Constraint Back-translation Improves Complex Instruction Following of Large Language Models

Anonymous ACL submission

Abstract

Large language models (LLMs) struggle to follow instructions with complex constraints in format, length, etc. Following the conventional instruction-tuning practice, previous works conduct post-training on complex instruction-response pairs generated by feeding complex instructions to advanced LLMs. However, even advanced LLMs cannot follow complex instructions well, thus limiting the quality of generated data. In this work, we find that *existing datasets inherently contain implicit complex constraints* and propose a novel data generation technique, *constraint back-translation*. Specifically, we take the high-quality instruction-response pairs in existing datasets and only adopt advanced LLMs to add complex constraints already met by the responses to the instructions, which naturally reduces costs and data noise. In the experiments, we adopt Llama3-70B-Instruct to back-translate constraints and create a high-quality complex instruction-response dataset, named CRAB. We present that post-training on CRAB improves multiple backbone LLMs' complex instruction-following ability, evaluated on extensive instruction-following benchmarks. We further find that constraint back-translation also serves as a useful auxiliary training objective in post-training. Our code, data, and models will be released to facilitate future research.

1 Introduction

Large language models (LLMs) have achieved remarkable performance in numerous natural language processing tasks (Zhao et al., 2023; OpenAI, 2024; Yang et al., 2024; Dubey et al., 2024; Team et al., 2024). However, they still fall short in following instructions with complex constraints (Zhou et al., 2023; Jiang et al., 2024; Qin et al., 2024), such as length constraints shown in Figure 1, which limits their effectiveness and usability.

To enhance the instruction-following ability of LLMs, the standard practice is to post-train the targeted LLM on a large set of instruction-response data pairs. For the complex instruction-following with multiple constraints, existing efforts (Sun et al., 2024; He et al., 2024) synthesize complex datasets by adding multiple constraints to existing instructions and generating responses with advanced LLMs like GPT-4 (OpenAI, 2024). While this data generation pipeline is straightforward and widely adopted, even the most capable LLMs cannot follow complex instructions well (Jiang et al., 2024; Qin et al., 2024), which limits the quality of generated data and necessites laborious filtering. The status quo urges the development of automatic data generation methods relying less on existing LLMs' complex instruction-following abilities.

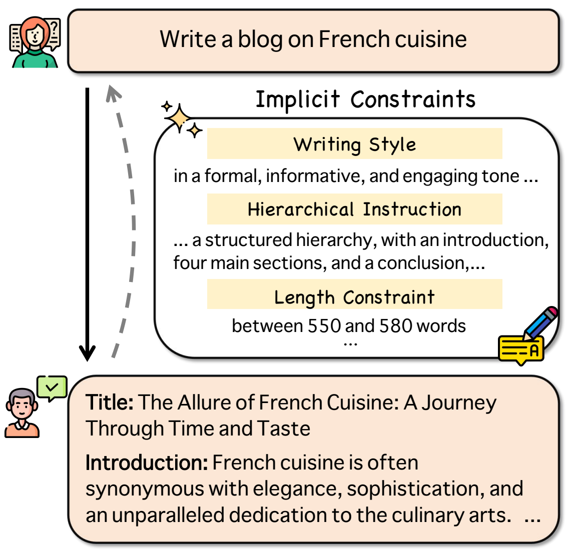

Write a blog on French cuisine
Implicit Constraints
Writing Style
in a formal, informative, and engaging tone ...
Hierarchical Instruction
... a structured hierarchy, with an introduction, four main sections, and a conclusion,...
Length Constraint
between 550 and 580 words
...
Title: The Allure of French Cuisine: A Journey Through Time and Taste
Introduction: French cuisine is often synonymous with elegance, sophistication, and an unparalleled dedication to the culinary arts. ...

Figure 1: Existing datasets inherently include implicit satisfied complex constraints in the responses.

Our key observation is that *existing datasets inherently include implicit complex constraints* so that we can reuse the widely-available high-quality instruction-following datasets (Xu et al., 2023; Taori et al., 2023; Mukherjee et al., 2023; Köpf et al., 2024) to synthesize complex instruction-response pairs. As shown in Figure 1, although the original concise instruction does not explicitly specify constraints like writing style or length, the response already satisfies some constraints in multiple dimensions. Therefore, we can efficiently create high-quality complex instruction-response pairs from existing datasets by generating constraints from responses and adding them to instructions. We dub this data generation method as *constraint back-translation*. It only requires discovering the constraints already met by responses rather than following the complex instructions with multiple constraints, which significantly reduces requirements for model capability. As a result, it is both cost-effective and capable of producing high-quality data with limited noise. We also find that constraint back-translation can serve as a useful auxiliary training objective in post-training, dubbed as the *reverse training* technique. Specifically, we use instructions and responses as inputs to train the model to output constraints in post-training. The intuition is that reverse training may enhance the model's understanding of constraints and improve its efficacy (Golovneva et al., 2024).

We adopt Llama3-70B-Instruct (Dubey et al., 2024) to back-translate constraints from a collection of existing data, generating a large-scale complex instruction-following dataset, named CRAB. Specifically, we sample a total of $13,500$ instances from existing high-quality instruction-following datasets (Peng et al., 2023; Es, 2023; Xu et al., 2023; Köpf et al., 2024) as the seed data, and manually define a scope of common constraints. We then use the original instruction, response, and constraint scope as inputs to Llama3-70B-Instruct to generate the corresponding implicitly satisfied constraints. Following previous works (Sun et al., 2024; He et al., 2024), we train the LLMs using the mixture of CRAB and ShareGPT dataset (Chiang et al., 2023), and we jointly adopt standard supervised fine-tuning and reverse-training on CRAB. In the experiments, we select the capable open-source LLMs Llama3 8B (Dubey et al., 2024) and Mistral 7B (Jiang et al., 2023) as backbone models and evaluate the complex instruction-following abilities of our models against various baselines on IFEval (Zhou et al., 2023) and FollowBench (Jiang et al., 2024). The results demonstrate that training on CRAB significantly enhances LLM performance in complex instruction following. We also conduct evaluation for general instruction-following abilities on AlpacaEval (Li et al., 2023b) and find that our models achieve even larger improvements to previous works focusing on enhancing complex instruction-following abilities like Conifer (Sun et al., 2024). This indicates that constraint back-translation yields higher general data quality than previous techniques relying on the ability of advanced LLMs. Ablation studies further validate the efficacy of our CRAB dataset and reverse training approach. Finally, we discuss the advantages and challenging scenarios for our constraint back-translation method with experiments

In summary, our contributions are threefold: (1) We propose constraint back-translation, a cost-effective and high-quality data construction method for complex instruction following. (2) We construct CRAB, a high-quality complex instruction-following dataset, and design a reverse training method for developing Llama3$_{\text{CRAB}}$, Mistral$_{\text{CRAB}}$ models with better complex instruction-following abilities. (3) We conduct extensive experiments to demonstrate the efficacy of CRAB and discuss key design choices and potential improvement opportunities to inspire future research on following complex instructions with multiple constraints.

