VERIF: Verification Engineering for Reinforcement Learning in Instruction Following

Anonymous ACL submission

Abstract

Reinforcement learning with verifiable rewards (RLVR) has become a key technique for enhancing large language models (LLMs), with verification engineering playing a central role. However, best practices for RL in instruction following remain underexplored. In this work, we explore the verification challenge in RL for instruction following and propose VERIF, a verification method that combines rule-based code verification with LLM-based verification from a large reasoning model (e.g., QwQ-32B). To support this approach, we construct a high-quality instruction-following dataset, VERINSTRUCT, containing approximately 22,000 instances with associated verification signals. We apply RL training with VERIF to two models, achieving significant improvements across several representative instruction-following benchmarks. The trained models reach state-of-the-art performance among models of comparable size and generalize well to unseen constraints. We further observe that their general capabilities remain unaffected, suggesting that RL with VERIF can be integrated into existing RL recipes to enhance overall model performance. We will release our datasets, codes, and models to facilitate future research.

1 Introduction

Reinforcement learning with verifiable rewards (RLVR) has emerged as a key technique for enhancing large language models (LLMs), leading to various advanced LLMs, such as DeepSeek R1 (Guo et al., 2025). The core component of RLVR is verification engineering. Recently, numerous works have explored reliable verification across diverse domains, such as math (Lambert et al., 2024; Guo et al., 2025; Luo et al., 2025b), code (Wang et al., 2024b; Luo et al., 2025a), logic (Xie et al., 2025), medicine (Chen et al., 2024; Wang et al., 2025), and finance (Qian et al., 2025b; Liu et al., 2025a).

In this work, we explore verification engineering for reinforcement learning in instruction following.

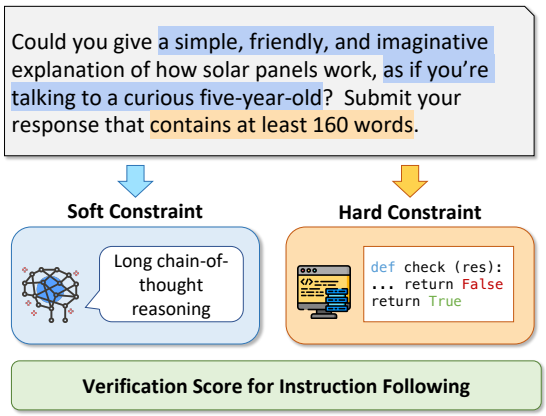

Figure 1: A simplified illustration of VERIF. The instruction constraints are categorized as soft or hard and verified using different methods in VERIF.

Specifically, this work focuses on the following of constraints in the instruction (Zhou et al., 2023), such as response length, as shown in Figure 1. The constraints are usually divided into two types: *hard* constraints, which can be verified using simple rules, e.g., length, and *soft* constraints, which require semantic judgment, e.g., style. Assessing whether a response satisfies these constraints provides a natural basis for verification in RLVR. However, reinforcement learning for instruction following remains underexplored. The only notable work, TULU 3 (Lambert et al., 2024), applies RLVR to enhance instruction following. However, the improvement is limited, and it focuses solely on hard constraints, neglecting soft constraints. Therefore, the best practice of verification engineering for RL in instruction following remains under-explored.

Given the above issues, we explore the best practice of RLVR in instruction following and propose VERIF, a verification method for instruction following that combines rule-based code verification with verification from a large reasoning model. As shown in Figure 1, hard constraints are verified through code, and soft constraints are handled by

a large reasoning model, which enables effective verification through long chain-of-thought reasoning (Liu et al., 2025b). VERIF requires no manual annotations or reference answers, offering an efficient solution for automatic verification. To support this approach, we construct a high-quality dataset, VERINSTRUCT, containing approximately $22,000$ instances with verification. The data construction involves two main steps: (1) instruction construction with multiple constraints, where we apply constraint back-translation (Qi et al., 2024) to augment existing instructions with additional constraints; (2) verification generation. For hard constraints such as length, we use Qwen2.5-72B-Instruct (Yang et al., 2024) to generate verification code. For soft constraints, they are verified online during RL training using large reasoning models.

We apply reinforcement learning with VERIF on two SFT-trained models using VERINSTRUCT, including TULU 3 SFT (Lambert et al., 2024) and DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025). Specifically, VERIF computes the final reward as the average of hard constraint scores (0 or 1) from code validation and soft constraint scores (0 or 1) determined by the QwQ-32B (Qwen, 2025). We train the models using the GRPO algorithm (Shao et al., 2024). We evaluate the trained models on several widely-used instruction-following benchmarks, including IFEval (Zhou et al., 2023), Multi-IF (He et al., 2024b), SysBench (Qin et al., 2024), FollowBench (Jiang et al., 2024), and CFBench (Zhang et al., 2024a). Experimental results show that the RLVR-trained models using VERIF achieve significant improvements. Notably, the model trained based on TULU 3 SFT achieves state-of-the-art performance among models of similar parameter scale and outperforms TULU 3 (Lambert et al., 2024), which is trained with extensive DPO data and rule-based RLVR. The results demonstrate the effectiveness of our verification method VERIF.

We conduct further analytical experiments. We first evaluate the generalization of the trained models on general instruction following tasks, including AlpacaEval 2.0 (Dubois et al., 2024) and MT-Bench (Zheng et al., 2023), and mathematical reasoning tasks, including GSM8K (Cobbe et al., 2021) and Omni-MATH (Gao et al., 2025), natural language understanding datasets: MMLU-Pro (Wang et al.) and DROP (Dua et al., 2019), and a natural language inference benchmark BBH (Suzgun et al., 2023). We observe that RL with VERIF preserves general and mathematical capabilities, indicating its potential as an additional RL stage to enhance instruction following without affecting other skills. We analyze the performance gains of trained models across different constraint types and find that RL with VERIF exhibits good generalization to unseen constraints. We also conduct ablation studies on the verification method, using only code validation or only LLM verification, both of which lead to notable performance drops. Finally, we develop a smaller and efficient 7B LLM as the *soft* constraint verifier. Specifically, we extract approximately 130k complex instructions from WildChat (Zhao et al., 2024) and Infinity Instruct (BAAI, 2024), collect responses from 6 different LLMs, and use QwQ to generate constraint verification. We then train DeepSeek-R1-Distill-Qwen-7B on this dataset as a generative verifier for *soft* constraints, achieving RL performance comparable to the model trained using QwQ-32B as the verifier.

2 Pilot Experiments

This section explores the potential of RL for instruction following (§ 2.1) and preliminarily explores different verification methods (§ 2.2) using the reward benchmark IFBench (Peng et al., 2025).

