STAIF: A Stage-wise Optimization for Complex Instruction Following

Anonymous ACL submission

Abstract

Following complex instructions with multiple explicit constraints remains a fundamental challenge for large language models (LLMs). Existing alignment methods, such as DPO, optimize holistic reward signals that often underemphasize strict satisfaction of individual constraints, particularly under out-of-distribution or multi-constraint settings. In this paper, we propose STAIF, a stage-wise optimization framework that decouples the alignment of subjective (soft) constraints from the optimization of objectively verifiable (hard) constraints. Stage 1 applies preference optimization with multiple negative samples to sharpen sensitivity to soft constraints, while Stage 2 applies Reinforcement Learning with Verifiable Rewards (RLVR) to enforce strict compliance with hard constraints. To support this method, we construct STAINSTRUCT, a high-quality bilingual (English, Chinese) dataset of approximately 31,000 complex multi-constraint instructions. Extensive analyses validate the design of STAIF and show state-of-the-art performance on representative benchmarks against strong baselines, as well as genuine generalization. Our code and dataset are released at https://anonymous.4open.science/r/STAIF-8F0F.

1 Introduction

Large language models (LLMs) have evolved into general-purpose assistants capable of handling diverse real-world tasks through natural language instructions. Users often specify multiple constraints simultaneously, spanning requirements on content, format, style, and more. In these settings, partial compliance is often insufficient; missing just one critical constraint can make the response unusable.

Despite progress in instruction tuning and preference alignment, reliable multi-constraint instruction following remains a fundamental challenge. Recent notable works such as Reinforcement

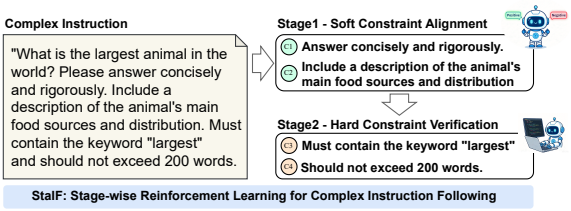

Figure 1: Illustration of STAIF: Stage 1 soft-constraint alignment; Stage 2 hard-constraint verification.

Learning from Human Feedback (RLHF), Direct Preference Optimization (DPO), and Tulu3 (Christiano et al., 2017; Ouyang et al., 2022; Rafailov et al., 2023; Lambert et al., 2024) typically optimize responses using a holistic preference or verification signal. While this paradigm is effective for overall helpfulness, it can obscure which individual constraints are satisfied or violated. A deeper issue is that constraints are heterogeneous in nature: subjective (soft) constraints (e.g., "in an encouraging tone") require comparative preference judgments, whereas objective (hard) constraints (e.g., "output valid JSON") demand deterministic, rule-based verification. Treating both types with a single holistic objective dilutes the binary correctness signal for hard constraints and fails to capture the nuanced preference distinctions needed for soft ones.

Compounding these methodological challenges is the scarcity of high-quality, large-scale training data specifically designed for complex instruction following. Several benchmark datasets, such as IFEval, IFBench, and CFBench, have recently been introduced to evaluate instruction following abilities (Zhou et al., 2023; Jiang et al., 2024; Zhang et al., 2025a), but constructing corresponding training data at scale remains difficult due to inherent challenges in ensuring constraint compatibility, avoiding logical conflicts, etc. This shortage is particularly acute for Chinese, where natively constructed complex-instruction datasets remain extremely limited.

To address these challenges, we propose Stage-

wise Instruction Following (**STAIF**), a two-stage alignment framework that explicitly decouples soft-constraint preference alignment from hard-constraint verifiable optimization (Figure 1). In Stage 1, we perform preference optimization to teach the model to distinguish responses that satisfy the intended soft constraints from those that omit or violate them, with a richer learning signal through multiple negative responses for better discrimination. In Stage 2, we apply Reinforcement Learning with Verifiable Rewards (RLVR) through Group Relative Policy Optimization (GRPO, Shao et al. (2024)) to directly optimize for strict satisfaction of objectively hard constraints. This stage-wise design provides separate learning signals to different constraint types, enabling the model to improve both preference alignment and strict constraint compliance without trade-offs.

To support our stage-wise training approach, we develop an automated complex-constraint data synthesis framework that explicitly models task-constraint compatibility and filters invalid, redundant, or conflicting constraints during expansion. Using this framework, we construct **STAINSTRUCT**, a large-scale bilingual dataset (30% English and 70% Chinese) containing approximately 31,000 complex multi-constraint instructions. For each instruction, we generate high-quality positive responses, multiple negative responses according to the soft constraints, and rule-based verification code for hard constraints, making the dataset uniquely suitable for both preference learning and verifiable-reward reinforcement learning.

We evaluate STAIF on three instruction-following benchmarks: IFEval, IFBench, and CF-Bench. Experiments on two competitive open-source base models, Qwen2.5-7B-Instruct and Tulu3-8B-SFT, show consistent and significant performance improvements across all benchmarks. Notably, STAIF trained on Tulu3-8B-SFT achieves performance competitive with large scale models such as Gemini 2.5 on IFEval and IFBench despite being substantially smaller in model size. Ablation studies confirm that both training stages contribute meaningfully to the final performance, and experiments with negative sampling demonstrate that multiple negative responses provide stronger contrastive learning signals. Fine-grained analysis further reflects genuine generalization rather than in-distribution memorization.

The key contributions of this paper are summarized as follows:

- We propose STAIF, a novel two-stage optimization framework with multiple negative samples that addresses the performance bottleneck of complex instruction following by decoupling soft-constraint preference alignment from hard-constraint verification.
- We propose an automated complex-constraint data synthesis framework and STAINSTRUCT, a large-scale high-quality bilingual dataset for complex instruction following.
- Extensive experiments on various instruction following benchmarks demonstrate state-of-the-art (SoTA) performance and strong generalization by STAIF.

2 Related Work

2.1 Complex Instruction Data Construction

Early instruction tuning works, including Alpaca [1], Vicuna[2], and CAMEL (Li et al., 2023), demonstrated the power of synthetic instruction data for open-source LLMs, but their datasets focused mainly on general instruction following or relatively simple constraints. To address this, Evol-Instruct pioneered depth/width instruction expansion using GPT-4 to generate more complex instructions (Xu et al., 2023), while Conifer proposed progressive learning for multi-level constraint following (Sun et al., 2024). AutoIF further automated the generation of instructions and verification code (Dong et al., 2025). However, existing datasets suffer from: 1) a lack of explicit task-constraint compatibility modeling, leading to invalid or conflicting constraints; 2) insufficiently diverse negative samples for contrastive learning; and 3) a scarcity of high-quality Chinese data. We construct STAINSTRUCT to address these issues.

