CARE-STaR: Constraint-aware Self-taught Reasoner

Zhiliang Li[1,2]*, Bo Tang[3,4]*, Yijun Niu[4], Beihong Jin[1,2]†, Qiwen Shi[1,2], Yuchen Feng[4], Zhiyu Li[4], Jie Hu[5], Mingchuan Yang[5], Feiyu Xiong[4]
[1]Institute of Software, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]University of Science and Technology of China, Suzhou Institute for Advanced Research, Suzhou, China
[4]MemTensor (Shanghai) Technology Co., Ltd, Shanghai, China
[5]China Telecom Corporation Limited Beijing Research Institute, Beijing, China

Abstract

In real-world applications, large language models (LLMs) often need to handle diverse and complex instructions. Specifically, when instructions are subject to multiple constraints, some of which are somewhat ambiguous, LLMs often fail to produce answers that satisfy all constraints, limiting their effectiveness in various tasks. To address this challenge, we examine the different constraints in the instructions and discover that the difficulty of determining whether an answer meets a constraint varies widely, from extremely straightforward to exceptionally perplexing. Correspondingly, we propose to assign constraints to different constraint levels. Furthermore, inspired by chain-of-thought (CoT) and self-taught reasoner (STaR), we propose a two-stage method named CARE-STaR (Constraint-AwaRE STaR). Our method distinguishes constraints within instructions by generating different CoTs and guides LLMs to autonomously learn optimal answers by setting positive rewards for the CoTs that are beneficial to generating accurate answers and iteratively optimizing these answers. We have conducted extensive experiments on three instruction-following benchmarks, taking three existing LLMs as base LLMs, respectively. Experimental results indicate that our method substantially enhances the capability of these LLMs to handle complex instructions, outperforming supervised fine-tuning (SFT). Our code is available at https://github.com/lzl0124/carestar.

1 Introduction

In recent years, large language models (LLMs), which have been used in a wide range of applications, such as dialogue systems, text summarization, and machine translation (Achiam et al., 2024), have achieved significant success. As LLMs are applied to an expanding range of domains, the instructions that users input into these LLMs have become increasingly complex and diverse. In real-world applications, users expect models to handle intricate domain-specific tasks and adapt to evolving scenarios. They impose multiple constraints on instructions to optimize outputs and ensure compliance with specific measurements. The increasing complexity of instructions poses significant challenges to the understanding capabilities of LLMs. Enhancing the capability of LLMs to deal with complex instructions remains an important yet under-explored research topic.

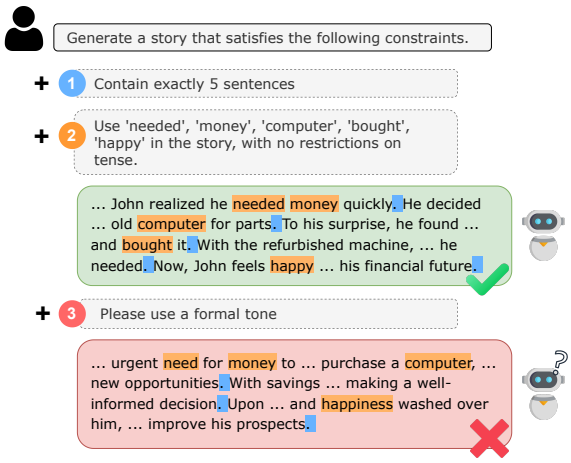

Figure 1: An example to illustrate the instruction-following capability of the LLM, indicating its shortcoming. The answer generated by the LLM meets constraints (1) and (2) when only these are given in the instruction, but fails when constraint (3) is added.

Recent research on instruction following (Xu et al., 2024; Sun et al., 2024; He et al., 2024a) has highlighted the importance of diverse and high-quality training data, which can be divided into two categories in terms of training strategy. One is to apply Supervised Fine-Tuning (SFT) (Ouyang et al., 2022) to complex instructions, while the other is to employ Direct Preference Optimization (DPO) (Rafailov et al., 2024). The primary differ-

*Equal contribution.
†Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2025, pages 21689–21703
July 27 - August 1, 2025

ences lie in both the training data format and the optimization objective (i.e., loss function). However, both kinds of methods face the same issue: they only capture direct associations between instruction and response, lacking a nuanced understanding of instruction. This lack of understanding is particularly pronounced when dealing with multi-constraint instructions. As shown in Figure 1, we have found that when an additional constraint is added to an instruction in an attempt to generate more accurate answers, the LLM fails to provide satisfactory answers. We argue that the reasons behind this are twofold. ① **Due to the inherent ambiguity of natural language**, certain constraints cannot be expressed precisely, and this vagueness can make it difficult for the model to correctly understand the task objective, leading to outputs that deviate from user expectations. Balancing such constraints presents a challenge to the model's *comprehension capabilities*. ② **Due to the complexity of the task**, the model sometimes cannot satisfy multiple constraints based solely on limited prior knowledge, especially when there are interdependencies between the constraints. Generating a response that satisfies a high proportion of constraints poses a challenge to the model's *constraint-handling capabilities*.

In this paper, we first examine the different constraints in the instructions. Existing instruction-following methods typically treat all constraints equally, which results in a lack of fine-grained optimization when the model handles constraints. However, we discover that some constraints are "vague" and often involve factors such as language style and creative expression. These constraints can be adjusted within a certain range and do not need to be followed rigidly in every detail. We call these constraints **Soft Constraints**. In contrast, some constraints are "precise" and typically can be checked by quantitative metrics, with evaluation results categorized as either "satisfied" or "not satisfied". We call them **Hard Constraints**. For instance, constraints related to length or format are typically hard constraints that must be strictly adhered to. Meanwhile, there are numerous constraints, and the difficulty of assessing whether they are satisfied lies somewhere on the spectrum between soft and hard constraints. This distinction creates new optimization space for LLMs, enabling the optimized LLMs to handle soft constraints more flexibly while ensuring adherence to hard constraints, ultimately producing responses that balance precision with creativity.

To echo this line of thought, we formulate the instruction-following task in LLMs to stipulate the optimization goal of LLMs, while also revealing the role of constraints. Subsequently, we propose to assign constraints to different constraint levels by the difficulty level of satisfying the different constraints, and put forward a two-stage method named CARE-STaR (Constraint-AwaRE Self-Taught Reasoner) to help LLMs strengthen the understanding of instructions and improve the capability to handle constraints, aiming to enable the models to provide answers that satisfy constraints well in different scenarios, thus enhancing the user experience.