2 Method

This section introduces the construction process of CRAB (§ 2.1) and the training method (§ 2.2).

2.1 Constructing CRAB

We begin by introducing the notions. Given an instruction x, which typically defines a specific task, such as "*Write a blog on French cuisine*", a set of constraints c, which specify conditions for the response, such as length restrictions, and a response y that satisfies both the constraints c and the instruction x, our goal is to construct a high-quality dataset of (x, c, y) triples. We first collect a set of high-quality (x, y) pairs from existing datasets and then apply constraint back-translation to generate the constraints c for each (x, y) pair. The data construction process is illustrated in Figure 2, which consists of three steps: data collection, constraint back-translation, and constraint combination. In the data collection process, we collect a comprehensive set of high-quality (x, y) pairs from existing datasets. We then back-translate the corresponding c for each (x, y) using Llama3-70B-Instruct and Python scripts automatically. Finally, we perform filtering and a combination of the constraints c to construct CRAB. More details of the data construction process are shown in appendix A.

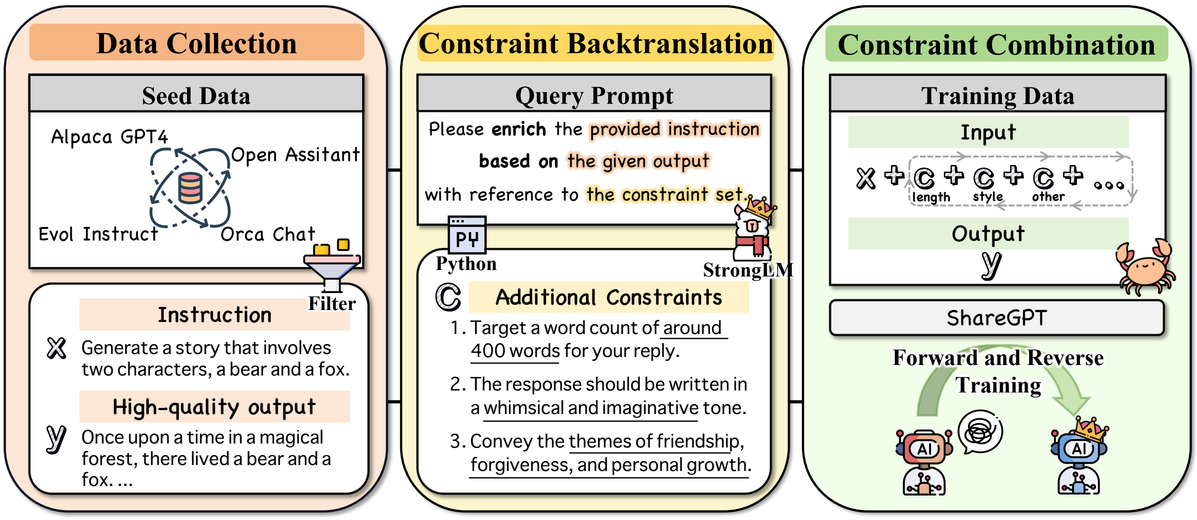

Data Collection
Seed Data
Alpaca GPT4
Open Assitant
Evol Instruct
Orca Chat
Filter
Instruction
Generate a story that involves two characters, a bear and a fox.
High-quality output
Once upon a time in a magical forest, there lived a bear and a fox. ...
Constraint Backtranslation
Query Prompt
Please enrich the provided instruction based on the given output with reference to the constraint set.
Python
StrongLM
Additional Constraints
1. Target a word count of around 400 words for your reply.
2. The response should be written in a whimsical and imaginative tone.
3. Convey the themes of friendship, forgiveness, and personal growth.
Constraint Combination
Training Data
Input
length
style
other
Output
ShareGPT
Forward and Reverse Training

Figure 2: The framework of constructing the proposed alignment training dataset.

Data Collection We first collect a comprehensive set of (x, y) as seed data from four existing widely-used high-quality supervised fine-tuning datasets, including Alpaca GPT4 (Peng et al., 2023), Orca Chat (Es, 2023), Evol Instruct (Xu et al., 2023), and OpenAssitant (Köpf et al., 2024). For Alpaca GPT4 and OpenAssistant, which contain the human annotated quality score for each instance, we use the instances with the highest quality. Moreover, to ensure that the responses include diverse constraints implicitly, we only consider instances where the response exceeds 300 words. We randomly sample the qualified instances using an examples-proportional mixture (Wei et al., 2022), resulting in a total of $4,500$ raw instances.

Constraint Back-translation We adopt the Llama3-70B-Instruct LLM and Python scripts to back-translate constraints for the seed data. We ultimately use Llama3-70B-Instruct to automatically generate constraints implicitly satisfied by the response from instruction-response pairs. To enhance the diversity of generated constraints, we manually collect 13 commonly used constraint types[1] as examples in the prompt for constraint generation, which results in over **100** constraint types. We then use Llama3-70B-Instruct to re-verify whether the response satisfies the generated constraints and exclude the constraints that are not met. Considering some constraints, e.g., length constraint, cannot be effectively followed by LLMs (Sun et al., 2024), leading to noisy back-translation, and some constraints can be easily generated using Python scripts, we choose to adopt Python scripts for 6 types of constraints. Specifically, we write and paraphrase several templates for each of these constraints. For example, for a length constraint, one template is "Please generate a response with fewer than `<placeholder>` words but more than `<placeholder>` words". We then use Python scripts to automatically identify the value for this constraint in the response and fill the templates to construct a constraint. For the length constraint, we randomly sample a range that includes the value to fill the template. For keyword and punctuation constraints, we randomly select corresponding items present in the response to fill the templates of constraints. We adopt ROUGE-L (Lin, 2004) with a threshold of 0.6 to exclude similar constraints. Finally, we sample and manually review 50 instruction-response pairs and their generated constraints, finding minimal noise and high compliance between constraints and response.