2.1 Potential for RL Training

We first explore the potential of RL in instruction following, as most previous works have adopted supervised fine-tuning (SFT; Ouyang et al., 2022) or direct preference optimization (DPO; Rafailov et al., 2023), with limited use of RL. This raises a key question: *Does RL hold untapped potential for instruction following*? To explore this question, we evaluate the pass@k performance of several LLMs on the instruction following benchmark IFEval (Zhou et al., 2023). The motivation is that RL enhances performance by increasing the likelihood of sampling correct responses, and a high pass@k at large k suggests untapped potential that RL can exploit (Yue et al., 2025a). The experimental results of TULU 3 SFT and DeepSeek-R1-Distill-Qwen-7B are shown in Figure 2. We can observe that the results are much higher with larger k, with pass@64 showing over a 20% increase compared to pass@1. This suggests that LLMs can sample correct answers on IFEval at higher k, with the potential that can be exploited during RL training.

2.2 Verification Engineering

We preliminarily conduct verification engineering using reward model benchmarks. Specifically,

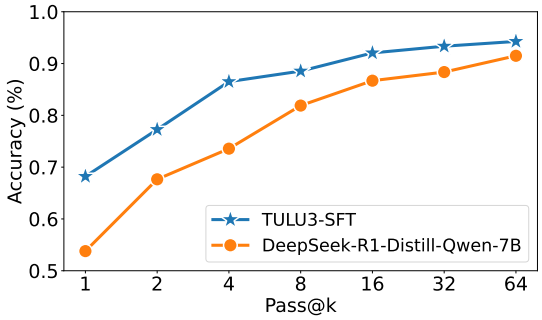

Figure 2: Pass@k results (%) of two SFT-trained LLMs on IFEval. We report the prompt-level strict score.

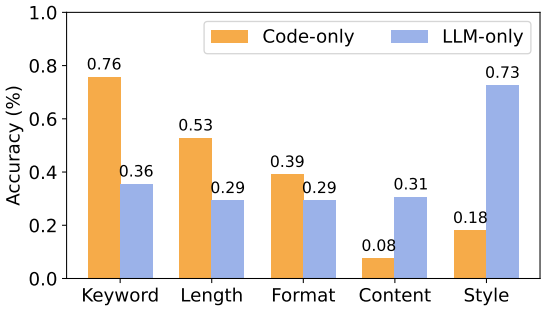

Figure 3: Accuracy (%) of code-only or LLM-only verification in verifying compliance with different types of constraints. LLM-only adopts QwQ-32B.

Method	Hard	Soft	Overall
Code-only	60.6	13.2	48.6
LLM-only$_{\text{QwQ}}$	31.5	48.1	37.4
LLM-only$_{\text{QWEN}}$	19.7	45.3	28.6
Code+LLM$_{\text{QwQ}}$	61.3	48.1	58.1

Table 1: Accuracy (%) of three verification methods on IFBench. "Hard" or "Soft" indicates that the rejected response only violates certain hard or soft constraints.

we evaluate different verification methods on IFBench (Peng et al., 2025), a benchmark designed for instruction-following rewards that consists of an instruction and two responses, where the task is to select the response that better follows the instruction. IFBench includes 3 common hard constraints: length, format, and keyword, and 2 common soft constraints: style and content. We explore three verification methods: (1) code-only verification, similar to RewardAgent proposed by Peng et al. (2025), which uses automatically generated code for each constraint verification; (2) LLM-only verification, which directly uses the LLM as the judge; (3) code+LLM verification, which applies code verification for hard constraints and LLM for soft constraints. We explore using QwQ-32B (Qwen, 2025) and Qwen2.5-72B-Instruct (Yang et al., 2024) as the LLMs. The results are shown in Table 1. We can observe that code+LLM verification performs much better and reasoning LLMs (QwQ) also perform better than non-reasoning LLMs (Qwen).

We further investigate the accuracies of code and LLM verification on different types of constraints, and report their respective accuracy for soft and hard constraints in Table 1, which further confirms that code verification is more effective for hard constraints and LLM verification performs better on soft constraints, supporting the rationale for the code+LLM verification approach. The detailed results across different constraint types are shown in Figure 3, and we can observe that LLMs perform particularly poorly on keyword and length constraints, which may be due to inherent limitations in numerical counting (Fu et al., 2024; Ball et al., 2024). Since keyword and length constraints can be efficiently verified with code, we conclude that in instruction-following verification, hard constraints should be checked with code, and soft constraints can be reliably verified by advanced LLMs.

3 Method

This section introduces the formalization of VERIF (§ 3.1), the construction process of VERINSTRUCT (§ 3.2), and the RL training method (§ 3.3).

3.1 Verification Method

Suppose we are given an instruction x, which includes the task decription and a set of constraints $C = \{c_1, c_2, ..., c_n\}$. We follow the task definition of instruction-following by Zhou et al. (2023): given x, generating a response y that satisfies all constraints in C. In this work, our primary goal is to accurately verify whether y meets all constraints and to apply this reliable verification in reinforcement learning training. Specifically, the constraint set C consists of two types: hard constraints C_h, which can be verified by simple rules or code (e.g., length), and soft constraints C_s, which require semantic understanding (e.g., style). As explored in § 2.2, we propose a hybrid verification approach, VERIF, that uses code verification for C_h and LLM verification for C_s. Formally, this is defined as:

$$\text{VERIF}(x, y) = F(\text{Code}(x, C_h), \text{LLM}(x, C_s))$$

$\text{Code}(x, C_h) \in \{0, 1\}$ denotes whether y satisfies all hard constraints in C_h, and $\text{LLM}(x, C_s) \in \{0, 1\}$ indicates whether y satisfies all soft constraints in C_s. F denotes the aggregation method

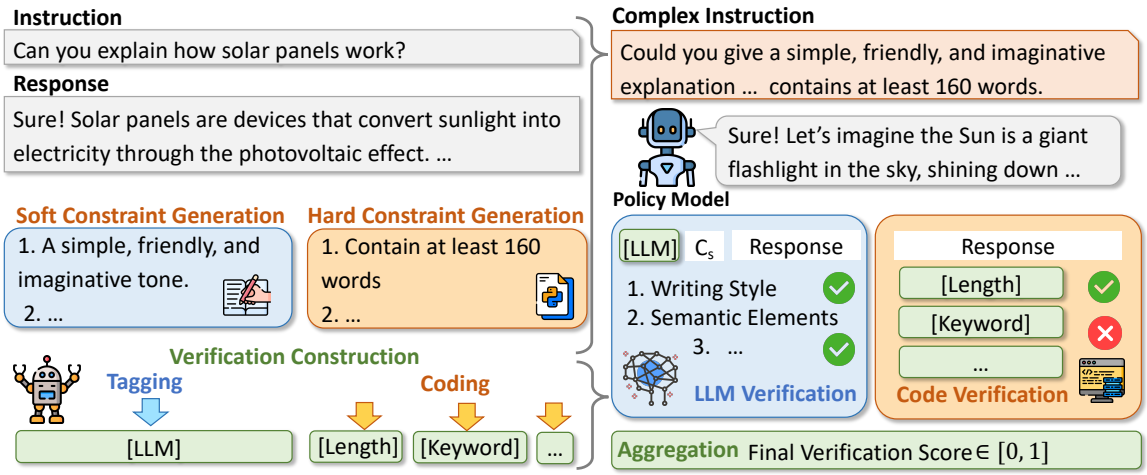

Figure 4: **Left:** The construction process for VERINSTRUCT, including complex instruction generation and verification construction. **Right:** Our verification method, VERIF, providing verification for instruction following.

used to combine the code verification score and the LLM verification score, including averaging or multiplication. In this work, we consider only three types of hard constraints, including length, format, and keyword. All other constraints are taken as soft and verified using LLMs. As explored in § 2.2, we use large reasoning models for LLM-based verification, which is a form of scaling up verification and has been demonstrated effective in practice (Liu et al., 2025b; ByteDance-Seed, 2025).