CARE-STaR is inspired by the chain-of-thought (CoT), which has been successful in helping models deal with mathematical problems or reasoning tasks (Wei et al., 2022; Nye et al., 2021; Kojima et al., 2022; Rajani et al., 2019; Shwartz et al., 2020). CARE-STaR utilizes CoT to guide an LLM in distinguishing constraint levels, as well as analyzing these constraints in detail to enhance the capability of the model to handle them effectively. Meanwhile, CARE-STaR is also an expansion of Quiet-STaR (Zelikman et al., 2024). In particular, our method first employs GPT-4 (Achiam et al., 2024) for a cold start in CoT generation, improving training stability. It then incorporates a reinforcement algorithm to optimize parameters, enabling the model to refine its CoT generation strategy and achieve more precise constraint analysis while better assisting in answering questions. This ultimately leads to answers with higher constraint satisfaction.

Our contributions can be summarized as follows.

- We formulate the instruction-following task within the framework of the fuzzy set. This formulation is flexible enough to allow setting different constraint levels for constraints, thus supporting fine-grained optimization of instruction following.
- We adopt two stages for improving instruction following: first employ a few labeled CoT exemplars to warm up the LLM, and then design a reinforcement algorithm to autonomously optimize the generation of CoTs that best contribute to instruction following.
- We conduct three instruction-following benchmarks. The results on the three LLMs show that each of them, when trained using our

method, exhibits greater performance improvement compared to the same LLM trained with SFT.

2 Formulation of Instruction Following

In this section, we borrow the concept of membership function from fuzzy theory (Fullér et al., 1998) to formulate the instruction-following task in LLMs, taking into consideration that the optimization of LLM has similarities with the fuzzy control process.

A multi-constraint instruction refers to an instruction issued by a user to LLM that incorporates multiple constraints, all of which should be satisfied simultaneously when generating an answer. Let $I = \{q, c_1, c_2, ..., c_m\}$ denote the multi-constraint instruction, where q denotes the question, c_i denotes the i-th constraint and m is the total number of constraints. Let the set X be the set of all possible answers generated by an LLM for a given instruction I. We define a fuzzy set F_{c_i} over X, in which each answer $a \in X$ is assigned a membership function $\mu_c(a, c_i) \in [0, 1]$ reflecting the degree to which it satisfies the constraint c_i. The larger the value of the membership function $\mu_c(a, c_i)$, the more a belongs to that fuzzy set F_{c_i}. Similarly, we introduce the membership function $\mu_q(a, q) \in [0, 1]$ for question q in the instruction I. Thus, the ideal goal pursued by the LLM is that the answer a to the instruction $I = \{q, c_1, ..., c_m\}$ satisfies the following equation:

$$\mu_q(a, q) + \sum_{i=1}^{m} \mu_c(a, c_i) = 1 + m \qquad (1)$$

However, it is often hard for LLM to achieve this objective. As a result, the instruction-following task is proposed to train the LLM so that it satisfies the following requirement:

$$max(\mu_q(a, q) + \sum_{i=1}^{m} \mu_c(a, c_i)) \qquad (2)$$

Notably, the roles of $\mu_q(a, q)$ and $\mu_c(a, c_i)$ can be undertaken by any SOTA LLM.

In the light of this definition, we set the fuzziness of constraint for each constraint and a constraint level function $L(c_i)$ to quantify its degree of hardness or softness. A lower value indicates that c_i is a highly fuzzy constraint, which means that it allows for flexible interpretation, while a higher value signifies that c_i is highly precise, requiring strict adherence. For simplicity, the range of L is currently set to $L(c_i) \in \{1, 2, 3, 4, 5\}$. c_i is identified as a hard constraint if $L(c_i) = 5$, or as a soft constraint if $L(c_i) = 1$.

In our implementation, we employ chain-of-thought (CoT) to perform "constraint analysis", which adopts the LLM to be trained to conduct the $L(c_i)$ evaluation. Furthermore, we design three loss functions $Loss_{talk}$, $Loss_{talk\_cot}$, and $Loss_{think}$ as described in the next section to autonomously optimize the generation of CoTs that most effectively enhance instruction following.

3 Method

We now describe our method, which aims to teach LLMs to effectively follow instructions and obtain satisfactory answers. Our method consists of two stages, i.e., the initial cold-start stage, followed by the self-taught stage.

In the first stage, we warm up the LLM to be trained to improve its generation capability of high-quality CoTs. In the second stage, whose process is shown in Figure 2, we first guide the model to generate multiple candidate CoTs for each instruction in the training dataset of stage 2. Then, we evaluate the quality of each CoT based on its impact on the answer prediction. Finally, we adjust the probabilities of generating CoTs according to the evaluation results. We iterate the process above to automate the generation of "constraints analysis" until finishing the designated number (defaulted to three) of epochs.

To be noted, we can adopt any typical instruction-tuning dataset $\mathcal{D} = \{x^{(i)}, y^{(i)}\}_{i=1}^{N}$ as our training dataset, which is not required to contain CoTs. Here, $x^{(i)}$ denotes an instruction and $y^{(i)}$ denotes an answer. The dataset is randomly split into two disjoint subsets, which are used for the two stages of the method, respectively.

3.1 Cold Start

In the cold-start stage, we leverage GPT-4 (Achiam et al., 2024) to generate a CoT for each input instruction in the training set of stage 1, using the prompt shown in Appendix Table 7. Then we perform SFT on the LLM to be trained using the (instruction, CoT) pairs as training exemplars, thus helping LLM learn to discern effective CoTs before it begins to generate CoTs autonomously.