Constraint Combination Finally, we combine individual constraints to form the final constraint c for each instruction. Previous studies have shown that increasing the number of constraints in the training data leads to better model performance (He et al., 2024). Therefore, we enhance each instruction with a combination of multiple constraints. Specifically, we randomly sample 6 to 8 constraints from each instruction's constraint set generated in the previous step, shuffle their order, and recombine them into the final constraint c. Similar to previous work (Sun et al., 2024; Qi et al., 2024), we add 1 to 3 in-context demonstrations for 50% of the data. Finally, we construct CRAB with $13,500$

[1]We sample 200 instruction-response pairs from ShareGPT to observe real-world constraint needs and summarize them.

instances, containing an average of 7.1 constraints.

2.2 Model Training

To further enhance LLMs' understanding of complex constraints, we propose a *reverse training* method that takes the instruction-response pair (x, y) as input to teach LLMs to generate the constraints c. The intuition is that correctly generating constraints requires sufficient comprehension first. Formally, the reverse training objective is to minimize $\mathcal{L}_\mathrm{r}$, where $\mathcal{L}_\mathrm{r} = -\log P_\theta(c|x, y)$ and the LLM is parameterized by θ. We also adopt the standard supervised fine-tuning (SFT; Ouyang et al., 2022), named *forward training*, to minimize $\mathcal{L}_\mathrm{f}$, where $\mathcal{L}_\mathrm{f} = -\log P_\theta(y|x, c)$. The final training objective is a combination of $\mathcal{L}_\mathrm{f}$ and $\mathcal{L}_\mathrm{r}$: $\mathcal{L} = \alpha\mathcal{L}_\mathrm{f} + (1 - \alpha)\mathcal{L}_\mathrm{r}$. We set α to 0 during 70% of the training process, and to 1 for the remaining time. We train the LLM using a mixture of ShareGPT (Chiang et al., 2023) and CRAB. We adopt $\mathcal{L}_\mathrm{f}$ when training on ShareGPT (Chiang et al., 2023) and adopt $\mathcal{L}$ on CRAB. We train a base LLM on this data and obtain the SFT version of our model. Based on the SFT trained model, we continue training using the Direct Preference Optimization objective (DPO; Rafailov et al., 2023). Specifically, same as the DPO phrase by Sun et al. (2024), we use the high-quality DPO dataset UltraFeedback (Cui et al., 2023) to conduct further training and obtain the DPO version of our model. More training details are presented in appendix B.

3 Experiments

In this section, we introduce the experimental setup (§ 3.1), experimental results (§ 3.2), and further analyses on our model (§§ 3.3 to 3.5).

3.1 Experimental Setup

Backbone Models We adopt two widely-used open-source base models, Mistral 7B (Jiang et al., 2023) and Llama 3 8B (Dubey et al., 2024), as our backbone models for developing Llama3$_\mathrm{CRAB}$ and Mistral$_\mathrm{CRAB}$. Specifically, we employ `Mistral-7B-v0.3` and `Meta-Llama-3-8B`, downloaded from Hugging Face (Wolf et al., 2019). During the SFT stage, we adopt a 5×10^{-6} learning rate, 256 batch size, and train the Mistral for 4 epochs and Llama 3 for 3 epochs. During the DPO optimization stage, we adopt 5×10^{-7} learning rate, 64 batch size, and 1 training epoch.

Baselines Our baselines include popular open-source and proprietary LLMs, divided into three main categories for comparison: (1) Proprietary LLMs, including GPT-3.5 (OpenAI, 2022) and GPT-4 (OpenAI, 2024). (2) General instruction-tuning LLMs, including Vicuna-V1.5 13B (Chiang et al., 2023), trained on the 125k ShareGPT dataset, WizardLM-V1.2 13B (Xu et al., 2023), trained on the 196k Evol-Instruct dataset, Zephyr beta 7B (Tunstall et al., 2023), trained with the UltraFeedback (Cui et al., 2023) dataset using the DPO objective (Rafailov et al., 2023), and Mistral-Instruct 7B v0.3 (Jiang et al., 2023), which achieves leading performance on chat benchmarks based on Mistral 7B (3) Models specifically optimized for complex instruction-following tasks, including Suri-I-ORPO (Pham et al., 2024), which is optimized for multi-constraint instruction-following tasks in long-form text generation, and the Conifer series (Sun et al., 2024), are trained on the data where the constraints, instructions, and responses are all generated using GPT-4.

Evaluation Datasets We use two widely-used and challenging complex instruction-following datasets IFEval (Zhou et al., 2023) and FollowBench (Jiang et al., 2024) for evaluation. IFEval consists of 541 instructions that can be automatically validated using Python scripts. Each instruction contains 1 to 3 constraints, primarily focusing on strict lexical and formatting constraints. FollowBench is a fine-grained, multi-constraint instruction-following benchmark and it categorizes the difficulty into five levels (L1 to L5) based on the number of constraints of an instruction, where L1 represents the simplest level with only one constraint, while L5 is the most difficult, with a combination of five constraints. It also includes five constraint categories, including *content*, *situation*, *style*, *format*, and *example*, along with a *mixed* constraint category that combines various categories of constraints. FollowBench contains a total of 820 instructions across more than 50 different NLP tasks, and it is automatically evaluated using either Python scripts or GPT-4. Please refer to the original paper for more details (Jiang et al., 2024).