2.2 Preference-based Alignment Methods

RLHF and DPO have become standard for LLM alignment, but their holistic reward signals can obscure individual constraint satisfaction, diluting learning signals in multi-constraint settings. Recent works such as MuSC (Huang et al., 2025) and IOPO (Zhang et al., 2025b) enhance constraint sensitivity through multi-granularity self-contrastive training and input-output preference optimization, but still treat soft and hard constraints uniformly,

[1] https://crfm.stanford.edu/2023/03/13/alpaca.html

[2] https://www.lmsys.org/blog/2023-03-30-vicuna/

failing to provide the deterministic signals required for strict hard-constraint compliance. In this paper, we apply preference alignment only to soft constraints and optimize hard constraints separately with a different approach.

2.3 Verification-based Methods

RLVR has proven to be effective for programmatically verifiable tasks such as mathematical reasoning and instruction following (Shao et al., 2024; Lambert et al., 2024). In instruction following, Tulu 3 applied RLVR with modest gains, while AutoIF and VerIF improved verification pipelines (Lambert et al., 2024; Dong et al., 2025; Peng et al., 2025). However, these works focus primarily on hard constraints and under-emphasize subjective soft preferences that are critical for user satisfaction. We design a two-stage framework to enhance for both soft and hard constraints.

3 Preliminaries

In this section, we formally define the instruction-following setting and summarize common alignment objectives used throughout the paper. Let x denote an instruction and y a model response. A policy $\pi_\theta(y \mid x)$ generates responses with parameters θ, and π_{ref} represents a base reference policy.

3.1 DPO

DPO bypasses explicit reward modeling and optimizes π_θ directly from preferences (Rafailov et al., 2023). Under an implicit reward parameterization, it yields the following maximum-likelihood objective:

$$\begin{aligned}&\mathcal{L}_{\text{DPO}}(\pi_\theta;\pi_{\text{ref}}) = \\ &-\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_p}\Big[\log\sigma\Big(\beta\log\tfrac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} \\ &-\beta\log\tfrac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\Big)\Big]\end{aligned} \tag{1}$$

with an implicit reward form:

$$r(x,y) = \beta\log\tfrac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} + \beta\log Z(x), \tag{2}$$

where $Z(x)$ is a partition function.

3.2 RLVR

In settings with objectively checkable outcomes, reinforcement learning can use a deterministic verifier $V(x,y) \in \{0,1\}$ (or a scalar score) as reward, avoiding a learned reward model. This verification reinforcement learning setup is commonly used when the correctness can be programmatically validated (Shao et al., 2024; Peng et al., 2025).

3.3 GRPO

GRPO estimates the advantages from a group of sampled completions $\{o_1,\ldots,o_G\}$ for the same prompt and uses normalized group-relative rewards (Shao et al., 2024). Given rewards $\{r_i\}_{i=1}^G$, the normalized advantage is

$$A_i = \frac{r_i - \text{mean}(r_1,\ldots,r_G)}{\text{std}(r_1,\ldots,r_G)}. \tag{3}$$

4 Data: StaInstruct

In this section, we propose an automated complex-constraint data synthesis framework and construct StaInstruct, a large-scale dataset designed for complex, multi-constraint instruction following tasks. The dataset is constructed in both English and Chinese, addressing the lack of Chinese benchmarks that capture linguistic nuances and culturally specific constraints. Figure 2 provides an overview of the data construction pipeline.

4.1 Data Construction

Taxonomy. A complex instruction can be segmented into two parts: a *task* (e.g., "write an essay") and a set of *constraints* (e.g., "in Markdown format").

To systematically ensure task diversity, constraint validity, and constraint complexity, we construct a taxonomy of both tasks and constraints, including the mapping between them. We group tasks into seven main categories, such as *Semantic Understanding* and *Analysis*, with 28 subcategories in total. For constraints, we adopt the comprehensive constraint type system of Jiang et al. (2024) and Zhang et al. (2025b), which contains five constraint categories (*Content*, *Situation*, *Style*, *Format*, *Example*) and 26 subcategories.

We analyze real-world user data to map out the relationship between task categories and constraint categories (Figure 3). It can be observed that for a task category, certain constraints are more likely to be posed by users than others. To ensure the validity of generated instructions in later steps, we restrict the allowable constraint categories for each task category accordingly.

Detailed descriptions of the task and constraint taxonomies can be found in Appendix A.

Instruction Expansion. Based on the categorization, instructions, especially simple instructions, from our data pool are expanded through LLM inference to form an input set of complex instructions. A human-LLM collaborative quality check

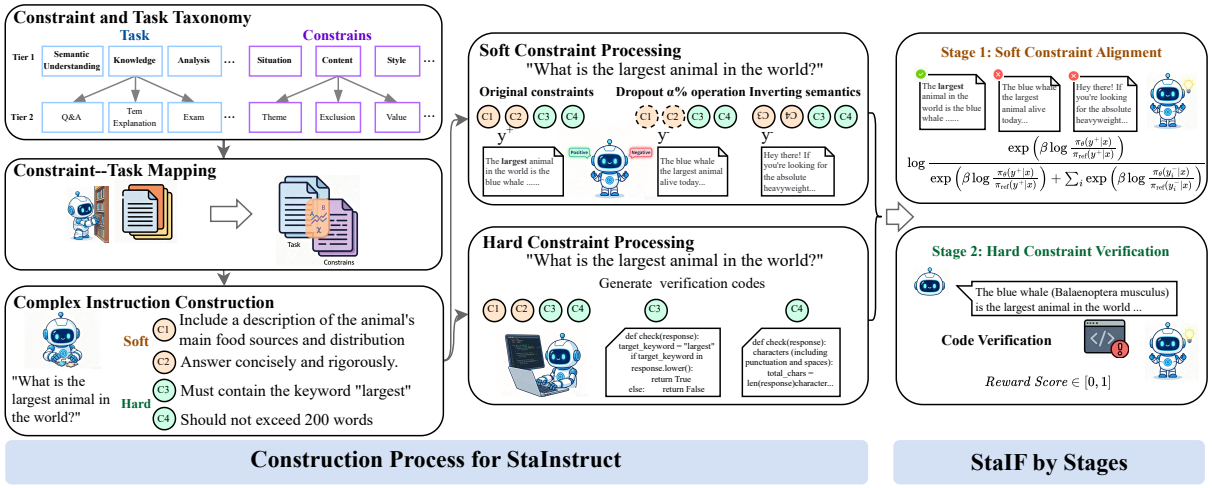

Figure 2: Overview of the data construction pipeline for STAINSTRUCT.

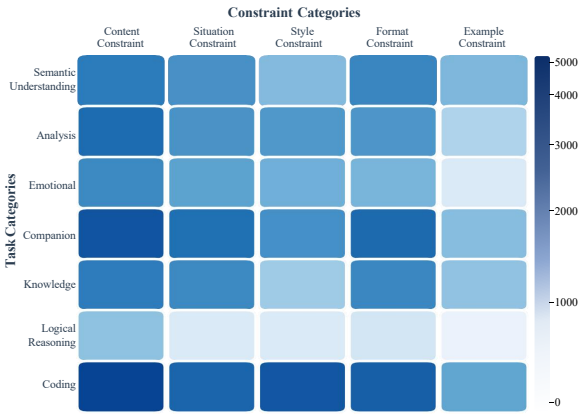

Figure 3: Mapping between task categories and constraint categories based on real-world user data.

is in place to remove redundant, conflicting, or inappropriate constraints.