This stage is necessary because we have observed that if we enter the second stage directly

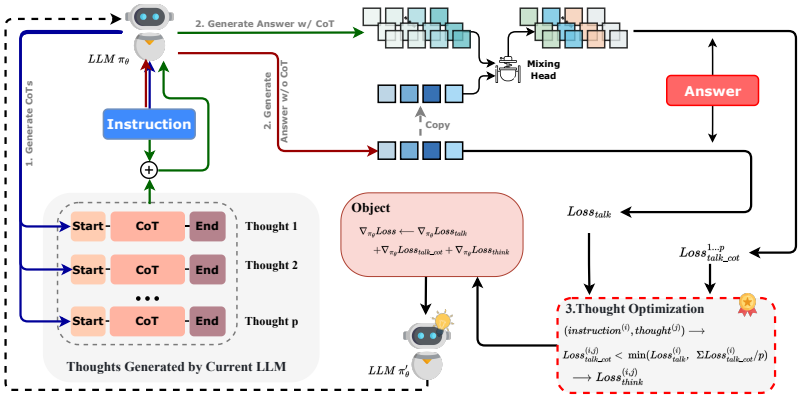

Figure 2: The framework of our self-taught stage. We first generate CoTs for each instruction in the training dataset. Then, we apply our designed reinforcement algorithm to increase the likelihood of CoTs that help the model complete the answer better. The dashed line indicates the fine-tuning outer loop. For a detailed description of the algorithmic steps, please refer to Algorithm 1 in the Appendix.

without the first stage, then the LLM often generates lengthy and suboptimal CoTs in the initial steps. These inaccurate CoTs will fail to effectively guide response generation, ultimately impacting the stability of model training. Ideally, CoTs are expected to be concise, enabling the model to produce more accurate and coherent answers through simple reasoning. To bridge the gap between the ideal and reality CoTs, we design the cold-start stage.

By first training on these exemplars, we improve the capability of the model, which enables it to produce concise and accurate CoTs in the second stage, thereby enhancing the stability and effectiveness of the overall training process.

3.2 Sampling CoTs

Stage 2, whose kernel is a reinforcement algorithm, comprises multiple training steps. At the beginning of each training step, we require the LLM to sample multiple candidate CoTs to complete the analysis of constraints in each instruction to get the corresponding constraint levels. The prompt used is also the one listed in Appendix Table 7. Specifically, for a given input instruction $x^{(i)}$, we obtain a set of CoT candidates $\mathbf{z}^{(i)} = \{z_1^{(i)}, z_2^{(i)}, ...z_p^{(i)}\}$, where p denotes the total number of sampled CoTs. Through this process, we construct an augmented dataset $\mathcal{D}' = \{x^{(i)}, y^{(i)}, \mathbf{z}^{(i)}\}_{i=1}^{N}$.

To separate the CoT from the answer, we use two learnable meta tokens (<|startthought|> and <|endthought|>) to mark the start and end positions of the CoT. <|startthought|> informs the LLM that it is in "thinking mode," while <|endthought|> indicates that the model has finished thinking and needs to provide an answer.

3.3 Learning to Answer "after Thinking"

Since we have introduced CoTs for constraint evaluation, the model has not yet developed the ability to capture the relationship between the CoT and the response. Therefore, we need to train the LLM to better utilize the CoTs for providing responses. At the same time, we do not want the answer of the LLM to overly rely on the CoT, thus losing the capability to respond directly. Hence, we divide the training of the generation of responses into two parts:

Talk directly: This part of the training objective aligns with the goal of SFT: the goal is to fine-tune a pre-trained LLM through supervised learning to generate the target answer. Specifically, SFT adopts maximum likelihood estimation (MLE) for training on the dataset $\mathcal{D}$. The training objective is to minimize the following loss function of current LLM $\pi_{\boldsymbol{\theta}}$:

$$Loss_{talk} = -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}}\left[\log \pi_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})\right] \quad (3)$$

Talk with thoughts: Inspired by the attention mechanism and Quiet-STaR, we adopt a **Mixing Head** (i.e., a 3-layer MLP) to output a weight value for each token of the answer, to decide the degree of reliance on CoTs or direct response. Specifically, in the initial few steps of the training, we set the last layer of **Mixing Head** to zero, so that in the early steps of training, the model primarily utilizes the direct response (without CoTs). In the later steps

of training, the model adjusts the weight values according to the following rule. If the CoT helps the prediction of a certain token of the answer, the corresponding weight value increases, enhancing the reliance on CoT; otherwise, it decreases, promoting a more direct response.

Given an instruction $x^{(i)}$, the corresponding answer $y^{(i)}$, and the j-th candidate CoT $z_j^{(i)}$, we can get the weighting values $w^{(i,j)}$ and compute the log-likelihood of the response conditioned on the j-th CoT and the response without any CoT, denoted as $\ell^{(i,j)}_{w/\text{ CoT}}$ and $\ell^{(i)}_{w/o\text{ CoT}}$, respectively. At position k, the weighting value and log-likelihood are given by:

$$w^{(i,j,k)} = \text{MH}\begin{pmatrix}\text{h\_sts}_\theta([x^{(i)}; z_j^{(i)}; y^{(i)}_{:k-1}]),\\ \text{h\_sts}_\theta([x^{(i)}; y^{(i)}_{:k-1}])\end{pmatrix} \quad (4)$$

$$\ell^{(i,j,k)}_{w/\text{CoT}} = \log p_\theta(y_k^{(i)} \mid x^{(i)}, z_j^{(i)}, y^{(i)}_{:k-1}) \quad (5)$$

$$\ell^{(i,k)}_{w/o\text{CoT}} = \log p_\theta(y_k^{(i)} \mid x^{(i)}, y^{(i)}_{:k-1}) \quad (6)$$

where $MH(\cdot)$ denotes **Mixing Head** and $\text{h\_sts}_\theta(\cdot)$ denotes the hidden states output by the last layer of the LLM.

Then we can compute a weighted sum at the position k, yielding the final log-likelihood.

$$\ell^{(i,j,k)}_{final} = w^{(i,j,k)} \cdot \ell^{(i,j,k)}_{w/\text{CoT}} + (1 - w^{(i,j,k)}) \cdot \ell^{(i,k)}_{w/o\text{CoT}} \quad (7)$$

Similarly, we can define the loss for this stage of training, following the same principle as SFT:

$$Loss_{talk\_cot} = -\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})\sim\mathcal{D}'}\left[\log \pi_\theta(\boldsymbol{y}\,|\boldsymbol{x},\boldsymbol{z})\right] \quad (8)$$

3.4 Optimizing Generation of CoTs

The key challenge in our method is to identify the most reasonable CoT that accurately completes the constraints analysis. We believe that if a CoT is sufficiently reasonable, it should lead to a higher generation probability of the gold response. Therefore, for each pair $(x^{(i)}, y^{(i)})$ in the training dataset $\mathcal{D}$, we aim to find a CoT $\tilde{z^{(i)}}$ such that $\pi_\theta(y^{(i)}|x^{(i)}, \tilde{z^{(i)}})$ is maximized.