3.2 Experimental Results

The experimental results are presented in Table 1. Our observations are as follows: (1) After training on the CRAB dataset, our models significantly outperform the corresponding base mod-

Model	Backbone	IFEval					FollowBench (HSR)						AVG
		[S]P	[S]I	[L]P	[L]I	AVG	L1	L2	L3	L4	L5	AVG	
GPT-3.5*	GPT	59.0	68.5	64.0	73.6	66.3	80.3	68.0	68.6	61.1	53.2	66.2	66.3
GPT-4†	GPT	76.9	83.6	79.3	85.4	81.3	84.7	76.1	71.3	74.5	62.4	73.8	77.6
Vicuna-v1.5-13B†	Llama2	43.1	53.6	46.6	58.0	50.3	**71.2**	**61.3**	48.3	38.0	33.1	**50.4**	50.4
WizardLM-v1.2-13B	Llama2	43.6	54.4	48.4	59.1	51.4	61.3	51.6	43.3	37.5	29.9	44.7	48.1
Conifer$_{\text{SFT}}$-13B†	Llama2	42.9	53.0	47.5	57.4	50.2	60.5	53.6	48.4	40.7	31.7	47.0	48.6
Zephyr-beta-7B†	Mistral	32.0	46.8	44.9	58.0	45.4	57.6	51.9	41.9	41.4	31.4	44.8	45.1
Mistral$_{\text{Instruct}}$-7B	Mistral	48.6	59.8	53.2	64.3	56.5	57.1	51.5	43.6	41.5	33.2	45.4	50.9
Suri$_{\text{I-ORPO}}$-7B	Mistral	47.3	58.0	51.4	62.0	54.7	45.4	41.4	24.2	18.6	15.2	29.0	41.9
Conifer$_{\text{SFT}}$-7B†	Mistral	45.8	57.1	50.8	62.0	53.9	54.3	49.5	49.3	40.8	30.5	44.9	49.4
Conifer$_{\text{DPO}}$-7B†	Mistral	48.1	59.1	52.3	63.3	55.7	60.3	53.6	48.0	**47.1**	**41.0**	50.0	52.9
Llama3-8B	Llama3	25.7	36.8	28.1	35.1	31.4	4.8	8.7	8.8	6.0	9.8	7.6	19.5
Llama3$_{\text{CRAB}}$	Llama3	39.4	50.2	43.8	54.2	46.9	57.5	44.9	34.9	25.2	20.0	36.5	41.7
Llama3$_{\text{CRAB}}$ + DPO	Llama3	40.3	52.0	47.7	58.9	49.7	64.6	49.0	41.6	35.8	36.8	45.5	47.6
Mistral-7B	Mistral	18.5	30.8	19.6	31.9	25.2	14.3	16.6	8.3	5.8	5.5	10.1	17.7
Mistral$_{\text{CRAB}}$	Mistral	47.9	57.3	51.6	61.2	54.5	63.9	54.4	40.1	30.4	27.9	43.3	48.9
Mistral$_{\text{CRAB}}$ + DPO	Mistral	**49.7**	**61.5**	**57.7**	**68.5**	**59.3**	66.1	53.6	**53.4**	42.4	31.7	49.4	**54.4**

Table 1: Experimental results (%) of the LLMs on IFEval and FollowBench. In IFEval, "[S]" and "[L]' denote strict and loose accuracy, "P" and "I" indicate the prompt and instruction level. In FollowBench, L1 (simplest) to L5 (hardest) denote different difficulty levels. We highlight the highest and second-highest scores of open-source LLMs using **bold** font and underline. † and * means the results are from Sun et al. (2024) and He et al. (2024).

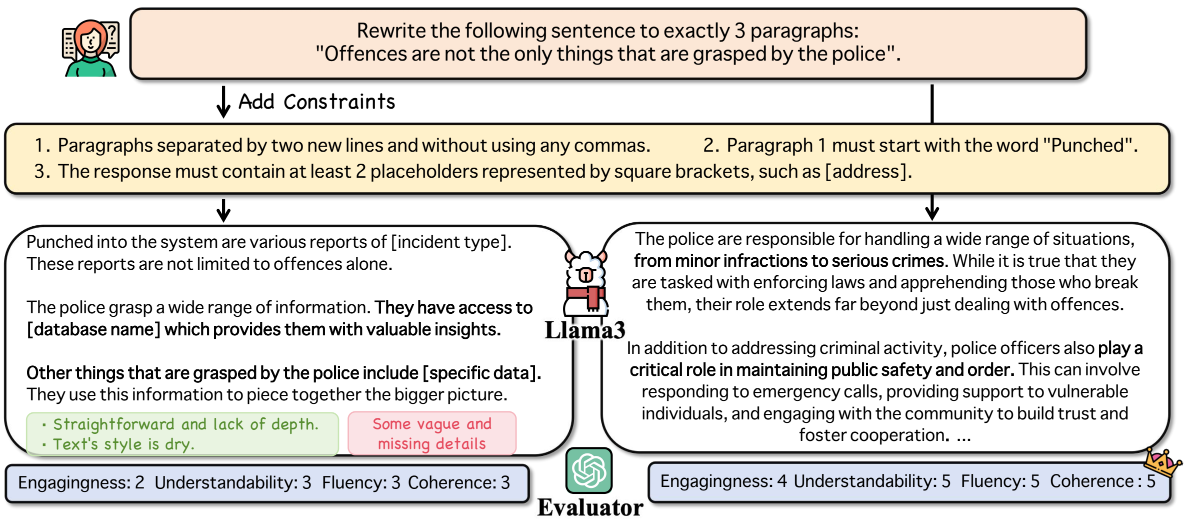
Model	LC WinRate	WinRate
GPT-3.5-turbo-0613†	22.4	14.1
GPT-4-0613†	30.2	15.8
WizardLM-70B†	17.6	14.4
WizardLM-v1.2-13B†	14.5	12.0
Vicuna-v1.5-13B†	10.5	6.7
Zephyr-beta-7B†	13.2	11.0
Conifer$_{\text{DPO}}$-7B†	17.1	11.3
Mistral$_{\text{CRAB}}$	13.3	7.9
Mistral$_{\text{CRAB}}$ + DPO	**18.1**	**17.6**
`(vs.) Conifer`$_{\text{DPO}}$	60.6	63.5