Constraint Processing. We first parse each instruction into a structured list of constraints, separating soft and hard constraints.

Following the algorithms in Section 5, for each constructed instruction x, we generate multiple negative samples $\{x_1^-, x_2^-, ..., x_n^-\}$ by 1) removing one or more soft constraints mimicking the situation of missing constraints; or 2) inverting the semantics of soft constraints mimicking the situation of constraint violations.

For hard constraints, we leverage the LLM code generation capability to generate rule-based verification code to enable automated deterministic constraint satisfaction checks. Invalid or syntactically erroneous code is corrected by human-LLM collaborative revision.

Response Generation and Verification. For each instruction x, we prompt an LLM to generate a set of responses, including a positive response y^+ that fully follows the exact set of constraints outlined in x, and multiple negative responses $\{y_1^-, y_2^-, ..., y_n^-\}$ generated from the negative sample set $\{x_1^-, x_2^-, ..., x_n^-\}$.

To validate the selected y^+, we follow previous work on automatic instruction-following verification and preference construction (Peng et al., 2025; Zhang et al., 2025b) by combining rule-based checks for objective constraints with model-based judgments paired with human review for subjective constraints. If no candidate satisfies all objective constraints, we regenerate responses or discard the instruction during filtering.

We use specialized LLMs for the aforementioned constructing steps to ensure data construction quality. For instruction expansion, constraint processing, and response generation/verification, we use GPT-4.1[3] for English and Doubao 1.8[4] for Chinese. For code generation, we use Doubao 1.8.

4.2 Dataset Statistics.

The constructed dataset, STAINSTRUCT, contains 31,088 instructions in total, among which 70% are in Chinese and 30% in English. The number of constraints per instruction ranges from 1 to 29, with an average of 7.8. Figure 4 shows the distribution of constraint counts. Around 47% of the constraints are soft constraints, while 53% are hard constraints. Figure 5 summarizes the constraint-

[3] https://openai.com/index/gpt-4-1/
[4] https://seed.bytedance.com/blog/official-release-of-seed1-8-a-generalized-agentic-model

type composition of STAINSTRUCT, with the inner circle representing the distribution of the five major categories as described in Section 4.1 and the outer circle representing that of the sub-categories.

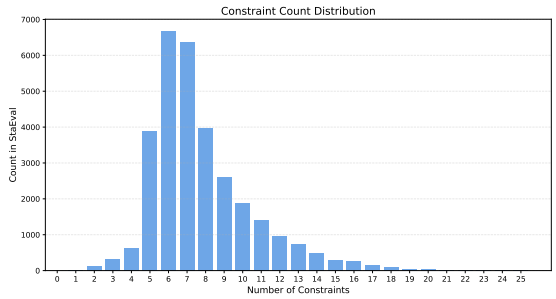

Figure 4: Distribution of the number of constraints per instruction in STAINSTRUCT.

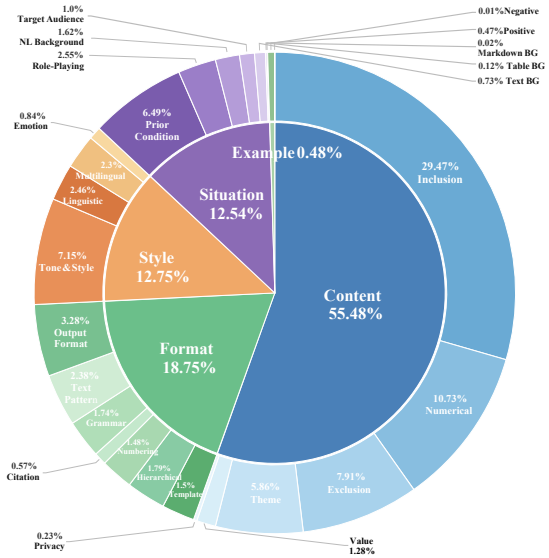

Figure 5: Constraint type distribution in STAINSTRUCT.

5 Method

In this section, we propose STAIF, a stage-wise optimization algorithm designed for complex instruction following.

We assume that an instruction x consists of a set of constraints $C = \{c_1, c_2, \dots\}$. Our complete optimization pipeline has two sequential stages: 1) preference alignment for soft constraints, and 2) RLVR for hard constraints (Shao et al., 2024; Peng et al., 2025), decoupling the optimization for the two types of constraints with customized paradigms and training objectives.

Stage 1: Preference Alignment for Soft Constraints. The core optimization objective of this stage is to enable the model to achieve precise alignment with the implicit, subjective preference-based constraints embedded in complex instructions.

In view of the core characteristics of subjective constraints, which are the absence of absolute quantitative criteria and the reliance on human preference judgment, we adopt a DPO-style objective (Rafailov et al., 2023) rather than conventional RLHF. Standard RLHF requires a separate reward model and PPO optimization, which can be costly and unstable. DPO directly optimizes the policy from pairwise preferences through an implicit reward function, providing a simpler and more stable way to capture fine-grained preference signals.

As described in Section 4, for each instruction x in STAINSTRUCT, we construct one preferred response y^+ and multiple dispreferred responses $\{y_1^-, y_2^-, \dots, y_n^-\}$. Our setting corresponds to a "winner-takes-all" choice among multiple candidates, where only the top choice (the preferred response that satisfies the exact set of constraints of an instruction as discussed in Section 4) is preferred and the rest (dispreferred responses) are unranked "losers". We therefore adopt the Plackett–Luce model (Plackett, 1975; Luce, 2012), and derive our preference expression as follows:

$$p(y^+ \succ \{y_1^-, \dots, y_n^-\} \mid x) = \frac{\exp(r(x, y^+))}{\exp(r(x, y^+)) + \sum_{i=1}^{n} \exp(r(x, y_i^-))}, \quad (4)$$

where $r(x, y)$ is the reward function in Equation 2.

We then construct the corresponding maximum likelihood objective function using the aforementioned preference function for multi-candidate preference probabilities:

$$\mathcal{L}_{\text{StaIF}}(\pi_\theta; \pi_{\text{ref}}) = -\,\mathbb{E}_{(x,\mathbf{y})\sim\mathcal{D}}\left[\log \mathsf{S}_{\mathbf{y}}\left(\beta \log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)}\right)\right], \quad (5)$$

where $\mathbf{y} = \{y^+, y_1^-, \dots, y_n^-\}$, $\mathsf{S}(x)$ is the *softmax* function, and β controls the strength of the implicit KL regularization.