Following the process outlined above, for the input $x^{(i)}$, we can obtain $Loss^{(i)}_{talk}$ and $\{Loss^{(i,j)}_{talk\_cot} | j = 1, 2, \ldots, p\}$. If $Loss^{(i,j)}_{talk\_cot}$ is lower, it indicates that the j-th CoT is more helpful; otherwise, the CoT is less helpful. However, due to the limited number of sampled CoTs, we often cannot obtain the optimal one. Therefore, we use $Loss_{talk}$ as a reference to filter the CoTs. For the j-th CoT of the i-th instruction, we define the reward $r^{(i,j)}$ as follows:

$$r^{(i,j)} = max(0, Loss^{(i)}_{talk} - Loss^{(i,j)}_{talk\_cot}) \quad (9)$$

The reward signifies the improvement in the prediction of the answer when a CoT is involved. Following the methods in TRICE (Hoffman et al., 2024) and Quiet-STaR, we subtract the mean reward of the p sampled CoTs from the original reward of each CoT, and remove any negative values to improve training stability:

$$r^{(i,j)} = \max(0, r^{(i,j)} - \overline{r^{(i)}}) \quad (10)$$

We then use this reward to increase the probability of generating CoTs that perform better than the average:

$$Loss^{(i,j)}_{think} = -r^{(i,j)} \cdot \log\, \pi_\theta(z^{(i,j)}|x^{(i)}) \quad (11)$$

Through the iterative learning process, the model will gradually learn to generate CoTs, which not only accomplishes the task of constraint analysis but also enhances the response to ensure compliance with the constraints.

4 Experiments

4.1 Experimental Setup

We conduct experiments on three popular base LLMs: **Mistral-7B-Instruct-v0.3** (Jiang et al., 2023a), **LLaMA-3.2-3B-Instruct** (Dubey et al., 2024), and **Qwen2.5-7B-Instruct** (Yang et al., 2024a). For brevity, we omit the suffix 'Instruct' from all model names below.

We employ WizardLM (Xu et al., 2024) as the training dataset. The reason for this is that WizardLM is a resultant dataset of executing the Evol-Instruct approach that allows for incrementally generating more complex instructions by taking initial seed instructions as input. This gradual generation of increasingly sophisticated instructions makes WizardLM a suitable choice for our training needs.

Apart from adopting our method to train the base models, we also adopt SFT to train the base models as baselines, since both share the same training data

Models		Change Case	Combination	Content	Format	Keywords	Language	Length	Punctuation	StartEnd	P-Level	I-Level
Mistral-7B-v0.3	Base Model	0.6742	0.2615	**0.8679**	0.8471	**0.7117**	0.7742	0.5035	0.2121	0.8358	0.5268	0.6451
	SFT	0.7303	0.5846	0.8113	0.8089	0.6564	0.7742	0.5315	0.2424	**0.8507**	0.5600	0.6630
	Ours	**0.7753**	**0.7538**	**0.8679**	**0.8918**	0.7055	**0.8387**	**0.5664**	**0.2425**	0.7910	**0.6248**	**0.7134**
Llama-3.2-3B	Base Model	0.7753	0.3846	0.8679	0.8280	0.8037	0.9032	**0.7552**	**0.9848**	0.8358	0.6968	0.7889
	SFT	0.7528	**0.6154**	0.7736	0.8599	0.7791	**0.9355**	0.6783	0.8636	**0.8955**	0.7024	0.7830
	Ours	**0.8090**	0.4615	**0.8868**	**0.8790**	**0.8221**	0.9032	0.7343	0.8030	0.8657	**0.7246**	**0.7974**
Qwen2.5-7B	Base Model	0.7640	0.6769	0.8302	0.8854	**0.7975**	**1.0000**	0.6224	**0.9697**	**0.9403**	0.7320	0.8058
	SFT	0.7640	0.6769	**0.9433**	**0.9235**	0.7791	0.9032	0.6434	0.8333	**0.9403**	0.7338	0.8058
	Ours	**0.8202**	**0.7692**	0.8491	0.9172	0.7423	**1.0000**	**0.6573**	0.8636	0.9104	**0.7579**	**0.8106**

Table 1: Overall performance on IFeval.

format. For completeness, we also compare our method with DPO, and the corresponding results are presented in Appendix A.4.

For evaluation, we choose the following three challenging instruction-following benchmarks, which contain complex constraints and allow us to observe the effect of thoughts on instruction-following capability.

- IFeval (Zhou et al., 2023) is designed to evaluate a series of "verifiable constraints" without relying on manual evaluation or LLM-based assessments.
- FollowBench (Jiang et al., 2023b) aims to evaluate the model performance across multiple types of scenarios (including content, situation, format, and mixed) by progressively adding constraints.
- CELLO (He et al., 2024b) designs eight scenarios (such as QA, planning, summarization, etc.) using complex instructions to comprehensively assess the capability of LLMs to follow complex instructions.

It is worth mentioning that the instructions in the training dataset differ significantly from those in the three benchmarks. This shows that our method is agnostic to the training dataset.

4.2 Main Results

Tables 1 and 2 list results on three instruction-following benchmarks.

From these tables, we find that the models trained using our method perform excellently regardless of the benchmark. Taking the IFeval benchmark as an example, the models trained using our method are, on average, 8.71% higher in terms of P-Level metrics and 4.09% higher in terms of I-Level compared to the corresponding original models. On average, the models trained using our method are 6% higher on the P-Level metric and 3.34% higher on the I-Level metric, compared to the corresponding models trained using SFT. In particular, of the three base models, Mistral-7B-v0.3 can obtain the maximum performance improvement, compared to the other base models. Additionally, we found that our method achieves limited performance gains in Qwen2.5-7B. To investigate the underlying cause of this phenomenon, we conduct additional experiments, whose details are provided in Appendix A.5.

Models		Followbench			CELLO
		CSL	HSR	SSR	Avg
Mistral-7B-v0.3	Base Model	2.550	0.576	0.677	0.632
	SFT	2.450	0.594	0.667	0.634
	Ours	**2.675**	**0.625**	**0.692**	**0.714**
Llama-3.2-3B	Base Model	2.300	0.569	0.658	0.633
	SFT	2.625	0.612	**0.704**	0.647
	Ours	**2.725**	**0.622**	0.699	**0.676**
Qwen2.5-7B	Base Model	2.900	0.672	0.746	0.745
	SFT	2.800	0.650	0.737	0.751
	Ours	**2.975**	**0.699**	**0.764**	**0.761**

Table 2: Performance on Followbench and CELLO benchmarks. The full results of CELLO are provided in Table 6 of the Appendix.