Table 2: Winning rate (%) of the investigated LLMs on Alpaca-Eval 2.0 (Li et al., 2023b). "LC" denotes length-controlled (Dubois et al., 2024). † means the results are sourced from the original leaderboard.

els and the open-source models trained through SFT on general instruction-following datasets. Our DPO version of models achieves the best performance among the compared models. It demonstrates the effectiveness of our data and training approach. (2) Our models surpass Conifer Sun et al. (2024), which is specifically trained for complex instruction-following, on IFEval. It suggests that our model performs better in following lexical and format constraints. However, our models slightly lag behind Conifer on FollowBench. We provide an in-depth discussion on the performance across different constraint categories in FollowBench in § 3.5. We observe that the performance decline is primarily due to the style constraint, where our models significantly underperform in this constraint category compared to Conifer. Nonetheless, Our models achieve significant improvements in real-world scenarios, i.e., the *mixed* constraint in FollowBench (Jiang et al., 2024), compared to Conifer. (3) Training with the DPO objective consistently improves model performance on both evaluation datasets. In this paper, we focus on constructing high-quality SFT data by constraint back-translation, we leave the development of DPO data for complex instruction-following as future work.

3.3 Analysis on General Instruction Following

The complex instruction-following ability not only involves following complex constraints but also encompasses the basic ability to follow instructions themselves, e.g., "*Write a blog on French cuisine*", named as general instruction following. In this section, we further evaluate our model's general instruction-following capability. Given that IFEval and FollowBench primarily focus on evaluating the ability to follow constraints, we adopt another widely-used dataset, AlpacaEval (Li et al., 2023b), which serves as an easy-to-use and high-quality automatic evaluator for instruction-following ability. Specifically, we use AlpacaEval 2.0, which contains 805 instructions, and use `gpt-4-1106-preview` as the evaluator to get the final weighted win rate. The evaluation results are presented in Table 2, where the "LC WinRate" represents the length-controlled win rate (Dubois et al., 2024). The default reference model is `gpt-4-1106-preview`. We can observe that our


Rewrite the following sentence to exactly 3 paragraphs:
"Offences are not the only things that are grasped by the police".

1. Paragraphs separated by two new lines and without using any commas. 2. Paragraph 1 must start with the word "Punched".
3. The response must contain at least 2 placeholders represented by square brackets, such as [address].

Punched into the system are various reports of [incident type]. These reports are not limited to offences alone.

The police grasp a wide range of information. **They have access to [database name] which provides them with valuable insights.**

Other things that are grasped by the police include [specific data]. They use this information to piece together the bigger picture.

The police are responsible for handling a wide range of situations, **from minor infractions to serious crimes.** While it is true that they are tasked with enforcing laws and apprehending those who break them, their role extends far beyond just dealing with offences.

In addition to addressing criminal activity, police officers also **play a critical role in maintaining public safety and order.** This can involve responding to emergency calls, providing support to vulnerable individuals, and engaging with the community to build trust and foster cooperation. ...

Engagingness: 2 Understandability: 3 Fluency: 3 Coherence: 3

Engagingness: 4 Understandability: 5 Fluency: 5 Coherence : 5

Add Constraints
Llama3
· Straightforward and lack of depth.
· Text's style is dry.
Some vague and missing details
Evaluator

Figure 3: An example of responses generated with and without constraints by Llama3-70B-Instruct. The evaluator is `gpt-4o-0806`. For better visualization, we present only a subset of the responses generated without constraints.

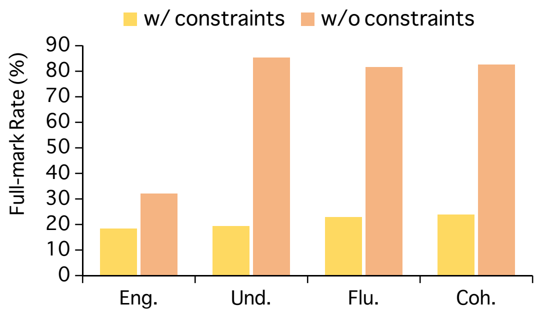


Figure 4: Full-mark rates (%) of the responses generated with and without constraints. The evaluator is `gpt-4o-0806`, focusing on four widely-used dimensions: Engagingness (Eng.), Understandability (Und.), Fluency (Flu.), and Coherence (Coh.).

model significantly outperforms the baseline model Conifer and even exceeds the performance of the 70B version of WizardLM. We also conduct a head-to-head comparison between our model and Conifer. The LC win rate of our model reaches 60.6, significantly outperforming Conifer, which demonstrates that our model possesses a superior general instruction-following capability.

We conduct a further analysis to explore the potential reasons why our model outperforms Conifer in general instruction-following. The primary difference between our model and Conifer is the data construction process. We utilize *constraint back-translation*, where the response is generated directly from the instruction without constraints. In contrast, Conifer uses instruction and corresponding constraints to generate the response. We hypothesize that a possible reason is that the response quality in Conifer is lower than CRAB, that is, generating a response conditioned on both instruction and constraints may result in **lower content quality**, such as lower coherence, compared to the response directly generated from instruction without constraints. Intuitively, incorporating constraints may limit the model's capacity when generating responses. To validate this intuition, we conduct a controlled analytical experiment. Specifically, we sample 100 instructions and their corresponding constraints from IFEval and FollowBench. We first use Llama3-70B-Instruct to generate responses based only on the instructions (w/o constraints). Then, we include the additional constraints and generate corresponding responses (w/ constraints). Following previous work on automated evaluation using advanced LLMs (Bai et al., 2024b; Chan et al., 2024), we employ `gpt-4o-0806` as the evaluator, assessing the responses on four dimensions: *engagingness*, *understandability*, *fluency*, and *coherence*, with scores ranging from 1 to 5. We report the full-mark (score 5) rate for each dimension. The results are shown in Figure 4. We can observe that responses generated with constraints significantly underperform those generated without constraints, which suggests that involving constraints when generation may reduce the content quality of the final response. We further conduct a case study, illustrated in Figure 3, showing an example of responses generated with and without constraints. We can find that when involving constraints, the response includes vague terms, such as "*[database name]*", and lacks sufficient details and depth. While previ-

Model	IFEval	FollowBench		AVG
	AVG	L1-L2	L3-L5	
Mistral$_{\text{CRAB}}$	54.5	59.1	32.8	48.9
(-) *Reverse training*	52.1	56.2	33.5	47.3
(-) *Forward training*	53.9	57.1	32.1	48.0
(-) *In-Context Demons*	53.6	55.8	30.0	47.0
InstBackT$_{\text{SFT}}$	52.7	55.4	29.3	46.2

Table 3: Experimental results (%) of the ablation study. *In-Context Demons* denotes in-context demonstrations.