In particular, our objective function is mathematically equivalent to InfoNCE loss (Oord et al., 2018), a foundational objective in contrastive learning. This connection provides a theoretical justification for our use of multiple negative samples: increasing the number of negatives tightens the mutual information lower bound optimized by InfoNCE, leading to more robust learning of constraint satisfaction patterns. This is consistent with the experimental results in Section 6.3.

Detailed derivations of the above can be found in Appendix B.

Stage 2: Verification for Hard Constraints. In this stage, we optimize objective constraint satisfaction using RLVR, addressing a core limitation of existing preference optimization methods: insufficient learning signals for hard constraints. The rewards in RLVR are computed by deterministic checkers (Zhou et al., 2023; Peng et al., 2025) through the following steps: 1) For each hard constraint c_j^H of instruction x, construct the verification function $f_j(x, y)$ based on the rule verification code generated in Section 4.1. $f_j(x, y) = 1$ if y meets constraint c_j^H, and $f_j(x, y) = 0$ otherwise. 2) The reward function is constructed as the average of the verification functions of all hard constraints in x: $R(x, y) = \text{avg}_j(f_j(x, y))$.

With the reward function, we then adopt the GRPO algorithm as the core reinforcement learning optimization framework for this stage (Shao et al., 2024). Compared with the conventional PPO algorithm, GRPO eliminates the need for an additional value network, and is optimally suited for the large-model fine-tuning scenario based on deterministic rule-based rewards in this study.

6 Experiments

6.1 Experimental Setup

Three representative instruction-following benchmarks are used to evaluate our method's performance: **IFEval**, **IFBench**, and **CFBench**. For IFEval and IFBench, we report prompt-level and instruction-level accuracy under both strict and loose matching. For CFBench, we report constraint satisfaction rate (CSR), instruction satisfaction rate (ISR), and priority satisfaction rate (PSR) to provide a comprehensive view of constraint-following performance.

We train STAIF on two base models: **Qwen2.5-7B-Instruct** (Hui et al., 2024) and **Tulu3-8B-SFT** (with RL, Lambert et al. (2024)). These models represent competitive instruction-tuned baselines from different model families, allowing us to assess the generalizability of our approach. Detailed implementation settings are given in Appendix C.

We compared STAIF against a broad set of strong open-source and proprietary baselines across the benchmarks, including frontier large/mid-scale models: GPT-4.1, DeepSeek-v3.2 (Liu et al., 2025), Gemini 2.5 (Comanici et al., 2025), Doubao 1.8, and QwQ-32B[5] as general reference, and models with comparable model size for comparison: Llama3.1-8B-Instruct (Grattafiori et al., 2024), Tulu3-8B (Lambert et al., 2024), Crab-7B-DPO (Qi et al., 2025), and VerIF (Peng et al., 2025). For the two base models used to train STAIF, we conduct controlled comparisons by evaluating the same base models trained, respectively, under SFT (Wei et al., 2021) and DPO (Rafailov et al., 2023) with our training data STAINSTRUCT.

6.2 Main Results

Experimental results are in Table 1. STAIF demonstrates strong performance across all benchmarks.

Performance on Par with Large-scale Models. Notably, STAIF trained on Tulu3-8B-SFT achieves competitive or even superior performance compared to models with significantly larger parameter sizes on IFEval and IFBench. Specifically, on IFEval, STAIF on Tulu3-8B-SFT achieves very similar performance with Gemini 2.5, both achieving an average score of 92.0%. On the more challenging IFBench, STAIF achieves an average score of 52.7%, outperforming Gemini 2.5 (46.4%) by a notable margin of +6.3%. These results demonstrate the effectiveness of our approach in improving instruction-following capabilities.

SoTA Performance among Models of Similar Size. STAIF establishes a clear and consistent advantage among models of comparable parameter scale. On IFEval and IFBench, STAIF on Tulu3-8B-SFT achieves SoTA performance, while on CFBench, STAIF on Qwen2.5-7B-Instruct achieves SoTA performance. Specifically, STAIF on Tulu3-8B-SFT outperforms Tulu3-8B by an average margin of +9.4% on IFEval, +26.2% on IFBench, while STAIF on Qwen2.5-7B-Instruct outperforms all models of similar size by an average margin of at least +12.7%.

Advantage over Single Training Paradigms. Compared with VerIF, which optimizes both types of constraints jointly in a single stage (Peng et al., 2025), STAIF achieves performance improvements of +3.9% on IFEval, +27.1% on IFBench, and +1.7% on CFBench, justifying the advantage of treating soft and hard constraints in separate stages. Here we use the results under Tulu3-8B-SFT for both models for comparability.

[5] https://qwenlm.github.io/blog/qwq-32b/

Model	IFEval				IFBench				CFBench		
	P-S	**P-L**	**I-S**	**I-L**	**P-S**	**P-L**	**I-S**	**I-L**	**CSR**	**ISR**	**PSR**
GPT-4.1	87.8	91.2	91.1	93.2	38.7	43.8	42.1	47.5	83.0	61.0	70.0
DeepSeek-v3.2	86.4	90.5	90.8	93.5	39.7	43.5	42.3	46.8	87.0	65.0	72.0
Gemini 2.5	89.8	91.1	92.9	94.0	39.7	50.3	43.0	52.5	83.0	60.0	70.0
Doubao 1.8	83.4	87.0	88.5	90.0	29.3	36.4	32.8	39.7	83.0	60.0	68.0
QwQ-32B	83.5	87.7	88.8	91.8	36.3	40.4	39.7	44.2	87.0	61.0	70.0
Llama3.1-8B-Instruct	64.8	69.3	74.6	78.1	22.8	26.9	25.1	31.0	58.0	20.0	26.0
Tulu3-8B	78.2	80.6	84.6	86.9	23.1	27.2	25.7	29.9	74.0	35.0	47.0
Crab-7B-DPO	47.7	57.7	59.9	68.1	15.9	21.8	17.6	24.5	54.0	17.0	23.0
VerIF	84.5	87.1	89.3	91.4	23.5	24.8	25.9	28.1	72.0	33.0	43.0
Qwen2.5-7B-Instruct	69.4	72.6	78.3	81.0	24.4	27.5	27.7	30.7	80.0	44.0	56.0
+SFT	75.8	79.1	82.0	84.8	29.2	34.7	32.5	38.5	81.0	44.0	54.0
+DPO	78.1	81.6	83.8	86.8	35.2	38.7	36.3	40.6	80.0	45.0	56.0
+ STAIF	**87.0**	**88.7**	**91.0**	**92.3**	**44.2**	**48.3**	**46.0**	**49.9**	**82.0**	**51.0**	**61.0**
Tulu3-8B-SFT	68.3	70.7	75.7	78.0	22.1	23.5	22.7	25.1	69.0	29.0	40.0
+SFT (secondary)	77.4	81.3	83.8	87.0	28.6	34.0	31.9	37.0	73.0	33.0	46.0
+DPO	78.2	81.6	84.1	86.5	32.9	37.1	34.1	38.1	70.0	32.0	41.0
+ STAIF	**90.4**	**91.1**	**92.9**	**93.5**	**50.3**	**52.0**	**53.1**	**55.2**	**73.0**	**35.0**	**45.0**

Table 1: Main results on IFEval, IFBench, and CFBench. For IFEval and IFBench we report accuracy (%) at the **prompt** (P) and **instruction** (I) levels under **strict** (S) and **loose** (L) matching: **P-S**, **P-L**, **I-S**, **I-L**. For CFBench, we report **CSR**, **ISR**, and **PSR** (%).