4.3 Ablation Study

Ablation study is carried out to investigate the impact of the soft and hard constraint concepts introduced in Section 2, the cold-start stage and the self-taught iteration proposed in Section 3. Here, given a base model, we adopt its base version, constraint version, cold-start version, and self-taught version. For more details on the definitions of these versions, see Appendix A.2.

The results, summarized in Figure 3, reveal that:

- **The introduction of distinction between constraints has improved the constraint handling capability of LLMs**. For example, the constraint version of Mistral-7B results in a 6.28% improvement on IFeval, demonstrating that the model can more effectively handle constraints in this benchmark, such as format-

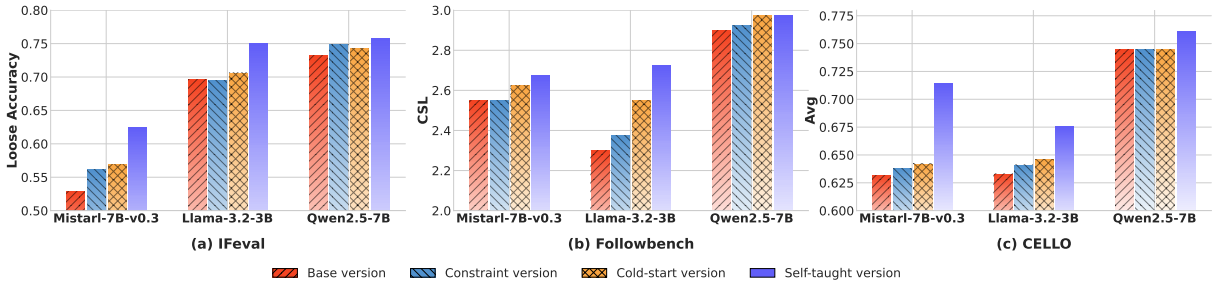

Figure 3: Ablation study.

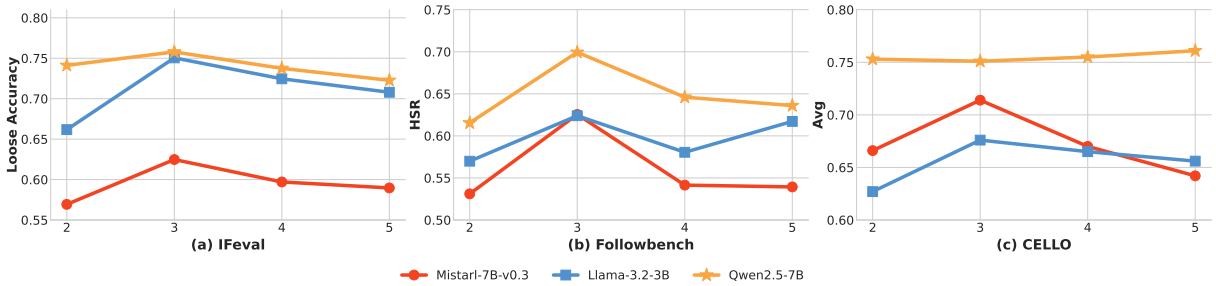

Figure 4: Effect of number of sampled thoughts on performance.

ting and word counting, which are treated as hard constraints. This result is particularly promising, because no additional knowledge is introduced to the base model during this process. Instead, simply prompting LLM to be aware of the differences in constraints within the instruction appreciably improves the extent to which the answers given by the LLM conform to the different constraints.

- **After the cold-start stage, LLMs gain the capability to generate beneficial CoTs**, which is helpful for the subsequent self-taught stage. Taking Llama-3.2-3B as an example, it results in a 7.4% improvement over the constraint version and a 10.9% improvement compared to the base version on Followbench. The cold-start version always outperforms the constraint version, as the LLM begins to analyze the constraints rather than merely pondering them.

- **After the self-taught stage, LLMs demonstrate further improvements in instruction-following tasks**, validating the effectiveness of our reinforcement algorithm. Taking Mistral-7B-v0.3 as an example, it achieves a 9.7% improvement on IFeval, an 11.2% improvement on CELLO, and a 1.9% improvement on Followbench compared to the model that finishes the cold-start stage of training.

In sum, incrementally adding training stages proposed in our method into LLMs can help LLMs progressively enhance their handling of constraints.

4.4 Effects of Number of Sampled Thoughts

We investigate the impact of the number of CoTs sampled during training on performance. As Figure 4 shows, we observe that for models with 3B or 7B parameters, setting the number to 3 usually yields the best results. Specifically, when the number is set to 2, the model often struggles to sample the optimal CoT. Conversely, when the number exceeds 3, the limited capability of the model increases the likelihood of introducing noise to our self-taught stage, which in turn compromises training stability. Additionally, we note that for different base models, those with higher performance tend to exhibit greater training stability. For instance, in the case of Qwen, we observe that as the number of CoTs increases, there is only a slight improvement in performance on the CELLO benchmark.

4.5 Case Study

Figure 5 presents an example of an instruction that the original model fails to answer correctly but the model after training with our method successfully resolves. This instruction includes three constraints, where the "formal tone" constraint is more flexible and can influence the fulfillment of the other constraints, as a formal tone often requires

Example:

Instruction:
Using words in the 'Input' to generate a story, with no restrictions on tense. The story should contain exactly 5 sentences. Please using a formal tone.

Input:
needed, money, computer, bought, happy

Output:
<|startthought|>
- Generate a story using words from the input: 5
- The story should contain exactly 5 sentences: 5
- Use a formal tone: 4
...
<|endthought|>
John realized he needed a computer to ... in a few days. Despite his tight budget, ... for the job. With determination, he ... at his part-time job. Finally, he had enough money to buy ... eyeing. Overwhelmed with happiness, John felt a sense of relief and excitement... new computer home.

Figure 5: Example of an LLM trained with our method to answer an instruction with multiple constraints.

adherence to specific norms, which may impact word choice or sentence structure.

From the response, we find that these constraints are rated accordingly. The constraints on "sentence counting" and "word choice" are given high scores, while the "formal tone" constraint is rated a secondary score. The answer meets all the hard constraints and incorporates fewer colloquial expressions, such as "a sense of relief and excitement," to fulfill the "formal tone" requirement. More details are shown in Appendix A.3.