3.2 Data Construction Method

We construct a high-quality instruction-following dataset for reinforcement learning, where each instance is paired with a corresponding verification. Prior works on enhancing instruction-following of LLMs (Sun et al., 2024; Dong et al., 2024; Qi et al., 2024) have primarily focused on generating complex instructions and corresponding high-quality responses for supervised fine-tuning (SFT). In this work, we focus on generating complex instructions with associated verification, eliminating the efforts to generate and filter high-quality responses. As shown in Figure 4, the construction process consists of two main parts: (1) **Complex instruction generation.** We adopt the constraint back-translation approach (Qi et al., 2024) to generate complex instructions, which produces few unrealistic cases. Specifically, we randomly sample 25,000 data instances from four high-quality datasets, including Alpaca GPT4 (Peng et al., 2023), Orca Chat (Es, 2023), Evol Instruct (Xu et al., 2023), and OpenAssitant (Köpf et al., 2024). We use Llama3.1-70B-Instruct (Grattafiori et al., 2024) to generate constraints implicitly satisfied by each response, such as language style. Since LLMs often struggle with understanding length constraints (Sun et al., 2024), we instead automatically synthesize them based on response length using Python scripts. We combine the generated constraints with the original instruction to form the final complex instruction. (2) **Verification construction.** We then automatically generate a verification method for each constraint. For hard constraints, including length, format, and keyword presence, we use Qwen2.5-72B-Instruct to generate verification Python code. Given the simplicity of these generated Python code scripts, we manually check them and find nearly no errors. For soft constraints, we do not generate code but instead tag them with "LLM", which indicates that verification during RL training should be online produced by an LLM. We finally filter out instructions with fewer than 2 constraints, resulting in VERINSTRUCT, which contains 22,000 instructions, each including an average of 6.2 constraints and corresponding verification methods. The details of VERINSTRUCT are placed in Appendix A.

3.3 RL Training

We conduct reinforcement learning using VERIF on VERINSTRUCT. Specifically, we adopt the GRPO algorithm (Shao et al., 2024) and perform 16 rollouts per prompt for value estimation. For each response, the reward is provided online by VERIF. To reduce the overhead of LLM-based verification, we input all soft constraints C_s to the LLM at once to assess whether the response satisfies all of them in a single pass. We conduct RL training using the

Model	IFEval				Multi-IF			SysBench	FollowBench	CFBench
	Pr. (S)	Pr. (L)	Ins. (S)	Ins. (L)	Turn 1	Turn 2	Turn 3	ISR	SSR	ISR
GPT-4o	79.9	84.8	85.6	89.6	82.3	71.7	59.3	80.2	75.3	80.0
QwQ-32B	82.8	86.1	88.0	90.4	64.2	56.6	48.4	67.8	73.5	80.0
Qwen2.5-7B-Instruct	71.5	74.1	79.4	81.3	75.3	57.9	47.0	–	65.9	74.0
LLaMA3.1-8B-Instruct	72.6	77.3	80.8	84.2	71.3	62.8	54.6	–	65.9	71.0
TULU 3	79.7	82.8	85.1	87.5	82.1	63.2	51.2	48.9	70.3	72.0
Crab-7B-DPO	47.3	57.1	59.7	67.9	47.2	36.5	28.9	–	56.3	62.0
Conifer-7B-DPO	48.1	52.3	59.1	63.3	50.7	37.6	26.6	–	56.9	62.0
UltraIF-8B-DPO†	71.3	75.4	79.4	83.1	69.6	58.3	46.9	–	62.6	–
R1-Distill-Qwen-7B	59.9	65.1	70.4	74.2	55.8	43.6	32.7	16.9	53.9	66.0
+VERIF	75.6	79.5	82.7	85.5	66.0	53.8	41.9	26.5	61.0	68.0
TULU 3 SFT	68.4	71.7	76.3	79.5	67.3	50.9	40.3	33.2	62.0	63.0
+VERIF	84.5	87.1	89.3	91.4	79.4	65.2	54.0	54.7	68.6	72.0

Table 2: Experimental results (%) on several representative instruction-following benchmarks. "Pr." and "Ins." denote prompt-level and instruction-level metrics respectively. "S" and "L" mean strict and loose respectively. † denotes the results are sourced from the original paper (An et al., 2025). All the other results are reproduced by us in this paper. For reasoning LLMs, we remove the thinking tokens and evaluate using only the final response.

VeRL framework[1] and integrate a parallel reward computation mechanism to accelerate RL training.

4 Experiments

This section introduces experimental setup (§ 4.1), main results (§ 4.2), analytical experiments (§§ 4.3 to 4.5), and developing a smaller verifier (§ 4.6).

4.1 Experimental Setup

Reported Models We conduct RL training based on two SFT-trained models: TULU 3 SFT (Lambert et al., 2024) and DeepSeek-R1-Distill-Qwen-7B (Guo et al., 2025). For the specific implementation of VERIF, we use QwQ-32B as the LLM verifier and set F in Equation 3.1 as average. For comparison, we evaluate TULU 3 (Lambert et al., 2024), which is trained directly based on TULU 3 SFT with extensive DPO and RLVR training. We also evaluate various industrial models, including GPT-4o (Hurst et al., 2024), QwQ-32B (Qwen, 2025), Qwen2.5-7B-Instruct (Yang et al., 2024), LLaMA3.1-8B-Instruct (Grattafiori et al., 2024), and open-source models specifically optimized for instruction following, including Conifer (Sun et al., 2024), Crab (Qi et al., 2024), and UltraIF (An et al., 2025). More details are placed in appendix B.

Evaluation benchmarks We evaluate the models on several representative instruction-following benchmarks, including IFEval (Zhou et al., 2023), the most commonly used dataset; Multi-IF (He et al., 2024b), which includes multi-turn and multilingual instruction following; SysBench (Qin et al., 2024), which evaluates instruction following to system prompts; FollowBench (Jiang et al., 2024) and CFBench (Zhang et al., 2024a), which cover a comprehensive range of constraint types.