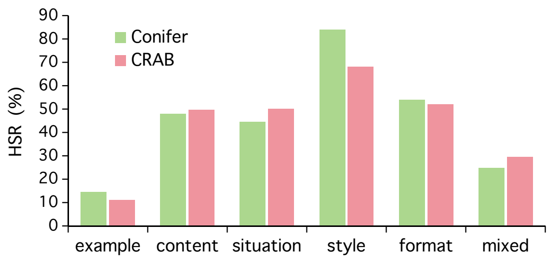


Figure 5: Experimental results on different categories of constraints in FollowBench of Mistral$_{\text{CRAB}}$ and Conifer$_{\text{SFT}}$.

ous work on complex instruction-following mainly focuses on enhancing the ability to follow multiple constraints, we encourage future work to prioritize response content quality, and constraint back-translation can serve as a potential solution. Assessment details of this analysis are in appendix C.

3.4 Ablation Study

We conduct an ablation study to analyze the key factors influencing model performance. Specifically, we investigate three key factors in developing our model: *reverse training*, *forward training*, i.e., standard supervised fine-tuning, and *in-context demonstrations*. We exclude each factor and keep all other conditions identical, to the model separately. When excluding reverse and forward training, we set the loss ratio α in § 2.2 to 1 and 0, respectively. The backbone model is Mistral. The results are presented in Table 3, where L1-L2 in FollowBench represent simpler constraints and L3-L5 denote more complex constraints. We can observe that removing any of these factors leads to a decline in model performance, which demonstrates the effectiveness of these factors in developing our model. For more complex constraints following, adding in-context demonstrations during training is effective, as excluding in-context demonstrations leads to a significant performance drop in L3-L5. The reason may be that in-context demonstrations enhance the model's ability to understand multiple in-context instructions and complex constraints.

We further compare with a competitive baseline model, InstBackT$_{\text{SFT}}$, which is trained on the data generated by *instruction back-translation* (Li et al., 2024). The key difference between instruction and constraint back-translation is that the former uses advanced LLMs to generate both instructions and constraints from responses, while the latter focuses on generating constraints from instruction and response pairs. The results are presented in Table 3. We can observe that InstBackT$_{\text{SFT}}$ significantly underperforms compared to Mistral$_{\text{CRAB}}$, which suggests that instruction back-translation may produce lower-quality data for complex instruction following. The possible reason is that generating both instructions and constraints simultaneously is more challenging than generating constraints alone. It further demonstrates the efficacy of the constraint back-translation method in creating high-quality training data for complex instruction following.

3.5 Analysis on Constraint Category

We further investigate our model's performance across different constraint categories to analyze its strengths and potential limitations. Specifically, we analyze the results on FollowBench, which includes five categories of constraints, including *example*, *content*, *situation*, *style*, and *format*. Please refer to the original paper (Jiang et al., 2024) for the detailed definitions for each constraint category. FollowBench also includes a *mixed* category which is designed for simulating **real-world** scenarios (Jiang et al., 2024), where various types of constraints are combined to form the final constraint. We compare our model Mistral$_{\text{CRAB}}$ with the Conifer model, which is trained on the data generated using the standard pipeline: generating the constraints first and then generating the response based on the instruction and constraints. The results on different constraint categories of FollowBench are shown in Figure 5. We can observe that our model significantly outperforms Conifer on the *mixed* constraint, which represents real-world scenarios, suggesting that our model is more effective in handling complex instruction-following scenarios. However, in the *style* constraint category, e.g., "*Write in the style of Shakespeare*", our model performs significantly worse than Conifer. The possible reason is that the style constraints in our dataset CRAB may be not sufficiently diverse. The data for *style* constraints requires deliberate construction, and the pipeline that generates constraints first and then responses is more effective at generating diverse style constraints, but the responses in our seed data have limited style diversity. It suggests a limitation of constraint back-translation, as it relies on diverse responses to generate specific categories of constraints, such as style constraint. Combining the constraint back-translation method with other data generation methods to produce higher-quality data for those specific constraints can further enhance the model's complex instruction-following ability, and we leave this exploration as future work.

4 Related Work

4.1 Instruction Following

Instruction following involves following user intentions to generate helpful responses, which is fundamental to modern LLMs (Zhang et al., 2023). Ouyang et al. (2022) first propose the practice of aligning LLMs to follow human instructions, using SFT and RLHF to train models, which is the key factor in the success of ChatGPT (OpenAI, 2022). Subsequently, numerous studies focus on enhancing the instruction-following capabilities of LLMs, particularly for open-source models, which can be summarized in two main aspects: (1) data-driven approaches, which design an automated pipeline or use human annotation to produce high-quality training data (Xu et al., 2023; Taori et al., 2023; Chiang et al., 2023; Peng et al., 2023; Kim et al., 2023; Cui et al., 2023; Mukherjee et al., 2023; Ivison et al., 2023; Köpf et al., 2024; Qi et al., 2024; Liu et al., 2024; Li et al., 2024; Bai et al., 2024a). (2) new training methods, including novel objectives (Rafailov et al., 2023; Gallego, 2024; Zhou et al., 2024; Hejna and Sadigh, 2024; Meng et al., 2024) or training pipelines (Tunstall et al., 2023; Li et al., 2024; Yuan et al., 2024; Chen et al., 2024).

A more challenging instruction following scenario is complex constrained instruction following, where the responses should further satisfy specific constraints, such as length. Previous studies have shown that LLMs struggle to follow these instructions (Jiang et al., 2024; Qin et al., 2024). Recent efforts focus on enhancing this ability by constructing high-quality training data (Sun et al., 2024; He et al., 2024). This process typically involves collecting a set of instructions, constructing constraints, and then generating responses based on the instructions and constraints using advanced LLMs. This work introduces *constraint back-translation*, which generates constraints from instruction-response pairs, reducing data construction costs and noise.