5 Related Work

5.1 Complex Instruction Tuning

Existing research on improving the capability of LLMs to handle complex instructions concentrates on the generation of different complex instructions. These generated instructions are then used to fine-tune the original pre-trained models. For example, Xu et al. (2024) proposed Evol-Instruct to rewrite seed instructions step by step into more complex instructions, aiming to enhance the overall capability of LLMs for tasks with varying complexity. Yang et al. (2024b) introduced an effective data augmentation technique that breaks down complex instructions into simpler sub-instructions, and modifies and reconstructs them to form new instructions, enhancing the capability of LLMs to detect subtle variations.

When fine-grained constraints are gradually added to the instruction, LLMs usually struggle to meet them (Sun et al., 2023). He et al. (2024a) found that training LLMs with instructions containing multiple constraints can enhance their understanding of complex instructions and proposed a "discriminative" approach to acquiring data with complex constraints. Conifer (Sun et al., 2024) proposes a progressive learning method by fine-tuning the model using instructions with increasing numbers of constraints to improve its performance on complex constraints. However, these methods do not consider the differences between constraints, which makes it difficult for the model to handle them with fine-grained processing.

5.2 Training to Think

LLMs exhibit enhanced performance on reasoning tasks when they explicitly write their reasoning steps first (Wei et al., 2022). However, trying to equip LLMs with such capabilities often requires the construction of massive reasoning datasets. Consequently, recent research exploring self-improvement techniques to enhance LLMs' reasoning capabilities has garnered our interest. STaR (Zelikman et al., 2022) iteratively enhances the model's reasoning capability through the following process: using few-shot prompting to have the model generate both thoughts and answers from the training data, and then filtering the thoughts based on the correctness of the answers for SFT. Its extended method, V-STaR (Hosseini et al., 2024), trains a verifier to assess the correctness of the CoTs using DPO, by leveraging both correct and incorrect CoTs generated during the iterative process, and utilizes the verifier to select the correct CoT during the inference phase. Quiet-STaR (Zelikman et al., 2024) aims to teach LLMs to generate a thought segment after each token to explain the future text, thereby improving predictions for the next token. However, these methods have only shown improvements in mathematical and reasoning tasks and have not been applied to the instruction-following tasks. Wu et al. (2024) argue that CoTs can be applied to any task and propose an algorithm called "TPO", which iteratively searches and optimizes the process of thought generation, allowing the model to learn how to think independently. However, this approach does not account for complex constraints.

6 Conclusion

In this paper, we propose the CARE-STaR method, which trains LLMs in two stages to push the upper limit of the capability of the original LLMs to follow instructions. Our method adopts a self-taught mechanism to tackle multiple constraints in the instruction, eliminating the need for any specific instruction dataset during training. Experimental results show that our method outperforms SFT in handling complex instructions.

7 Limitations

CARE-STaR enhances the performance of LLMs in instruction-following tasks by introducing CoTs to guide the differentiation of constraints within instructions. However, generating multiple CoTs for each instruction in the training set incurs significant computational costs. Future work could optimize this process to improve its efficiency. Additionally, we have observed that our method results in only limited improvements for Qwen2.5-7B. One possible explanation is that the model already demonstrates strong performance, reducing the impact of additional reasoning steps. Furthermore, the limited relevance between our training dataset and the test set may have also played a negative role in these results.

Besides, our experiments were conducted only on models of sizes 3B and 7B. Whether this method remains effective for larger models remains an open question for future exploration.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No. 62072450.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2024. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Robert Fullér et al. 1998. *Fuzzy reasoning and fuzzy optimization*. 9. Turku Centre for Computer Science Abo.

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. *arXiv preprint arXiv:2404.15846*.

Qianyu He, Jie Zeng, Wenhao Huang, Lina Chen, Jin Xiao, Qianxi He, Xunzhe Zhou, Jiaqing Liang, and Yanghua Xiao. 2024b. Can large language models understand real-world complex instructions? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18188–18196.

Matthew Douglas Hoffman, Du Phan, David Dohan, Sholto Douglas, Tuan Anh Le, Aaron Parisi, Pavel Sountsov, Charles Sutton, Sharad Vikram, and Rif A Saurous. 2024. Training chain-of-thought via latent-variable inference. *Advances in Neural Information Processing Systems*, 36.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023a. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2023b. Followbench: A multi-level fine-grained constraints following benchmark for large language models. *arXiv preprint arXiv:2310.20410*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. 2019. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*.

Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. Unsupervised commonsense question answering with self-talk. *arXiv preprint arXiv:2004.05483*.

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*.

Jiao Sun, Yufei Tian, Wangchunshu Zhou, Nan Xu, Qian Hu, Rahul Gupta, John Wieting, Nanyun Peng, and Xuezhe Ma. 2023. Evaluating large language models on controlled generation tasks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3155–3168, Singapore. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tianhao Wu, Janice Lan, Weizhe Yuan, Jiantao Jiao, Jason Weston, and Sainbayar Sukhbaatar. 2024. Thinking llms: General instruction following with thought generation. *arXiv preprint arXiv:2410.10630*.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.

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

Jiuding Yang, Weidong Guo, Kaitong Yang, Xiangyang Li, Zhuwei Rao, Yu Xu, and Di Niu. 2024b. Optimizing and testing instruction-following: Analyzing the impact of fine-grained instruction variants on instruction-tuned llms. *arXiv preprint arXiv:2406.11301*.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

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 Appendix

A.1 Prompt for Generating Thoughts

By incorporating a chain of thought, we can prompt the model to provide additional steps before answering, which helps it perform better. For instance, Zero-Shot Reasoner (Kojima et al., 2022) improved the performance of the model in various reasoning tasks by simply adding the phrase "Let's think step by step". Our method optimizes the content of the chain of thought specifically for instruction-following tasks. We prompt the model to first evaluate the constraints within the instruction, and then provide a step-by-step procedure for answering the instruction based on the evaluation results. Our prompt for generating thoughts is detailed in Table 7. To further investigate the impact of reasoning order on model performance, we also conduct an additional experiment on Mistral-7B-v0.3, where the model is prompted to first generate the step-by-step procedure and then perform constraint evaluation. The results are shown in Table 4. We think that the results are consistent with the following viewpoint: When LLMs rely on rating scores in writing answer steps, this reliance is inherently logically reasonable.

A.2 Four LLM Versions for Ablation Study

In the ablation study, we investigate the necessity of each component introduced in our method by gradually adding them to the base model. Given an LLM, the ablation study adopts four versions: the base version, constraint version, cold-start version, and self-taught version.

Base version: The base version is the base model itself (i.e., **Llama-3.2-3B**, **Qwen2.5-7B**, or **Mistral-7B-v0.3**), which directly executes instructions to provide answers.