[1] https://github.com/volcengine/verl

4.2 Main Results

All experimental results are presented in Table 2. We have the following observations: (1) Reinforcement learning with VERIF demonstrates strong performance. Compared to their corresponding backbones (R1-Distill-Qwen-7B and TULU 3 SFT), the trained models using RL perform much better. Notably, the model trained based on TULU 3 SFT even outperforms the original TULU 3 (Lambert et al., 2024), which is trained based on TULU 3 SFT using approximately 271k DPO pairs and specialized RLVR data. Among models with similar parameter scales, the model trained based on TULU 3 SFT achieves state-of-the-art performance and surpasses several open-source models developed by industry using larger datasets and more resources. This demonstrates the potential of RL training for instruction following and the effectiveness of VERIF in providing reliable rewards. (2) RL with VERIF generalizes effectively to unseen instruction-following tasks. Although the training dataset VERINSTRUCT contains only English and single-turn instruction-following data, the trained model shows substantial improvements on multilingual, multi-turn (Multi-IF) instruction following, and following system prompts (SysBench). This suggests that the patterns of instruction following may be inherently generalizable and that RL further enhances this generalization (Chu et al., 2025). (3) RL with VERIF benefits both reasoning and non-

Model	AlpacaEval 2.0	MT-Bench	GSM8K	Omni-MATH	MMLU-Pro	BBH	DROP
Qwen2.5-7B-Instruct	37.5	7.8	91.4	13.6	56.5	71.8	77.2
Llama3.1-8B-Instruct	29.4	6.0	83.6	10.8	48.1	63.0	74.4
TULU 3	39.9	7.5	88.4	14.2	35.9	68.5	69.4
R1-Distill-Qwen-7B	16.6	5.7	87.0	35.0	54.3	21.5	74.0
+VERIF	15.5	5.9	90.0	33.6	54.8	32.2	75.6
TULU 3 SFT	7.9	6.3	78.8	11.4	36.4	67.4	58.3
+VERIF	22.0	7.0	83.4	12.4	36.0	67.9	59.5

Table 3: Experimental results (%) on various general natural language benchmarks.

reasoning models. As reinforcement learning has demonstrated its effectiveness in enhancing reasoning abilities on challenging tasks (Guo et al., 2025; Luo et al., 2025a), such as math and code, we suggest integrating instruction-following training into RL pipelines. Our further analysis (§ 4.3) shows that general capabilities, such as mathematical reasoning and language understanding, do not degrade after RL with VERIF and may even slightly improve, indicating that RL with VERIF can be integrated into broader model development for enhancing the model's instruction following capabilities. (4) Models developed by the academic community, such as Conifer, Crab, and UltraIF, show relatively lower performance, which is reasonable given their focus on exploring effective SFT data synthesis and limited training resources. Given that there is abundant open-source SFT data, such as Infinity Instruct (BAAI, 2024) with approximately 7 million instances, we encourage the research community to devote more attention to constructing RL data instead, as RL data remains scarce and RL has been demonstrated to be effective for instruction following. In conclusion, RL with VERIF effectively enhances instruction-following capabilities, and we encourage more efforts on developing effective RL methods or data for instruction-following.

4.3 Analysis on General Capabilities

We further investigate the general capabilities of the trained models to assess the broader impact of RL with VERIF. Specifically, we conduct an evaluation on various representative general benchmarks, including general instruction-following datasets that focus on task completion and are evaluated using LLM-as-a-judge: AlpacaEval 2.0 (Dubois et al., 2024) and MT-Bench (Zheng et al., 2023), mathematical reasoning benchmarks: GSM8K (Cobbe et al., 2021) and Omni-Math (Gao et al., 2025), natural language understanding datasets: MMLU-Pro (Wang et al.) and DROP (Dua et al., 2019), and a natural language inference benchmark BBH (Suzgun et al., 2023). The results are shown in Table 3. We can observe that RL training does not degrade general performance and even improves the performance in some cases, such as MT-Bench, GSM8K, and BBH. We attribute this to a key difference between RL and SFT: while SFT learns and memorizes patterns from data and is prone to catastrophic forgetting (Chu et al., 2025), RL typically maximizes optimal patterns it has learned (Yue et al., 2025b), thereby reducing the risk of knowledge forgetting. These results suggest a promising finding that instruction-following reinforcement learning can be integrated into existing RL pipelines to enhance adherence to instructions without compromising the model's general capabilities.

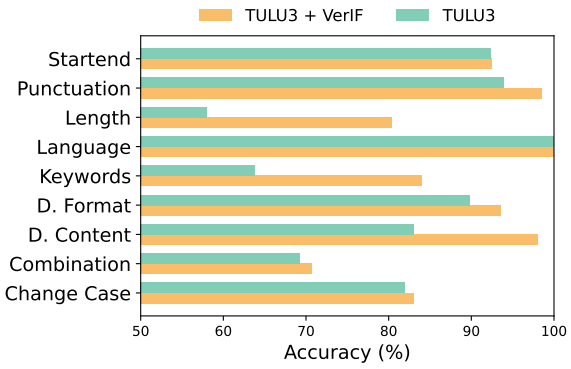

Figure 5: Prompt-level strict scores (%) across different types of constraints on IFEval. "D." denotes Detectable.

4.4 Analysis on Constraint Types

VERINSTRUCT includes only five constraint types: length, keyword, format, content, and style. We further investigate the improvements across different constraint types in IFEval to analyze the generalization of constraint adherence. Results are shown in Figure 5. We observe clear improvements across most types except for "Startend" and "Language" (which already achieves 100% accuracy). This indicates that RL training can generalize instruction-

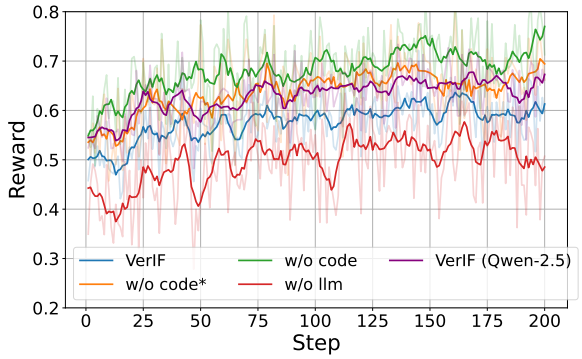

Figure 6: Reward curves during RL training with different verification methods. We visualize the first 200 steps and smooth the data for better visualization.

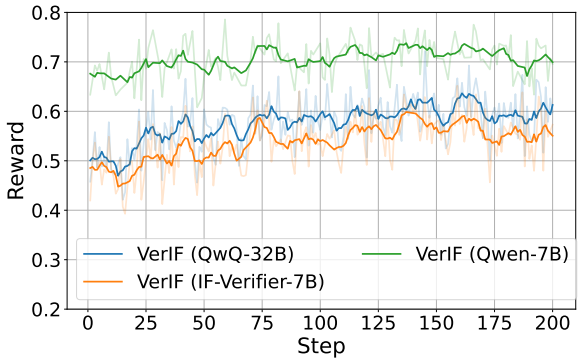

Figure 7: Reward curves during RL training using different LLM verifiers in VERIF. Qwen-7B is short for DeepSeek-R1-Distilled-Qwen-7B.