4.2 Back-translation

Back-translation is first proposed in the field of machine translation (Sennrich, 2015; Hoang et al., 2018), which mainly is used for data augmentation. It first trains a model to back-translate the target language into the source language, then uses this model to generate parallel training data from a large amount of monolingual target language data, which sufficiently saves human translation efforts. Considering its simplicity and efficacy, back-translation has also been widely applied to various tasks, such as style transfer (Prabhumoye et al., 2018; Toshevska and Gievska, 2021) and paraphrase generation (Wieting et al., 2017; Mallinson et al., 2017).

Recently, several studies have explored applying back-translation to the field of large language models to efficiently generate high-quality data automatically (Li et al., 2023a; Pham et al., 2024; Köksal et al., 2023). Li et al. (2023a) proposed reversing the training objective to automatically generate corresponding instructions for existing unsupervised corpora, while Pham et al. (2024) and Köksal et al. (2023) leveraged the powerful general capabilities of LLMs to generate instructions from the corpus directly. Although Pham et al. (2024) also generated constraints, it fell within the realm of *instruction back-translation* and did not involve dedicated optimization or exploration for constraint generation. In this work, we propose *constraint back-translation*, an effective data generation approach that generates high-quality constraints based on instruction-response pairs.

5 Conclusion

In this paper, we aim to enhance large language models' capability for complex constrained instruction following. We propose a *constraint back-translation* data generation method, which can reduce data noise and generation costs, resulting in a high-quality complex instruction-following dataset CRAB. We also propose a *reverse training* method and develop Llama3$_{\text{CRAB}}$ and Mistral$_{\text{CRAB}}$ based on CRAB. Extensive experiments demonstrate the effectiveness of our data generation and training methods. We further conduct extensive analytical experiments and discuss the key factors, advantages, and potential limitations of our model.

Limitations

As discussed in § 3.5, for certain types of constraints, such as style constraint, the constraints generated through constraint back-translation may lack sufficient diversity if the original response data itself is not diverse enough. We leave further improvements to constraint back-translation as future work. Another limitation of our study is that we do not use a larger base model due to computational constraints. We believe that using a larger base model could develop a more advanced LLM in following complex constraints, but this does not affect our overall experimental conclusions.

Ethical Considerations

We discuss potential ethical concerns related to this work: (1) **Intellectual property**. Our research leverages several widely used SFT datasets, and we strictly comply with their licensing terms. We will share CRAB under the CC BY-SA 4.0 license[2]. (2) **Intended use and Potential risk control**. The goal of this paper is to introduce CRAB, designed to enhance the performance of LLMs on complex instruction tasks. CRAB is built using widely available public datasets. We trust that the original publishers have anonymized and sanitized these datasets appropriately. The data construction process does not include additional social bias. Additionally, we randomly sampled 100 instances from our dataset and found no sensitive information. (3) **AI assistance**. We used GPT-4 to paraphrase some sentences and check grammar.

References

Yushi Bai, Xin Lv, Jiajie Zhang, Yuze He, Ji Qi, Lei Hou, Jie Tang, Yuxiao Dong, and Juanzi Li. 2024a. Longalign: A recipe for long context alignment of large language models. *arXiv preprint arXiv:2401.18058*.

Yushi Bai, Jiahao Ying, Yixin Cao, Xin Lv, Yuze He, Xiaozhi Wang, Jifan Yu, Kaisheng Zeng, Yijia Xiao, Haozhe Lyu, et al. 2024b. Benchmarking foundation models with language-model-as-an-examiner. *Advances in NeurIPS*, 36.

Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. 2020. Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289.

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

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. 2024. Chateval: Towards better llm-based evaluators through multi-agent debate. In *ICLR*.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. Self-play fine-tuning converts weak language models to strong language models. In *ICML*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E Gonzalez, et al. 2023. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality. *See https://vicuna. lmsys. org (accessed 14 April 2023)*, 2(3):6.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*.

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

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*.

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

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

Víctor Gallego. 2024. Refined direct preference optimization with synthetic data for behavioral alignment of llms. *arXiv preprint arXiv:2402.08005*.

Olga Golovneva, Zeyuan Allen-Zhu, Jason Weston, and Sainbayar Sukhbaatar. 2024. Reverse training to nurse the reversal curse. *arXiv preprint arXiv:2403.13799*.

Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. 2024. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. *Preprint*, arXiv:2404.15846.

Joey Hejna and Dorsa Sadigh. 2024. Inverse preference learning: Preference-based rl without a reward function. *Advances in Neural Information Processing Systems*, 36.

Cong Duy Vu Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative back-translation for neural machine translation. In *2nd*

Workshop on Neural Machine Translation and Generation, pages 18–24. ACL.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A Smith, Iz Beltagy, et al. 2023. Camels in a changing climate: Enhancing lm adaptation with tulu 2. *arXiv preprint arXiv:2311.10702*.

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. 2023. 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. 2024. Follow-Bench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of ACL*, pages 4667–4688.

Sungdong Kim, Sanghwan Bae, Jamin Shin, Soyoung Kang, Donghyun Kwak, Kang Yoo, and Minjoon Seo. 2023. Aligning large language models through synthetic feedback. In *Proceedings of EMNLP*, pages 13677–13700.

Abdullatif Köksal, Timo Schick, Anna Korhonen, and Hinrich Schütze. 2023. Longform: Effective instruction tuning with reverse instructions. *arXiv preprint arXiv:2304.08460*.

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

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason Weston, and Mike Lewis. 2023a. Self-alignment with instruction back-translation. *arXiv preprint arXiv:2308.06259*.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. 2024. Self-alignment with instruction back-translation. In *ICLR*.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023b. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Yantao Liu, Zhao Zhang, Zijun Yao, Shulin Cao, Lei Hou, and Juanzi Li. 2024. Aligning teacher with student preferences for tailored training data generation. *arXiv preprint arXiv:2406.19227*.