Constraint version: Constraint version provides a simple implementation of soft and hard constraints on the base model, enabling it to consider the varying nature of constraints in the instructions. In particular, we use the prompt, one example of which is shown in Table 3, to inform the model that the instruction may contain multiple constraints, which can differ from one another and be categorized into hard and soft constraints. In this way, we attempt to guide the base model to revise the way it produces the answer.

Cold-start version: Cold-start version aims to explore the quality of CoTs produced by the model trained in the cold-start stage (hereafter referred to as the cold-start model). For this purpose, we first execute the cold-start model, taking the prompt in Table 7 as input, to generate a CoT for each instruction. Then, we execute the base model by inputting the (Instruction, CoT) pair to generate a response. The quality of the generated response can serve as an indicator of the effectiveness of the CoT.

Self-taught version: This version is the model trained by our two-stage method, which generates CoTs first and then utilizes these CoTs to complete the final answer.

By comparing the results of the four versions above, we can assess the impact of each component of our method on the instruction-following task.

You are a helpful AI assistant. You will be given an instruction that describes a task. The instruction may contain multiple constraints and the constraints can be categorized into hard and soft constraints. Hard constraints are precise and quantifiable, with clear yes/no or pass/fail criteria. They must be strictly followed (e.g., word counting, specific format). Soft constraints are more flexible, allowing for some variation in their fulfillment (e.g., tone, creativity, or style). You need to write a response that appropriately completes the instruction directly. **Instruction**: %s

Table 3: An example of the prompt to handle the hard and soft constraints.

A.3 Case Study Details

We provide the complete answers to the instructions in the case study. As shown in Table 8, the base model is able to successfully complete the instruction when faced with only the constraints of "sentence counting" and "word choice". However, when the relatively vague constraint of "formal tone" is added, the model fails to meet the first two constraints. After being trained with our method, the model performs the aforementioned instruction again. As shown in Table 9, it extracts and evaluates the constraints within the instruction. Then, based on this analysis, it outlines the steps to perform the instruction. Finally, the model successfully handles the constraints.

A.4 Comparison with a DPO Baseline

We apply the code and training dataset provided by He et al. (2024a) on Mistral-7B, and then let the DPO-tuning Mistral-7B-v0.3 run the benchmarks.

The main results are listed in Table 5. From the results, we observe that although DPO tuning leads to better performance on IFeval, it underperforms on CELLO and FollowBench. Considering that the dataset used for this method is specifically optimized for IFeval (i.e., using data containing the constraints in IFeval), and that it performs unsatisfactorily on other benchmarks, we believe that training an LLM with this DPO method does not bring a stable performance increase to the LLM's capability of instruction-following.

A.5 Additional Experiments on Qwen2.5

To investigate the reason behind the limited performance gains observed in Qwen2.5-7B, we further evaluate three additional models from the Qwen2.5 family using the IFeval benchmark. We consider three types of models: the original base models, models tuned with SFT, and those tuned using our method. The benchmarking results are presented in Figure 6.

From the results, we observe the following: ① **our method consistently outperforms the SFT approach** across all tested models. Notably, SFT does not always yield performance improvements, regardless of the model size. ② Within the same model family, **our method tends to deliver greater improvements for smaller models than for larger ones**. One possible explanation is that larger models, with more parameters, already possess stronger instruction-following capabilities, leaving less room for improvement. In contrast, our method provides a more cost-efficient way to enhance performance, compared to scaling up the model size.

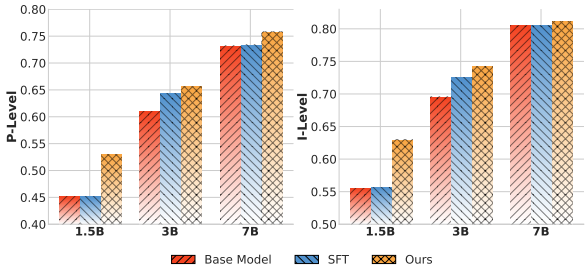

Figure 6: Effect of model sizes on performance.

A.6 Implementation Details

We conduct all experiments using four NVIDIA A100 80GB GPUs. Our training leverages DeepSpeed ZeRO Stage 2 for efficient memory management. For optimization, we employ the AdamW optimizer with a warmup of 20 steps, a weight decay of 0.001, a learning rate of 5e-7, and a batch size of 4. During training, we generate multiple CoTs for each instruction using a sampling temperature of $T = 0.9$. In contrast, for evaluation, we employ greedy decoding to generate CoTs. The maximum length for sampled CoTs is set to 200, and any exceeding this limit will be truncated.

Models	**IFeval**	**CELLO**	**FollowBench**
	P-Level	Avg	HSR
Original	**0.625**	**0.714**	**0.625**
Inverse	0.590	0.711	0.584

Table 4: Effect of reasoning order on instruction-following performance. ***Original*** *denotes the setting where the model performs constraint evaluation before giving steps to answer, while* ***Inverse*** *reverses the order.*

Models	**IFeval**		**CELLO**	**FollowBench**		
	P-Level	I-Level	Avg	CSL	HSR	SSR
Base Model	0.527	0.645	0.632	2.550	0.576	0.677
DPO	**0.647**	**0.732**	0.651	2.125	0.518	0.636
Ours	0.625	0.713	**0.714**	**2.675**	**0.625**	**0.692**

Table 5: Comparison with a DPO Baseline.

Algorithm 1 CARE-STaR

Input: Language model (lm) π_θ^0, training steps num_steps, batch_size b, num_cots p
Output: Language model $\pi_\theta^{num\_steps}$