Model	Method	IFEval				IFBench				CFBench		
		P-S	**P-L**	**I-S**	**I-L**	**P-S**	**P-L**	**I-S**	**I-L**	**CSR**	**ISR**	**PSR**
Qwen2.5-7B	**STAIF**	**87.0**	**91.0**	**88.7**	**92.3**	**44.2**	**46.0**	**48.3**	**49.9**	**82.0**	**51.0**	**61.0**
	w/o stage1	80.2	86.3	82.8	88.1	37.8	39.7	41.2	43.9	80.0	45.0	58.0
	w/o stage2	79.4	85.5	82.0	87.6	38.4	40.3	42.5	45.4	81.0	46.0	58.0
Tulu3-8B	**STAIF**	**90.4**	**92.9**	**91.1**	**93.5**	**50.3**	**53.1**	**52.0**	**55.2**	**73.0**	**35.0**	**45.0**
	w/o stage1	85.1	89.3	87.0	90.2	42.2	43.3	45.6	47.8	70.0	33.0	41.0
	w/o stage2	80.1	85.8	83.0	88.2	35.0	37.3	38.7	41.5	72.0	33.0	44.0

Table 2: Ablation study on the two training stages of STAIF, evaluated on Qwen2.5-7B and Tulu3-8B.

In the last part of Table 1, where the two base models are trained with the same training data under different paradigms, we observe that DPO generally yields better performance than SFT. Comparing STAIF to DPO, we observe consistent performance improvements of $+7.2\%/+11.5\%/+17.0\%$ across three benchmarks on Qwen2.5-7B-Instruct, and of $+9.4\%/+17.1\%/+3.3\%$ on Tulu3-8B-SFT. This further supports our design choice of decoupling the alignment of soft constraints from optimization on hard constraints.

6.3 Further Analysis

Ablation Study. To understand the contribution of each training stage, we conduct ablation experiments by removing Stage 1 (w/o stage1) and Stage 2 (w/o stage2) in STAIF respectively.

Results are reported in Table 2. Both stages contribute meaningfully to the overall performance, and removing either one consistently degrades the performance across all benchmarks and both base models. For Qwen2.5-7B, Stage 1 brings an average performance gain of $+5.4\%$, $+6.5\%$, and $+3.7\%$ on IFEval, IFBench, and CFBench, respectively, while Stage 2 separately contributes $+6.1\%$, $+5.5\%$, and $+3.0\%$. On Tulu3-8B, the average performance gain is $+4.1\%$, $+7.9\%$, and $+3.0\%$ for Stage 1, and $+7.7\%$, $+14.5\%$, and $+1.3\%$ for Stage 2. The largest improvements are observed on the out-of-distribution IFBench benchmark, where STAIF achieves up to a $+15.8\%$ performance gain by adding Stage 2, demonstrating that decoupling soft and hard constraint optimization significantly improves generalization to unseen instructions.

Model	Method	Negative Sample	IFEval				IFBench				CFBench		
			P-S	P-L	I-S	I-L	P-S	P-L	I-S	I-L	CSR	ISR	PSR
Qwen2.5-7B	Stage 1 only	Single	76.3	83.5	80.4	86.5	34.0	36.1	38.1	40.6	80.0	43.0	56.0
		Multiple	79.4	85.5	82.0	87.6	38.4	40.3	42.5	45.4	81.0	46.0	58.0
	STAIF	Single	85.0	89.7	86.3	90.6	42.2	44.9	46.5	47.5	80.0	46.0	56.0
		Multiple	**87.0**	**91.0**	**88.7**	**92.3**	**44.2**	**46.0**	**48.3**	**49.9**	**82.0**	**51.0**	**61.0**
Tulu3-8B	Stage 1 only	Single	76.5	83.5	79.4	86.1	31.6	33.7	34.6	37.0	69.0	30.0	40.0
		Multiple	80.1	85.8	83.0	88.2	35.0	37.3	38.7	41.5	72.0	33.0	44.0
	STAIF	Single	87.5	91.0	88.3	91.7	48.3	50.2	50.7	52.8	72.0	33.0	43.0
		Multiple	**90.4**	**92.9**	**91.1**	**93.5**	**50.3**	**53.1**	**52.0**	**55.2**	**73.0**	**35.0**	**45.0**

Table 3: Comparison of single vs. multiple negative samples in STAIF, evaluated on Qwen2.5-7B and Tulu3-8B.

Effect of Multiple Negative Samples. STAIF employs multiple negative samples during training to provide a richer contrastive learning signal. In Section 5, we provide theoretical support for this design choice. To validate it empirically, we compare STAIF as well as the model with only soft-constraint alignment (Stage 1, see Section 5), against the variant trained with only a single negative sample.

Table 3 shows that using multiple negative samples consistently outperforms the single negative variants across both base models and all benchmarks. This performance improvement holds for both the full STAIF model and Stage 1 only, showing that the benefit is not stage-sensitive. This analysis validates our design choice: exposing the model to diverse negative examples during training provides a richer learning signal, enabling it to better discriminate between constraint-satisfying and constraint-violating responses.

Granular Analysis on Constraint Types. To better understand how our method generalizes across different distributions of instruction constraints, we break down performance by constraint type on IFEval and IFBench. We compare the Tulu3-8B base model, Tulu3-8B+Stage 1, and Tulu3-8B+STAIF.

Figure 6 shows that STAIF yields broad and consistent gains across most IFEval constraint categories. Similar improvements are also observed on IFBench. This is particularly noteworthy given that constraints in IFBench are drawn from a substantially different distribution than those in the training data. The observed improvements on most constraint types in IFEval and IFBench serve as evidence of genuine generalization rather than in-distribution memorization.

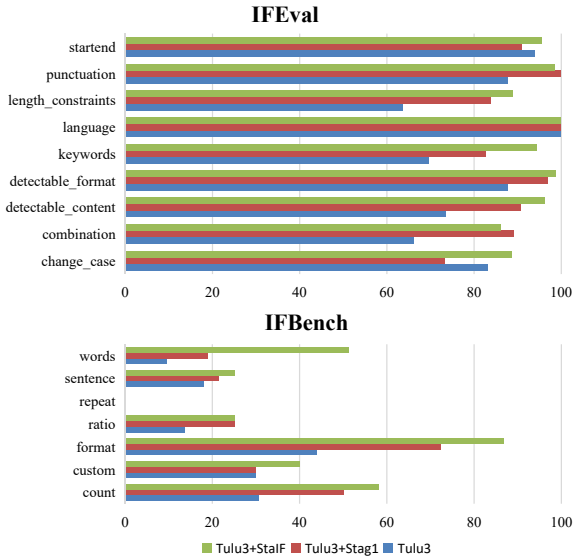

Figure 6: Performance by constraint type on IFEval and IFBench (prompt-level, loose accuracy) for Tulu3, Tulu3+Stage 1, and Tulu3+STAIF.

