. Training language models to follow instructions with human feedback. *Adv. Neural Inf. Process. Syst.*, abs/2203.02155.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2025. VerIF: Verification engineering for reinforcement learning in instruction following. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 30312–30327, Stroudsburg, PA, USA. Association for Computational Linguistics.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. 2025. Generalizing verifiable instruction following. *arXiv [cs.CL].*

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y K Li, Y Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv [cs.CL].*

Taiwei Shi, Yiyang Wu, Linxin Song, Tianyi Zhou, and Jieyu Zhao. 2025. Efficient reinforcement finetuning via adaptive curriculum learning. *arXiv [cs.LG].*

Shivalika Singh, Angelika Romanou, Clémentine Fourrier, David Ifeoluwa Adelani, Jian Gang Ngui, Daniel Vila-Suero, Peerat Limkonchotiwat, Kelly Marchisio, Wei Qi Leong, Yosephine Susanto, Raymond Ng, Shayne Longpre, Sebastian Ruder, Wei-Yin

Ko, Antoine Bosselut, Alice Oh, Andre Martins, Leshem Choshen, Daphne Ippolito, and 4 others. 2025. Global MMLU: Understanding and addressing cultural and linguistic biases in multilingual evaluation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 18761–18799, Stroudsburg, PA, USA. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, Stroudsburg, PA, USA. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *arXiv [cs.CL]*.

Mian Zhang, Shujian Liu, Sixun Dong, Ming Yin, Yebowen Hu, Xun Wang, Steven Ma, Song Wang, Sathish Reddy Indurthi, Haoyun Deng, Zhiyu Zoey Chen, and Kaiqiang Song. 2025. Complex logical instruction generation. *arXiv [cs.CL]*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv [cs.CL]*.

A Training Data

The `code_contests` dataset (Li et al., 2022) is a large-scale, high-quality benchmark for competition-level code generation, originally introduced by Google DeepMind. It aggregates competitive programming problems from multiple platforms, including Codeforces, Description, and Aizu Online Judge. Each problem instance contains a natural language description, multiple correct and incorrect solutions in various programming languages (e.g., C++, Python, Java), and a comprehensive set of test cases.

From this dataset, we extract problem descriptions together with the corresponding Python solutions and test cases. In addition, we prompt GPT-5 to generate alternative solutions and additional test cases for each problem to further increase diversity. These solution functions and test cases are then passed into LogicIFGen (Zhang et al., 2025) to produce sequential execution instructions in natural language. We apply the same data-cleaning pipeline as described in Zhang et al. (2025), filtering out test cases involving excessively large numerical values or high-precision outputs, which are unsuitable for sequential execution. We follow Zhang et al. (2025) to use Abstract Syntax Tree to get the *Cyclomatic Complexity* (C), *Nesting depth* (D), the number of function invocation (F), and the *Function length* (L) and calculate the complexity score as folllows:

$$\text{Score} = D \times 3 + F \times 2 + C \times 1 + L \times 0.5.$$

Finally, we obtain 441,564 verifiable multi-step instructions corresponding to 27,324 functions covering a wide range of logical complexity as shown in Figure 3. We randomly sample 80k instructions for model training. An example of the sequential execution instruction and its corresponding source function is as follows:

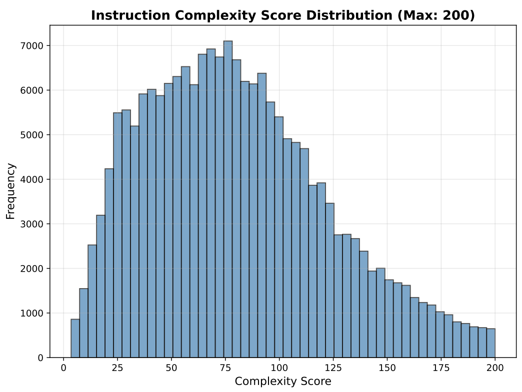


Figure 3: Sequential Execution Data Complexity

Sequential Execution Instruction

INPUTS:
You're working with three separate integer values called a, b, and c. Each one is just a whole number on its own—there's no extra structure or list, just these three numbers.
LOGICS:
Okay, let's walk through it step by step. First, what you're going to do is create a counter named k and set it to 0. That's the only counter you'll need. Next, you need to figure out the total. So take the value in a, add to it the value in b, and then add the value in c. Write that sum down and call it total. At this point, you want to pick out the single biggest number among a, b, and c. Here's how: compare a and b—if a is larger, keep a in mind; if b is larger, keep b in mind. Then take whichever one you kept and compare it to c—whichever is larger there is your largest. Write that value down and call it largest. Now you're going to compare that largest to the sum of the other two. To get that sum, take total and subtract largest; call the result remainder. Here comes the decision point: check if largest is strictly greater than remainder. If largest > remainder, then you go down the first path: Add 1 to your counter k (so it goes from 0 to 1). Set a new value called original_result equal to remainder. You're done—skip the next step and move straight to the finish. If largest is not greater than remainder (meaning largest <= remainder), you go down the second path: Calculate original_result by dividing total by 2 and throwing away any fraction (just the integer part). Leave k at whatever it already is (which will still be 0). Then you're done with the steps. Either way, you finish with two things: original_result and your counter k.
OUTPUTS:
When you finish, you'll have: 1) original_result — a single integer you computed by one of those two paths; and 2) a small statistics map with one entry: 'k'. Its value is the final count you ended up with (either 0 or 1).
Input Values: [2, 3, 5]

Function:

```python
def f(a, b, c):
    k = 0
    total = a + b + c
    largest = max(a, b, c)
    if largest > total - largest:
        k += 1
        original_result = total - largest
        return original_result, {'k': k}
    original_result = total // 2
    return original_result, {'k': k}
```

B Experimental Settings

The detailed training and evaluation parameters are shown in Table 3. All the experiments are conducted on two NVIDIA A100 GPUs.

We evaluate our models on five diverse benchmarks spanning mathematical reasoning, code generation, and general knowledge: GSM8K (Cobbe et al., 2021) (1,319 grade-school math problems with chain-of-thought prompting), MATH500 (Lightman et al., 2023) (500 competition-level mathematics problems), MBPP+ (Austin et al., 2021) (378 Python programming tasks), MMLU-lite (Singh et al., 2025) (615 multiple-choice questions across various domains in English), and BBH (Suzgun et al., 2023) (Big-Bench Hard, comprising 6,511 samples across 23 challenging reasoning tasks). All evaluations are conducted using the NeMo Evaluator framework with models served via

Parameter	Value
Model and Data Configuration	
Base Model	Qwen3-0.6B (Instruct)
EMA Momentum	0.2
Max Mixture Ratio (R)	3
Sampling Temperature (τ)	0.3
Reward Threshold	0.5
Decay Factor (λ)	0.8
Number of Complexity Intervals	40
Initial Data Source Ratio	1:1
Training	
Temperature	1e-6
Global Batch Size	512
Max Steps	500
GRPO Group Size	8
KL Loss Coefficient	0.001
Max Completion Tokens	4096
Evaluation	
Decoding Temperature	0.7
Top-p	0.8
Top-k	20
Max Completion Tokens	8192

Table 3: Experimental Settings for `RoIFRL`

vLLM. We use consistent generation parameters across all benchmarks. Each sample is evaluated with a single generation.

C Ablation Experiments

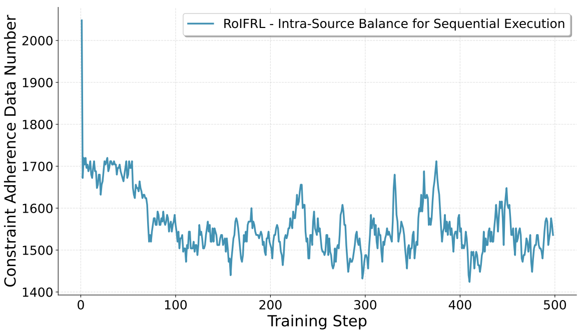


Figure 4: Batch Ratio of Constraint Adherence Data