Model	IFEval	Multi-IF	CFBench
VERIF	84.5	54.0	72.0
w/o code*	81.7	51.2	70.0
w/o code	76.2	52.0	73.0
w/o LLM	74.7	46.0	59.0
VERIF (Qwen-2.5)	76.9	48.5	71.0

Table 4: Ablation results (%) for different verification methods. "Qwen-2.5" uses Qwen2.5-72B-Instruct as the LLM instead of QwQ-32B. We report the prompt-level strict score for IFEval, the Turn 3 score for Multi-IF, and the ISR score for CFBench.

following ability to unseen constraint types. For constraint types covered in VERINSTRUCT, such as length, keyword, and content, the improvements are more pronounced, which demonstrates the precision of the verification provided by VERIF. This also suggests that incorporating datasets with richer constraint types can further improve performance. We encourage the community to explore more diverse data for RL for instruction following.

4.5 Ablation Studies

We conduct ablation studies on the verification method. Specifically, we perform three ablations: (1) "w/o code*", which uses only the LLM to verify all constraints; (2) "w/o code", which uses only the LLM for soft constraints; (3) "w/o LLM", which verifies only hard constraints using Python code scripts. We conduct RL training using different verification methods based on TULU 3 SFT. The reward curves during training are shown in Figure 6. We observe that using only code verification yields lower rewards and limited growth, likely due to the difficulty of following hard constraints. In contrast, using only LLM verification results in higher and more pronounced reward growth, possibly because the LLM verifier is easier to fit or hack (Li et al., 2024). The results are shown in Table 4. We can observe that removing any verification component degrades model performance compared to VERIF. Notably, "w/o LLM", which uses only Python scripts for hard constraint verification, performs significantly poorly. This may be due to that approximately 77.7% constraints in training data are soft. This suggests that using code verification for hard constraints only, as adopted in training TULU 3 (Lambert et al., 2024), is sub-optimal for RL in instruction following. We also adopt Qwen2.5-72B-Instruct as the LLM verifier in VERIF and find it significantly underperforms QwQ-32B. The potential reason may be that in our implementation of VERIF, the LLM is required to verify whether a response satisfies all soft constraints in a single pass, which requires step-by-step reasoning and poses significant challenges. The results demonstrate the potential of scaling up verification (Liu et al., 2025b). In conclusion, we suggest that the best practice for verification in RL for instruction following is VERIF with reasoning models as the LLM verifier. We further explore a smaller reasoning LLM as the verifier in § 4.6.

4.6 Training a Smaller Verifier

Although we have demonstrated VERIF with a large reasoning model, such as QwQ-32B, is effective for RL in instruction following, the long outputs of QwQ-32B lead to high latency during online reward computation. For example, when training TULU 3 SFT, we adopt 8 H800 GPUs for deploying QwQ-32B and set batch size to 32, rollouts to 16, and the average time to obtain the reward for a batch reaches about 180 seconds, accounting for roughly 80% of the time per training step. To

Model	IFEval				Multi-IF			SysBench	FollowBench	CFBench
	Pr. (S)	Pr. (L)	Ins. (S)	Ins. (L)	Turn 1	Turn 2	Turn 3	ISR	SSR	ISR
VERIF (QwQ-32B)	84.5	87.1	89.3	91.4	79.4	65.2	54.0	54.7	68.6	72.0
VERIF (Qwen-7B)	77.1	80.4	84.3	86.6	78.5	60.8	49.0	42.7	62.0	72.0
VERIF (IF-Verifier-7B)	80.0	84.5	86.0	89.4	80.1	63.7	52.7	49.5	68.8	70.0

Table 5: Experimental results (%) of models trained using different LLM verifiers. Qwen-7B is short for DeepSeek-R1-Distilled-Qwen-7B. The base model used for RL training is TULU 3 SFT.

address this, we explore using smaller reasoning models as LLM verifiers while maintaining comparable performance. A straightforward approach is to distill a verifier from QwQ-32B. Therefore, we distill 130k SFT data instances from QwQ, where each instance consists of an instruction, a response, and a critic indicating whether a response satisfies the given constraints in the instruction. The data collection process is detailed in Appendix C.

We fine-tune DeepSeek-R1-Distill-Qwen-7B on the collected dataset, resulting in **IF-Verifier-7B**. We then conduct RL training on TULU 3 SFT using the new LLM verifiers in VERIF. Figure 7 shows the reward curves during training. We can observe that DeepSeek-R1-Distill-Qwen-7B yields higher initial rewards, but its reward growth is limited. IF-Verifier-7B exhibits a similar reward trajectory as QwQ-32B. The results of the trained models are shown in Table 5. We observe that using IF-Verifier-7B as the LLM verifier in VERIF significantly outperforms DeepSeek-R1-Distill-Qwen-7B and achieves competitive performance to QwQ-32B. Moreover, IF-Verifier-7B reduces computational cost a lot. Deploying IF-Verifier-7B requires only one single H800 GPU, with an average reward computation time of 120 seconds per batch, which can be further reduced with multi-GPUs. This makes VERIF a practical method for effective RL training under limited resources. This work preliminarily explores more efficient LLM verifiers and encourages further efforts (Liu et al., 2025b).

5 Related Work

Instruction following requires models to generate responses that satisfy complex user instructions. Recent work has primarily focused on following constraints in instructions, such as length and keyword (Zhou et al., 2023). Existing efforts to enhance instruction-following capabilities primarily focus on methods for (1) collecting SFT data, including directly distilling from larger LLMs (Sun et al., 2024; He et al., 2024a; Dong et al., 2024; Ren et al., 2025), back-translation (Qi et al., 2024; Pham et al., 2024), and training dedicated instruction composers (An et al., 2025), and (2) collecting preference pairs (Cheng et al., 2024; Pham et al., 2024; Dong et al., 2024; Zhang et al., 2024b). Notably, two works are similar to ours: AutoIF (Dong et al., 2024), which constructs both instructions and corresponding verification code, but but does not explore RL training or soft constraints verification; and TULU 3 (Lambert et al., 2024), which adopts RLVR for instruction following but the improvement is limited and also does not consider soft constraints. In summary, the best practice for RL in instruction following remains underexplored.

As RL has proven to be an effective post-training technique, many prior studies have explored its applications across various domains, primarily focusing on verification engineering, such as math (Lambert et al., 2024; Guo et al., 2025; Luo et al., 2025b; ByteDance-Seed, 2025), code (Wang et al., 2024b; Luo et al., 2025a), logic (Xie et al., 2025), tool using (Feng et al., 2025; Jin et al., 2025; Qian et al., 2025a; Li et al., 2025; Zheng et al., 2025), machine translation (Wang et al., 2024a; He et al., 2025), medicine (Chen et al., 2024; Wang et al., 2025), and finance (Qian et al., 2025b; Liu et al., 2025a). In this work, we explore the best practice of RL for instruction following and propose VERIF, an effective verification method for RL training.