7 Conclusion

We propose STAIF, a stage-wise framework that separates soft-constraint alignment from hard-constraint optimization, combining preference learning with multiple negative samples and RL with verifiable rewards. We also construct STAINSTRUCT, a large-scale bilingual dataset of complex multi-constraint instructions, under an automated complex-constraint data synthesis framework.

Our methodology achieve SoTA performance on representative benchmarks with different languages and constraint distributions, demonstrating the efficacy of STAIF and the potential of RL in complex instruction following. In future work, we plan to explore further methodology and data enhancement, as well as broader application scenarios.

Limitations

In the training dataset, we set the number of negative samples to be three. The three negative samples are constructed respectively through 1) dropping 50% of the soft constraints; 2) dropping all the soft constraints; and 3) inverting the semantics of all the soft constraints. While the experiment results show effectiveness of using multiple negative samples, the number of negative samples and the construction scheme are not data driven or tuned to achieve better performance.

Ethical Considerations

This work investigates complex instruction-following capabilities of LLMs. For data collection, we utilize real-world user queries that have been thoroughly cleaned to remove all personally identifiable information (PII) as well as toxic or sensitive content. We also employ LLMs for both data generation and automatic evaluation of experimental results. Given that, we acknowledge that our approach inherits the known risks associated with LLMs, including potential risk of bias and fairness (Anthis et al., 2025; Gallegos et al., 2024). We mitigate these risks by manually checking the quality of a random sample of the results.

We use AI assistance (Claude and ChatGPT) to refine some wordings.

References

Jacy Reese Anthis, Kristian Lum, Michael Ekstrand, Avi Feller, and Chenhao Tan. 2025. The impossibility of fair LLMs. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 105–120, Vienna, Austria. Association for Computational Linguistics.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. 2025. Self-play with execution feedback: Improving instruction-following capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pages 39286–39313.

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.

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*.

Hui Huang, Jiaheng Liu, Yancheng He, Shilong Li, Bing Xu, Conghui Zhu, Muyun Yang, and Tiejun Zhao. 2025. Musc: Improving complex instruction following with multi-granularity self-contrastive training. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10667–10686.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

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 the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4667–4688.

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. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, and 1 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*.

R.D. Luce. 2012. *Individual Choice Behavior: A Theoretical Analysis*. Dover Books on Mathematics. Dover Publications.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

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.

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.

Robin L Plackett. 1975. The analysis of permutations. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 24(2):193–202.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2025. Constraint back-translation improves complex instruction following of large language models. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pages 2388–2398.

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. *Advances in neural information processing systems*, 36:53728–53741.

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*.

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*.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.

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*.

Tao Zhang, Chenglin Zhu, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, and 1 others. 2025a. Cfbench: A comprehensive constraints-following benchmark for llms. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 32926–32944.

Xinghua Zhang, Haiyang Yu, Cheng Fu, Fei Huang, and Yongbin Li. 2025b. Iopo: Empowering llms with complex instruction following via input-output preference optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 22185–22200.

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*.

A Taxonomies

Detailed categorization and description for task and constraint:

Category	Sub-category	Description
Semantic Understanding	Semantic Analysis	Analyze the logical structure, semantic relationships, and core meaning of text.
	Sentiment Classification	Identify the emotional tendency or intensity of text.
	Element Extraction	Extract key information units (e.g., people, events, relationships) from unstructured text.
	Element Recognition	Identify extracted entities (e.g., "Apple" → "Organization", "Beijing" → "Location").
	Natural Language Inference	Determine the logical relationship among contents (Entailment/Contradiction/Neutral).
	Text Processing	Preprocessing (cleaning, segmentation, normalization) or postprocessing (formatting, filtering) of text data.
Analysis	Content Analysis	Generate analysis of certain topic or area (e.g., industry, products, culture) as instructed.
	Content Review	Provide subjective evaluations, critical comments, or feedback as instructed.
	Information Inquiry	Provide consulting (e.g., skills) to user queries.
Emotional Companion	Suggestion	Offer practical solutions or actionable advice for user life problems/scenarios.
	Recommendation	Recommend items, content, or services tailored to user preferences or needs.
	Casual Conversation	Engage in informal, natural dialogue to fulfill social or emotional needs.
Knowledge	Knowledge Q&A	Answer factual, conceptual, or procedural questions.
	Term Explanation	Define professional terms, clarify connotations, and illustrate usage scenarios.
	Exam	Solve academic/examination questions (multiple-choice, subjective, computational) across disciplines.
Logical Reasoning	Symbolic Reasoning	Perform logical deductions using mathematical symbols, formal languages, or logical operators.
	Common Sense Reasoning	Infer conclusions based on universal daily life knowledge and intuitive judgment.
	Relational Reasoning	Deduce the connection between entities (e.g., causal, hierarchical, associative relationships).
	Conditional Reasoning	Derive potential consequences or conclusions based on given premises or hypothetical conditions.
Coding	Code Generation	Convert natural language descriptions into executable code snippets or full programs.
	Code Completion	Predict and fill in incomplete code (e.g., function names, syntax, logical blocks) during coding.
	Code Debugging & Fixing	Identify syntax errors, logical bugs, or performance issues and provide corrected code.
	Code Refactoring	Optimize code structure (readability, maintainability, efficiency) without changing core functionality.
	Code Explanation	Clarify the purpose, logical flow, and implementation details of existing code.
Writing	Content Creation	Generate original content (articles, stories, reports, etc.) based on user requirements or themes.
	Summary Generation	Condense long text into a concise summary that retains key information and logical relationships.
	Translation	Convert text between different languages while preserving semantic accuracy, style, and cultural nuances.
	Content Rewriting	Revise existing content to improve clarity, fluency, tone, or adapt to specific usage scenarios.

Table 4: Task Taxonomy.