for $i = 0$ to $num\_steps - 1$ **do**
  Sample p CoTs $\mathbf{z}_{j,1}, \ldots, \mathbf{z}_{j,p}$ for each input x_j in the batch X in parallel
  for $j = 1$ to b in parallel **do**
    $h_j^{original} \leftarrow hidden\_states_{\pi_\theta^i}([x_j; y_j])$
    $h_{j,1:p}^{thought} \leftarrow hidden\_states_{\pi_\theta^i}([x_j; z_{j,1:p}; y_j])$
    $\log p_j^{original} \leftarrow lm\_head(h_j^{original})$
    $\log p_{j,1:p}^{thought} \leftarrow lm\_head(h_{j,1:p}^{thought})$
    $w_{j,1:p} \leftarrow Mixing\_Head([h_j^{original}; h_{j,1:p}^{thought}])$
    $\log p_{j,1:p}^{final} \leftarrow w_{j,1:p} \cdot \log p_j^{original} + (1 - w_{j,1:p}) \cdot \log p_{j,1:p}^{thought}$
    $\mathcal{L}_j^{talk} \leftarrow -\log p_j^{original}(y_j)$
    $\mathcal{L}_{j,1:p}^{talk\_cot} \leftarrow -\log p_{j,1:p}^{final}(y_j)$
    $r_{j,1:p} \leftarrow \max(0, \mathcal{L}_j^{talk} - \mathcal{L}_{j,1:p}^{talk\_cot})$
    $r_{j,1:p} \leftarrow r_{j,1:p} - \overline{\mathbf{r}_j}$
    $\nabla_{\pi_\theta^i} \mathcal{L}_{j,1:p}^{think} \leftarrow -r_{j,1:p} \cdot \mathbb{I}(r_{j,1:p} > 0) \cdot \log p(z_{j,1:p}|x_j)$
    $\nabla_{\pi_\theta^i} \mathcal{L}_j \leftarrow \nabla_{\pi_\theta^i} \mathcal{L}_j^{talk} + \sum_{k=1}^{p} \left(\nabla_{\pi_\theta^i} \mathcal{L}_{j,k}^{talk\_cot} + \nabla_{\pi_\theta^i} \mathcal{L}_{j,k}^{think} \right)$
  end for
  $\pi_\theta^{i+1} \leftarrow \pi_\theta^i - \alpha \frac{1}{batch\_size} \sum_{j=1}^{batch\_size} \nabla_{\pi_\theta^i} \mathcal{L}_j$
end for

Models		Complex Task Description						Complex Input					All
		Extraction	Planning	Meta.	Writing(S)	BS(S)	Average	Keywords	QA	Sum.	Structure	Average	Average
Mistral-7B-v0.3	Base Model	0.550	**0.755**	0.623	0.718	0.758	0.681	0.396	0.517	0.641	0.781	0.584	0.632
	SFT	**0.617**	0.665	0.616	**0.785**	**0.860**	**0.709**	0.621	0.554	0.426	0.635	0.559	0.634
	Ours	0.615	0.712	**0.708**	0.760	0.748	0.708	**0.750**	**0.694**	**0.645**	**0.790**	**0.720**	**0.714**
Llama-3.2-3B	Base Model	0.568	0.519	0.444	**0.749**	0.822	0.621	0.580	0.588	0.666	**0.750**	0.646	0.633
	SFT	**0.624**	0.641	0.380	0.659	0.781	0.617	**0.673**	**0.698**	0.668	0.671	**0.677**	0.647
	Ours	0.585	**0.646**	**0.559**	0.730	**0.884**	**0.681**	0.630	0.632	**0.676**	**0.750**	0.672	**0.676**
Qwen2.5-7B	Base Model	**0.705**	0.824	0.635	**0.829**	**0.806**	**0.760**	0.695	0.708	0.761	0.755	0.730	0.745
	SFT	0.700	**0.829**	0.612	0.813	0.766	0.744	0.779	0.697	**0.791**	0.768	0.759	0.751
	Ours	0.686	0.824	**0.654**	0.807	0.785	0.751	**0.788**	**0.737**	0.783	**0.779**	**0.772**	**0.761**

Table 6: Overall performance on CELLO.

Instruction:
%s
Task:
Please analyze the provided instruction.
Write your thought process briefly after "Here is my thought process:". In the thought process, divide your response into two sections: "Requirement Evaluation" and "Steps to Answer".
1. Requirement Evaluation:
- Identify and list all requirements from the Instruction.
- Assign a rating score (1 to 5) to each requirement:
- The constraint with score 5 indicates that it should be strictly followed.
- The constraint with score 1 indicates that it can be loosely followed.
- Use concise bullet points without additional explanation.
2. Steps to Answer:
- Outline the minimal steps necessary to directly execute the Instruction.
- Do not include steps for analyzing, reasoning, or checking; limit steps to direct execution of the Instruction.
- Keep steps concise, with no more than one sentence per step.
- If formatting is involved, provide examples in an abstract way (e.g., for the requirement 'separate 2 answers by ******', you can give an example like '[ANS1]******[ANS2]').
Here is my thought process:

Table 7: Prompt for generating a CoT to complete constraint analysis.

/*Instruction with 2 constraints*/
Instruction:
Using words in the 'Input' to generate a story, with no restrictions on tense. The story should contain exactly 5 sentences.
Input:
needed, money, computer, bought, happy
Output:
After losing his job, John realized he needed money quickly. He decided to sell his old computer for parts. To his surprise, he found a working computer at a thrift store and bought it. With the refurbished machine, John was able to start freelancing, earning him the money he needed. Now, John feels happy and relieved, knowing he has secured his financial future.

/*Instruction with 3 constraints*/
Instruction:
Using words in the 'Input' to generate a story, with no restrictions on tense. The story should contain exactly 5 sentences. Please using a formal tone.
Input:
needed, money, computer, bought, happy
Output:
The individual recognized the urgent need for money to secure his family's future, and after much deliberation, he decided to purchase a computer, which he believed would open up new opportunities. With savings meticulously set aside, he visited several stores to compare models and features, ultimately making a well-informed decision. Upon returning home with his new acquisition, a sense of accomplishment and happiness washed over him, knowing this investment could significantly improve his prospects.

Table 8: Example of a base model to complete the "sentence counting", "word choice" and "formal tone" constraints.

Instruction:
Using words in the 'Input' to generate a story, with no restrictions on tense. The story should contain exactly 5 sentences. Please using a formal tone.
Input:
needed, money, computer, bought, happy
Output:
<startthought>
1. Requirement Evaluation:
- Generate a story using words from the input: 5
- The story should contain exactly 5 sentences: 5
- Use a formal tone: 4
2. Steps to Answer:
- Identify the words from the input: needed, money, computer, bought, happy.
- Create a narrative that incorporates these words in a logical sequence.
- Ensure the story consists of exactly 5 sentences.
- Maintain a formal tone throughout the story.
<endthought>
John realized he needed a computer to complete his project, which was due in a few days. Despite his tight budget, he understood the importance of having the right tools for the job. With determination, he saved every penny he could, working extra hours at his part-time job. Finally, he had enough money to buy the computer he had been eyeing. Overwhelmed with happiness, John felt a sense of relief and excitement as he carried his new computer home.

Table 9: Example of the self-taught version of the base model to handle the hard and soft constraints.