6 Conclusion

In this work, we propose VERIF, an effective verification method for RL in instruction following. We also construct VERINSTRUCT, a dataset for instruction following where each instruction is paired with corresponding verification signals. We perform RL training with VERIF on VERINSTRUCT, leading to significant improvements. The trained models achieve SoTA performance on several representative instruction-following benchmarks at a similar model scale, without hurting general capabilities. This work demonstrates the promising potential of RL in instruction following, and we encourage further exploration of novel RL methods and data.

Limitations

We discuss the limitations of our work here, including two main aspects: (1) The training dataset VERINSTRUCT includes only English data, which may limit the broader usage of the dataset. We observe that RL on VERINSTRUCT still generalizes well to multiple languages, and we encourage the community to collect more diverse data covering more languages. (2) VERIF relies on an LLM as the verifier, which inherits common issues of LLM-as-a-judge, such as potential biases (Ye et al., 2024) and vulnerability to adversarial attacks (Shi et al., 2024). We believe developing more robust and efficient LLM judges (Liu et al., 2025b) is a promising direction and leave it for future work.

Ethical Considerations

We discuss potential ethical concerns as follows: (1) Intellectual property. Alpaca-GPT4 and Infinity-Instruct are licensed under CC BY-NC 4.0[2], OpenAssistant is licensed under Apache License 2.0[3]. WildChat is licensed under ODC-By license[4]. Evol-Instruct and Orca-Chat do not specify explicit licenses. We strictly adhered to all claimed licenses. Our dataset will be released under the Apache License 2.0. We believe the original open-source datasets are properly anonymized, and we do not introduce any additional sensitive information. (2) Potential risk control. In this paper, we propose VERIF, a verification method for RL to improve instruction-following capabilities of LLMs. As VERIF includes an LLM verifier, it inherits the known risks of LLMs, such as potential bias (Gallegos et al., 2024; Ye et al., 2024). We do not introduce any additional risks. Users should not exploit VERIF for reward hacking (Skalse et al., 2022) and are responsible for verifying the compliance of the models trained using it. (3) AI assistance. We use ChatGPT and Claude to refine some sentences.

References

Kaikai An, Li Sheng, Ganqu Cui, Shuzheng Si, Ning Ding, Yu Cheng, and Baobao Chang. 2025. Ultraif: Advancing instruction following from the wild. *arXiv preprint arXiv:2502.04153*.

BAAI. 2024. Infinity instruct. *arXiv preprint arXiv:2406.XXXX*.

Thomas Ball, Shuo Chen, and Cormac Herley. 2024. Can we count on llms? the fixed-effect fallacy and claims of gpt-4 capabilities. *Transactions on Machine Learning Research*.

ByteDance-Seed. 2025. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning. *Preprint*, arXiv:2504.13914.

Junying Chen, Zhenyang Cai, Ke Ji, Xidong Wang, Wanlong Liu, Rongsheng Wang, Jianye Hou, and Benyou Wang. 2024. Huatuogpt-o1, towards medical complex reasoning with llms. *arXiv preprint arXiv:2412.18925*.

Jiale Cheng, Xiao Liu, Cunxiang Wang, Xiaotao Gu, Yida Lu, Dan Zhang, Yuxiao Dong, Jie Tang, Hongning Wang, and Minlie Huang. 2024. Spar: Self-play with tree-search refinement to improve instruction-following in large language models. *arXiv preprint arXiv:2412.11605*.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. 2025. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. 2024. Self-play with execution feedback: Improving instruction-following capabilities of large language models. *arXiv preprint arXiv:2406.13542*.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In *Proceedings of NAACL-HLT*, pages 2368–2378.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. 2024. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*.

Shahul Es. 2023. Orca-chat: A high-quality explanation-style chat dataset.

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. *arXiv preprint arXiv:2504.11536*.

Tairan Fu, Raquel Ferrando, Javier Conde, Carlos Arriaga, and Pedro Reviriego. 2024. Why do large language models (llms) struggle to count letters? *arXiv preprint arXiv:2412.18626*.

[2] https://creativecommons.org/licenses/by-nc/4.0/

[3] https://www.apache.org/licenses/LICENSE-2.0

[4] https://opendatacommons.org/licenses/by/1-0/

Isabel O Gallegos, Ryan A Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K Ahmed. 2024. Bias and fairness in large language models: A survey. *Computational Linguistics*, 50(3):1097–1179.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Chenghao Ma, Shanghaoran Quan, Liang Chen, Qingxiu Dong, Runxin Xu, and 1 others. 2025. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *Proceedings of ICLR*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Minggui He, Yilun Liu, Shimin Tao, Yuanchang Luo, Hongyong Zeng, Chang Su, Li Zhang, Hongxia Ma, Daimeng Wei, Weibin Meng, and 1 others. 2025. R1-t1: Fully incentivizing translation capability in llms via reasoning learning. *arXiv preprint arXiv:2502.19735*.

Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. 2024a. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. In *Findings of ACL*, pages 10864–10882.

Yun He, Di Jin, Chaoqi Wang, Chloe Bi, Karishma Mandyam, Hejia Zhang, Chen Zhu, Ning Li, Tengyu Xu, Hongjiang Lv, Shruti Bhosale, Chen Zhu, Karthik Abinav Sankararaman, Eryk Helenowski, Melissa Hall Melanie Kambadur, Aditya Tayade, Hao Ma, Han Fang, and Si-Ying Wang. 2024b. Multi-if: Benchmarking llms on multi-turn and multilingual instructions following. *ArXiv*, abs/2410.15553.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of ACL*, pages 4667–4688.

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*.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, and 1 others. 2024. Openassistant conversations-democratizing large language model alignment. *Advances in NeurIPS*, 36.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, and 1 others. 2024. T\" ulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, and 1 others. 2024. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *arXiv preprint arXiv:2411.16594*.

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

Zhaowei Liu, Xin Guo, Fangqi Lou, Lingfeng Zeng, Jinyi Niu, Zixuan Wang, Jiajie Xu, Weige Cai, Ziwei Yang, Xueqian Zhao, and 1 others. 2025a. Fin-r1: A large language model for financial reasoning through reinforcement learning. *arXiv preprint arXiv:2503.16252*.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. 2025b. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*.

Michael Luo, Sijun Tan, Roy Huang, Ameen Patel, Alpay Ariyak, Qingyang Wu, Xiaoxiang Shi, Rachel Xin, Colin Cai, Maurice Weber, Ce Zhang, Li Erran Li, Raluca Ada Popa, and Ion Stoica. 2025a. Deepcoder: A fully open-source 14b coder at o3-mini level. Notion Blog.

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

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. In *Proceedings of NeurIPs*.

Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. *arXiv preprint arXiv:2304.03277*.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Zijun Yao, Bin Xu, Lei Hou, and Juanzi Li. 2025. Agentic reward modeling: Integrating human preferences with verifiable correctness signals for reliable reward systems. *arXiv preprint arXiv:2502.19328*.