Category	Sub-category	Description
Content	Theme	The generated content should focus on a specific topic or field.
	Exclusion	Clearly specify the information or content that should not be included in the generated content.
	Inclusion	Clearly specify the particular information or content that must be included in the generated content.
	Value	The generated content should not contain information that violates values, such as safety, false information, discrimination, or bias.
	Numerical	Limit the length and number of words, sentences, and paragraphs in the generated content, or use numerical precision constraints to ensure accuracy.
	Target Audience	The generated content should target a specific audience, which affects terminology, detail level, and complexity.
	Prior Condition	When a specific intention is met, a particular process should be followed to perform an operation or output specific content.
	Natural Language Process Background	Add natural-language process information, such as procedures or business processes, to assist in generating answers.
	Markdown Process Background	Add markdown-formatted process information, such as procedures or business processes, to assist in generating answers.
Situation	Table Background Information	Background information is presented in table form, providing a series of markdown-formatted tables to assist in generating answers.
	Text Background Information	Background information is presented in text form, providing textual background information to assist in generating answers.
Style	Tone and Style	The generated content should adopt a specific tone and style, such as formal, polite, academic, concise, literary, romantic, or sci-fi.
	Emotion	The generated content should express a specific emotion or mood, such as being positive, inspiring, or empathetic.
	Linguistic Characteristics	Use specific linguistic features, such as metaphors, personification, and other rhetorical devices.
	Multilingual	The content should be generated in a specific language or switch between languages according to complex patterns.
Format	Output Format	The generated content should be in a specific data format, such as tables, JSON, HTML, LaTeX, or Markdown.
	Text Pattern	Use specified fonts, font sizes, or special emojis to satisfy formatting requirements.
	Grammar Structure	The generated content should strictly follow specific grammatical structures, such as subject–predicate–object or subject–verb patterns.
	Citation	The generated content should include citations to sources and follow specific citation or reference styles.
	Numbering and List	The generated content should use numbered lists or bullet points to organize information.
	Hierarchical Structure	The generated content should be organized according to a specific hierarchical structure, such as headings and subheadings.
	Template	The generated content should follow a specific layout or format, such as text alignment, paragraph indentation, and introduction–body–conclusion templates.
Example	Positive Example	Provide examples that meet the requirements and require the model to generate content based on these examples.
	Negative Example	Provide examples that do not meet the requirements and require the model to avoid generating similar content.

Table 5: Constraint Taxonomy.

B Mathematical Derivations

B.1 Derivation of Preference Function in Equation 4

The Plackett-Luce model is a probabilistic framework for modeling rankings over a set of discrete alternatives, derived from Luce's choice axiom (Luce, 2012) and Plackett's permutation analysis (Plackett, 1975). The general form of Plackett-Luce model is

$$p(\tau|\mathbf{y}) = \prod_{k=1}^{K} \frac{f(y_{\tau(k)})}{\sum_{j=i}^{K} f(y_{\tau(j)})} \tag{6}$$

where $\mathbf{y} = \{y_1, \ldots, y_K\}$, and τ represents a permutation of $\mathbf{y}$. The general form expresses the probability of a complete ranking. However, in our work, it is a partial ranking case where we only consider the top-ranked option $y_{\tau(1)}$ while the rest are unranked alternatives. Thus the Plackett-Luce model simplifies to:

$$p(y_{\tau(1)} \succ \mathbf{y} \backslash y_{\tau(1)}|\mathbf{y}) = \frac{f(y_{\tau(k)})}{\sum_{j=i}^{K} f(y_{\tau(j)})} \tag{7}$$

Let positive response y^+ be $y_{\tau(1)}$, negative responses $\{y_1^-, \ldots, y_n^-\}$ be $\mathbf{y} \backslash y_{\tau(1)}$, and replace the function f in the equation above with DPO score function $\exp(r(x, y)))$, we get our preference function as shown in Equation 4:

$$\begin{aligned} & p(y^+ \succ \{y_1^-, \ldots, y_n^-\} \mid x) \\ = & \frac{\exp(r(x, y^+))}{\exp(r(x, y^+)) + \sum_{i=1}^{n} \exp(r(x, y_i^-))}, \end{aligned} \tag{8}$$

B.2 Derivation of Maximum Likelihood Loss in Equation 5

With the preference function derived above, and plugging in the softmax function $\mathsf{S}_{\mathbf{y}}(y_i) = \frac{\exp(y_i)}{\sum_j \exp(y_i)}$, we get the maximum likelihood loss function as shown in Equation 9.

B.3 InfoNCE Equivalence

The canonical InfoNCE loss is defined in Equation 10, where x is the *anchor sample*, y^+ is the *positive sample* that is semantically paired with the anchor, $\{y_i^-\}_{i=1}^n$ is a set of *negative samples* that are not paired with the anchor, $s(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is a similarity function measuring the compatibility between anchor x and sample y, and $t > 0$ is the temperature parameter that controls the sharpness of the probability distribution over candidates.

In our study, the "similarity" between an instruction x and a response y is represented by the reward function $r(x, y) = \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$. Substituting the reward model into the canonical InfoNCE loss in Equation 10, and replacing t with $\beta = 1/t$ as the reverse temperature, which is a common thermodynamic interpretation of temperature in probabilistic models, we get our loss function shown in Equation 5 and Equation 9. This equivalence provides a principled explanation for our experimental result that multiple negative samples outperform single negative samples (Table 3). From the contrastive learning perspective:

- The InfoNCE loss optimizes a lower bound on the mutual information $I(x; y^+)$ between instructions and their correct constraint-satisfying responses
- Increasing the number of negative samples n tightens this mutual information lower bound, leading to more robust learning of constraint satisfaction patterns
- This effect is particularly pronounced in out-of-distribution settings (e.g., IFBench), where stronger generalization requires tighter mutual information estimation

$$\begin{aligned} & \mathcal{L}_{\text{StaIF}}(\pi_\theta; \pi_{\text{ref}}) \\ & = -\mathbb{E}_{(x,\mathbf{y})\sim\mathcal{D}} \left[\log \frac{\exp\left(\beta \log \frac{\pi_\theta(y^+|x)}{\pi_{\text{ref}}(y^+|x)}\right)}{\exp\left(\beta \log \frac{\pi_\theta(y^+|x)}{\pi_{\text{ref}}(y^+|x)}\right) + \sum_i \exp\left(\beta \log \frac{\pi_\theta(y_i^-|x)}{\pi_{\text{ref}}(y_i^-|x)}\right)} \right] \\ & = -\mathbb{E}_{(x,\mathbf{y})\sim\mathcal{D}} \left[\log \mathsf{S}_{\mathbf{y}} \left(\beta \log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\text{ref}}(y^+ \mid x)} \right) \right] \end{aligned} \tag{9}$$

$$\mathcal{L}_{\text{InfoNCE}} = -\mathbb{E}_{(x,y^+,\{y_i^-\}_{i=1}^n)\sim\mathcal{D}} \left[\log \frac{\exp\left(\frac{s(x,y^+)}{t}\right)}{\exp\left(\frac{s(x,y^+)}{t}\right) + \sum_{i=1}^{n} \exp\left(\frac{s(x,y_i^-)}{t}\right)} \right] \tag{10}$$

C Experimental Details

C.1 Training Setup.

We apply the same training recipe to both base models, Qwen2.5-7B-Instruct and Tulu3-8B-SFT. All training experiments are implemented using the open-source ms-swift framework[6] on NVIDIA A800 GPUs. Our approach employs a two-stage training pipeline, with hyperparameters detailed as follows.

Stage 1	
Initial learning rate	1×10^{-5}
Number of training epochs	3
β	0.1
Maximum sequence length	4,096
Per-GPU batch size	1

Stage 2	
Initial learning rate	1×10^{-6}
Maximum sequence length	4,096
Number of training epochs	2
Number of rollouts per iteration	16
KL divergence loss coefficient	1×10^{-3}

We use IFEval as the validation set to select the best checkpoint (Lambert et al., 2024) and train the models on STAINSTRUCT with early stopping if there is no improvement of performance on IFEval for more than 3 checkpoints.

C.2 Evaluation Setup.

Baseline Models. We compare our proposed approach against a comprehensive set of large language models (LLMs), including both large- and small-scale, closed- and open-source models. For large-scale models (GPT-4.1, Doubao 1.8, DeepSeek v3.2, and Gemini 2.5), we evaluate via official API endpoints. For small-scale models (Tulu3-8B, Llama-3.1-8B-Instruct, QwQ-32B, Crab-7B-DPO), we serve the models through vLLM 0.21.0 [7] with a temperature setting of 0.0 for high-throughput inference to ensure deterministic and reproducible outputs. All automatic evaluations are conducted using GPT-4.1 as the judge model, which has been widely adopted in recent LLM research for its high inter-annotator agreement and alignment with human preferences.

[6] https://pypi.org/project/ms-swift/
[7] https://github.com/vllm-project/vllm

D Prompts

Constraint Extraction Prompt

```
You are an expert in NLP 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
instruction, return a list of checker names
that should be applied to these constraints.

## 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+
words `<sep>` HighlightSectionChecker:
highlight at least 3 sections that have
titles in markdown format `<sep>`
ForbiddenWordsChecker: Do not use any commas.


## Example 2
**Instruction:** Analyze the sentiment of
the following review and perform sentiment
attribution. Present the results in JSON
format with fields "emotion" and "sentiment
attribution".

**Response:** FormatChecker: results must be
 in JSON format `<sep>` JSONFieldChecker:
JSON must contain fields "emotion" and "
sentiment attribution".
---
## [Instruction]
### Your task:
- Generate checker names and constraint
descriptions: Extract and generate
appropriate checker names and their
corresponding constraint descriptions from
the instruction.
- Use `<sep>` to separate outputs: Checker
names and descriptions must be separated by
`<sep>`.
- Only focus on explicitly stated
constraints: Only process constraints
clearly written in the instruction.
- Ensure constraint descriptions are
complete and scope is clearly defined: For
example, if a 300-word requirement is
involved, clearly specify whether the limit
applies to the entire text or a specific
part.
- Do not generate checkers for the task
itself or its quality: Only check
constraints, not the core task content or
its completion quality.
- Keep language consistent: If the
instruction is in Chinese/English, output
the constraint descriptions in the
corresponding language.
```

- Each checker is responsible for exactly one constraint: Strictly follow the principle of "one checker per constraint".
- Do not output constraints not present in the instruction: Only process constraints explicitly given in the instruction.
- If there are no constraints, output only `<null>`: If the instruction contains no constraints, output only `<null>`.

Hard/Soft Constraint Classification Prompt

```
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
****#Instruction:**
Task: Generate a short summary of the text
in at most 32 words. Additionally, I have a
specific constraint for your output: in your
 entire response, please refrain from the
use of any commas.

Text: The Swiss will turn 35 during the Rio
Games, where he will play in the singles and
 mixed doubles, although he has not decided
on the men's doubles. [...]

****#Checker:** NumberOfWordsChecker: summary
 must be in at most 32 words
****#response:**
```python
def check_following(response):
    words = response.split()
    return len(words) <= 32
```

Requirements
- The function should be self-contained with
 necessary imports.
- Do NOT use `nltk`.
- Only return exactly Python code, without
any other information.
- The code only judges the constraint in the
 Checker, not other constraints in the
instruction.
- All code must be inside the `
check_following` function.
- The function name must be `check_following
`.

****#Instruction:** [instruct]
****#Checker:** [check]
****#response:**
```

Soft Constraint Removal Prompt

```
## Role
You are an efficient data processing expert
specializing in simplifying instruction
constraints.

## Task
Analyze the instruction below, which
consists of a "core task" and multiple "
constraints". Your task is to:
1. Identify and retain the core task.
2. Remove the specified constraints while
keeping all other constraints unchanged.
3. Strictly maintain the same language as
the original instruction: if the original is
 in Chinese, output in Chinese; if in
English or another language, output in that
language.
4. Preserve the original format (core task
first, followed by remaining constraints in
list form).

## Restrictions
- Output only the simplified instruction,
without any explanation or additional
commentary.
---
**#Instruction:** {text}
**#Constraints to remove:** {constraints}
**#Simplified instruction:**
```

Soft Constraint Negation Prompt

```
## Role
You are an efficient data processing expert
specializing in constructing contrastive
instructions through "constraint negation"
while preserving the core task objective.

## Task
Analyze the instruction below, which
contains a "core task" and "constraints".
Based on the provided constraint, generate a
 negated instruction according to the
following rules:
1. Preserve the core task: Strictly retain
the original intent and goal of the core
task without altering its essence.
2. Precisely negate the constraint: Identify
 and reverse the modifying constraints, for
example:
   - Tone: "formal" -> "casual and
colloquial"
   - Length: "brief" -> "detailed and
lengthy"
   - Audience: "target users are adults" ->
"target users are children"
3. Logical consistency: The negated
requirement must be logically executable and
 free of contradictions.
4. Language consistency (highest priority):
The output language must match the input
instruction exactly.
   - If the input is in Chinese, the output
must be in Chinese.
   - If the input is in English, the output
must be in English.
   - Do not translate the instruction into
another language.
5. Format consistency: The core task comes
first, followed by the negated constraints
```

```
in list form.

## Example
**Input:** Please summarize this article.
Constraint: use a formal tone.
**Output:** Please summarize this article.
Constraint: use a humorous and casual tone.
---
**#Instruction:** {text}
**#Constraint:** {constraints}
**#Negated instruction:**
```

Code Generation Prompt

```
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
**##Instruction:**
Task: Generate a short summary of the text
in at most 32 words. Additionally, I have a
specific constraint for your output: in your
 entire response, please refrain from the
use of any commas.

Text: The Swiss will turn 35 during the Rio
Games, where he will play in the singles and
 mixed doubles, although he has not decided
on the men's doubles. [...]

**##Checker:** summary must be in at most 32
 words
**##response:**
```python
def check_following(response):
    words = response.split()
    return len(words) <= 32
```

Requirements
- The function should be self-contained with
 necessary imports.
- Do NOT use `nltk`.
- Only return exactly Python code, without
any other information.
- The code only judges the constraint in the
 Checker, not other constraints in the
instruction.
- All code must be inside the `
check_following` function.
- The function name must be `check_following
`.

Please generate code based on the given
instruction and checker.

##Instruction: [instruct]
##Checker: [check]
##response:
```