Chau Pham, Simeng Sun, and Mohit Iyyer. 2024. Suri: Multi-constraint instruction following in long-form text generation. In *Findings of ACL*, pages 1722–1753.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2024. Constraint back-translation improves complex instruction following of large language models. *arXiv preprint arXiv:2410.24175*.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tür, Gokhan Tur, and Heng Ji. 2025a. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*.

Lingfei Qian, Weipeng Zhou, Yan Wang, Xueqing Peng, Han Yi, Jimin Huang, Qianqian Xie, and Jianyun Nie. 2025b. Fino1: On the transferability of reasoning enhanced llms to finance. *arXiv preprint arXiv:2502.08127*.

Yanzhao Qin, Tao Zhang, Yanjun Shen, Wenjing Luo, Haoze Sun, Yan Zhang, Yujing Qiao, Weipeng Chen, Zenan Zhou, Wentao Zhang, and 1 others. 2024. Sysbench: Can large language models follow system messages? *arXiv preprint arXiv:2408.10943*.

Team Qwen. 2025. Qwq-32b: Reward model for factuality and harmlessness. Accessed: 2025-05-13.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Proceedings of NeurIPs*.

Qingyu Ren, Jie Zeng, Qianyu He, Jiaqing Liang, Yanghua Xiao, Weikang Zhou, Zeye Sun, and Fei Yu. 2025. Step-by-step mastery: Enhancing soft constraint following ability of large language models. *arXiv preprint arXiv:2501.04945*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junmei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300.

Jiawen Shi, Zenghui Yuan, Yinuo Liu, Yue Huang, Pan Zhou, Lichao Sun, and Neil Zhenqiang Gong. 2024. Optimization-based prompt injection attack to llm-as-a-judge. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 660–674.

Joar Skalse, Nikolaus Howe, Dmitrii Krasheninnikov, and David Krueger. 2022. Defining and characterizing reward hacking. *Advances in Neural Information Processing Systems*, 35:9460–9471.

Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. 2024. Conifer: Improving complex constrained instruction-following ability of large language models. *arXiv preprint arXiv:2404.02823*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of ACL*.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Shengyi Huang, Kashif Rasul, Alvaro Bartolome, Alexander M. Rush, and Thomas Wolf. The Alignment Handbook.

Bingning Wang, Haizhou Zhao, Huozhi Zhou, Liang Song, Mingyu Xu, Wei Cheng, Xiangrong Zeng, Yupeng Zhang, Yuqi Huo, Zecheng Wang, and 1 others. 2025. Baichuan-m1: Pushing the medical capability of large language models. *arXiv preprint arXiv:2502.12671*.

Jiaan Wang, Fandong Meng, Yunlong Liang, and Jie Zhou. 2024a. Drt-o1: Optimized deep reasoning translation via long chain-of-thought. *arXiv e-prints*, pages arXiv–2412.

Junqiao Wang, Zeng Zhang, Yangfan He, Yuyang Song, Tianyu Shi, Yuchen Li, Hengyuan Xu, Kunyu Wu, Guangwu Qian, Qiuwu Chen, and 1 others. 2024b. Enhancing code llms with reinforcement learning in code generation. *arXiv preprint arXiv:2412.20367*.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, and 1 others. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. 2025. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2502.14768*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, and 1 others. 2024. Justice or prejudice? quantifying biases in llm-as-a-judge. In *Neurips Safe Generative AI Workshop 2024*.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025a. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. 2025b. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *Preprint*, arXiv:2504.13837.

Tao Zhang, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, Bin Cui, and 1 others. 2024a. Cfbench: A comprehensive constraints-following benchmark for llms. *arXiv preprint arXiv:2408.01122*.

Xinghua Zhang, Haiyang Yu, Cheng Fu, Fei Huang, and Yongbin Li. 2024b. Iopo: Empowering llms with complex instruction following via input-output preference optimization. *arXiv preprint arXiv:2411.06208*.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatgpt interaction logs in the wild. In *Proceedings of ICLR*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*.

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 preprint arXiv:2311.07911*.

Appendices

A VERINSTRUCT

A.1 Dataset Construction Details

Following Qi et al. (2024), we collect original instructions and responses from several publicly available instruction-tuning datasets, including Alpaca GPT-4 (Peng et al., 2023), Orca Chat (Es, 2023), Evol Instruct (Xu et al., 2023), and OpenAssistant (Köpf et al., 2024). We then apply a back-translation-based method to extract both soft and hard constraints from the instruction-response pairs. Table 6 presents the prompt to generate soft constraints used in VERINSTRUCT construction. The hard constraints, including Length, Keyword, and selected aspects of Format, are automatically generated through Python-based processing. These constraints are subsequently combined to form the final constraint-enhanced prompt.

A.2 Dataset Statistics

Figure 8 shows the distribution of $22,000$ instances in the VERINSTRUCT dataset. Following IFBench (Peng et al., 2025), we categorize constraints into five types: length, keyword, format, content, and style. The left chart presents the proportional distribution of constraint types. Since certain format constraints, such as those requiring hierarchical output structures, are not easily verifiable via Python, we define format, content, and style as soft constraints, which together account for 77.7% of the total. Length and keyword are defined as hard constraints, making up the remaining 22.3%. The right chart categorizes the data by the number of constraints after merging.

B Experimental Details

We train our model using the open-source VeRL framework[5] with the GRPO algorithm (Shao et al., 2024), setting the KL loss coefficient to 1×10^{-3}. The batch size is set to 32, the number of rollouts to 16, the maximum generation length of rollout to $4,096$, and the learning rate to 1×10^{-6}. We save checkpoints every 20 steps during training. Following TULU 3 (Lambert et al., 2024), we use IFEval (Zhou et al., 2023) as the validation set to select the best checkpoint. We train the models for one epoch on VERINSTRUCT with early stopping if performance on IFEval does not improve for more than 5 checkpoints. The best checkpoints

[5] https://github.com/volcengine/verl

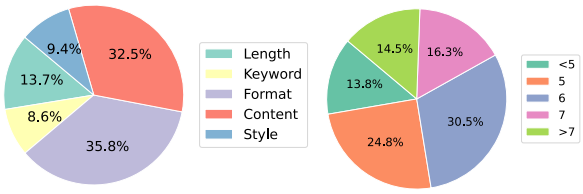

Figure 8: **left:** Proportional distribution of constraint types in the dataset. **right:** Distribution of the number of constraints per instruction.

are typically found within the first 200 steps. For evaluation, we set the sampling temperature to 0 to ensure reproducibility. For all evaluations using LLM-as-a-judge, we adopt gpt-4o-2024-11-20 as the judge. Since the Conifer model (Sun et al., 2024) is not publicly open-sourced, we instead train a model using its SFT and DPO data, and the reported results of Conifer are evaluated based on our reproduced model. For the evaluation of reasoning LLMs, we remove thinking tokens and evaluate only the final responses. For evaluation of general capabilities, we report the length-controlled win rate for AlpacaEval 2.0 (Dubois et al., 2024). Both training and evaluation are conducted on Nvidia H800 GPUs, with the entire training process taking approximately $1,900$ GPU hours in total.

C Training a Small Verifier

We provide a detailed description of the training data construction process and training details. Following the construction of VERINSTRUCT, we first generate an additional $20,000$ data instances. To ensure diversity, we additionally mined complex instructions from WildChat (Zhao et al., 2024) and Infinity Instruct (BAAI, 2024). Specifically, we use Qwen2.5-72B-Instruct to extract constraints from each instruction and classify them as hard or soft. For hard constraints, we adopt Qwen2.5-72B-Instruct to generate corresponding verification Python code scripts. The full prompt is presented in Table 7. For each instruction, we randomly sample a response from 6 different models, including Llama3.1-8B-Instruct (Grattafiori et al., 2024), Llama-3.3-70B-Instruct (Grattafiori et al., 2024), Qwen2.5-7B-Instruct (Yang et al., 2024), Qwen2.5-72B-Instruct (Yang et al., 2024), QwQ-32B (Qwen, 2025), DeepSeek-R1-Distilled-Qwen-32B (Guo et al., 2025). We then adopt QwQ-32B to generate a step-by-step verification indicating whether the output satisfies the instruction for each instruction-response pair. As a result, we collect

about 130k instruction–response pairs with corresponding step-by-step verification. For SFT training, we use the open-source alignment-handbook framework (Tunstall et al.). Based on DeepSeek-R1-Distill-Qwen-7B, we train the model on the collected dataset for 2 epochs, with 2×10^{-5} learning rate, 64 batch size, $8,192$ max sequence length, resulting in the verifier IF-Verifier-7B.

Prompt: Generating Constraints from Instruction and Output

As a linguist with expertise in contextual language nuances, please add constraints to enrich the #Given Instruction# based on the #Given Output#. The goal is to enhance the specificity and detail of the instruction to ensure that the response is more aligned with the output text.
To supplement the instruction using the output, please consider adding **specific and detailed** constraints across the following dimensions:

- **Desired_Writing_Style**: Specify the intended tone or narrative voice, such as humorous, formal, poetic, or conversational.
- **Semantic_Elements**: Clarify the core meaning, focus, or conceptual emphasis that the response should reflect.
- **Morphological_Constraints**: Indicate any forbidden words, expressions, or formatting (e.g., avoid passive voice or markdown).
- **Multi-lingual_Constraints**: Specify the language(s) or code-switching rules to be used in the response.
- **Hierarchical_Instructions**: Define a priority order among multiple tasks, outlining how they should be structured or emphasized.
- **Special_Output_Format**: Specify required formats, such as Python code, JSON structure, tables, LaTeX, or HTML.
- **Paragraphs_Constraints**: Indicate how many paragraphs are needed, and whether any separators (e.g., horizontal lines, "***") should be used.
- **Specific_Sentence**: Require inclusion of a specific sentence at the beginning or end of the response.
- **Key_Formatting**: Specify formatting of key phrases—such as using **bold**, *italics*, or ALL CAPS—based on content in the #Given Output#.
- **Item_Listing_Details**: Define how items should be listed, including use of symbols like bullets (•), numbers (1., 2., 3.), or dashes (-).

#Given Instruction#
`{Instruction}`

#Given Output#
`{Response}`

Please format your response directly in JSON, using `"Constraint_Type"` as the key and the specific constraint as its value. Ensure that each constraint is a concise and complete sentence of 10–20 words, and use varied phrasing across types.
If a specific type of constraint cannot be derived from the #Given Output#, assign the value `"NULL"`. For example: `"Constraint_Type": "NULL"`,
Do not include any headings or prefixes in your response.

Table 6: Prompt for generating format, content, and style constraint types based on the back-translation method.

Prompt: Extracting Constraints from Instruction

You are an expert in natural language processing and constraint checking. Your task is to analyze a given instruction and identify which constraints need to be checked.
The 'instruction' contains a specific task query along with several explicitly stated constraints. Based on the instructions, you need to return a list of checker names that should be applied to the constraints.
[Task Example 1]
Instruction: Write a 300+ word summary of the Wikipedia page "https://en.wikipedia.org/wiki/Raymond_III_Count_of_Tripoli". Do not use any commas and highlight at least 3 sections that have titles in markdown format, for example *highlighted section part 1*, *highlighted section part 2*, *highlighted section part 3*.
Response: NumberOfWordsChecker: 300+ word <sep> HighlightSectionChecker: highlight at least 3 sections that have titles in markdown format <sep> ForbiddenWordsChecker: Do not use any commas.
#Task Instruction#
`{Instruction}`

Your task:
- Generate the appropriate checker names with corresponding descriptions from the original instruction description. - Return the checker names with their descriptions separated by <sep>.
- Focus only on the constraints explicitly mentioned in the instruction.
- Ensure that each constraint is complete, such as specifying whether the 300-word limit applies to the entire text or a specific section. A defined scope is required.
- Do **not** generate checkers for the task query itself or its quality.
- If the instruction is in Chinese/English, please output the constraint in the same language.
- Each checker should be responsible for checking only one constraint.
- Do not output any constraints that are not included in the instruction.

Prompt: Classifying Constraints

Please classify whether the given checker can be judged using simple lexical rules.
#Checker#
`{checker_name}`

Classification rules:
- If the checker can be determined using simple lexical rules—such as word count, text length, number of paragraphs, number of sentences, or presence of specific keywords—output [[A]].
- If the checker requires semantic understanding—such as style, tone, sentiment, language, context, genre, or structure—and thus necessitates an additional semantic analysis model (e.g., a large language model), output [[B]].
- If the constraint is meaningless, non-informative, or irrelevant (e.g., "NA"), output [[C]].

Prompt: Generating code

You are tasked with implementing a 'Python' function 'check_following' that determines whether a given 'response' satisfies the constraint defined by a checker. The function should return 'True' if the constraint is satisfied, and 'False' otherwise.
[Example Input 1]
no more than 800 words
[Example Output 1]
def check_following(response): return len(response.split()) <= 800
[Example Input 2]
Include keywords 'cloud storage', 'open-source'
[Example Output 2]
import re def check_following(response): return bool(re.search(r'cloud storage', response, re.IGNORECASE) and re.search(r'open-source', response, re.IGNORECASE))
[Example Input 3]
The word 'huge' should appear 3 times
[Example Output 3]
import re def check_following(response): return len(re.findall(r'huge', response, re.IGNORECASE)) == 3
#Task Input Checker#
`{checker_name}`

[Requirements]
- The function should be self-contained with necessary imports.
- DO NOT use nltk.
- Only return exactly 'Python' code script, without any other info.

Table 7: Prompt for extracting constraints from instruction, classifying constraint types, and generating code for hard constraints.