Edward Loper and Steven Bird. 2002. Nltk: The natural language toolkit. *arXiv preprint cs/0205028*.

Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. 2017. Paraphrasing revisited with neural machine translation. In *Proceedings of ACL*, pages 881–893.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *arXiv preprint arXiv:2405.14734*.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*.

OpenAI. 2022. Introducing ChatGPT.

OpenAI. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

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 NeurIPS*, 35:27730–27744.

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

Chau Minh Pham, Simeng Sun, and Mohit Iyyer. 2024. Suri: Multi-constraint instruction following for long-form text generation. *arXiv preprint arXiv:2406.19371*.

Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W Black. 2018. Style transfer through back-translation. *arXiv preprint arXiv:1804.09000*.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2024. Adelie: Aligning large language models on information extraction. *arXiv preprint arXiv:2405.05008*.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. Infobench: Evaluating instruction following ability in large language models. *arXiv preprint arXiv:2401.03601*.

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

Rico Sennrich. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

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*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Alpaca: A strong, replicable instruction-following model. *Stanford Center for Research on Foundation Models.*, 3(6):7.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Martina Toshevska and Sonja Gievska. 2021. A review of text style transfer using deep learning. *IEEE Transactions on Artificial Intelligence*, 3(5):669–684.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. 2023. Zephyr: Direct distillation of lm alignment. *Preprint*, arXiv:2310.16944.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. In *Proceedings of ICLR*.

John Wieting, Jonathan Mallinson, and Kevin Gimpel. 2017. Learning paraphrastic sentence embeddings from back-translated bitext. *arXiv preprint arXiv:1706.01847*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. arxiv. *arXiv preprint arXiv:1910.03771*.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason E Weston. 2024. Self-rewarding language models. In *ICML*.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

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*.

Zhanhui Zhou, Jie Liu, Jing Shao, Xiangyu Yue, Chao Yang, Wanli Ouyang, and Yu Qiao. 2024. Beyond one-preference-fits-all alignment: Multi-objective direct preference optimization. In *Findings of ACL*, pages 10586–10613.

Appendices

A Data Collection

In this section, we provide a detailed explanation of our data construction process, divided into three parts: the details of constraint construction (appendix A.1), the data diversity of CRAB (appendix A.2) and the data distribution of CRAB (appendix A.3).

A.1 Details of Constraints Construction

Table 4 presents the definitions of each constraint in our constraint set. It is important to note that for the "Situation", clarifying the subject or object, or defining the circumstances under which the instruction applies, we observed that generating this constraint independently often results in this additional constraint being too similar to the original instruction. Therefore, we integrate it directly with the original interaction to develop a refined instruction. If selected during the combination process, instead of being added to the instruction like other constraints, it replaces the original instruction.

Among the constraints calculated using Python scripts, two categories are particularly unique: (1) Number-related categories: such as Length and Words Per Sentence, where we used NLTK (Loper and Bird, 2002) for calculation. (2) Keyword: We applied the lightweight, unsupervised keyword extraction method Yake (Campos et al., 2020) to extract the top 3 most significant keywords from the output text. Table 7 provides an example generated after the constraint back-translation process.

A.2 Dataset Diversity

We adopted 4 widely used post-training datasets for constructing our CRAB dataset: Alpaca GPT-4, Open Assistant, Evol-Instruct, and Orca Chat. These datasets are of high quality and contain diverse data instances for generating rich constraints. The "Rate" column in Table 4 shows the distribution of constraint types in the CRAB dataset. Moreover, for constraints generated by the LLM, since we provided 13 types of constraints as examples in the prompt, we analyzed the keywords of each generated constraint and extracted the top 10 keywords for each major category, presenting them as subcategories in Figure 7. The "situation" category was not included in the analysis because it is task-specific, making clustering difficult.

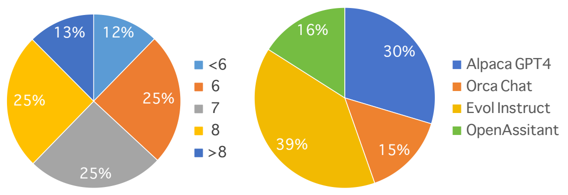


Figure 6: Proportion (%) of data in the CRAB by the number of constraints and the source dataset.

A.3 Dataset Distrubution

Figure 6 shows the distribution of $13,500$ instances in the CRAB. The left chart categorizes data by the number of constraints after combination, while the right chart categorizes data by the source dataset. To enhance data diversity during the combination stage, we randomly introduced 25% of data with a constraint count outside the 6–8 range, with the maximum number of constraints being 14.

B Model Training

For model training, we utilize the repository 'The Alignment Handbook' (Tunstall et al., 2023) to train Mistral 7B based models, and use OpenInstruct (Ivison et al., 2023) to train LLaMA 3 8B based models. The implementation of the ratio α between reverse training and forward training is achieved by segregating the dataset into two parts, since the model may memorize the data during the forward process.

All experiments in the paper are done using 8 NVIDIA A100 80GB GPUs. We adopt DeepSpeed ZeRO stage 2 for SFT and DPO training. The training of Mistral took approximately 48 GPU hours in total, while the training of LLaMA3 took around 72 GPU hours.

In the SFT stage, we set the learning rate to 5×10^{-6}, with a per-device batch size of 4 and 8 gradient accumulation steps. The warm-up ratio is set to 0.1. The Mistral-7B experiments are trained for 4 epochs with a maximum sequence length of 2048, while the LLaMA 3 8B experiments are trained for 3 epochs with a maximum sequence length of 4096. For DPO training, the learning rate is set to 5×10^{-7}, with a per-device batch size of 4, 2 gradient accumulation steps, and a maximum sequence length of 2048. The beta value for Mistral 7B experiments is set to 0.01, trained for 1 epoch with a cosine learning rate schedule, while for LLaMA 3 8B, the beta is 0.1, trained for 3 epochs with a linear learning rate scheduler following the setting in Ivison et al. (2023).

C Details on the impact of constraints on output quality

To explore the impact of constraints on output quality, we sampled 100 instruction pairs from FollowBench and IFEval, with each pair consisting of a instruction without constraints and its corresponding multi-constraint version (with over 3 constraints). Since IFEval does not provide instruction without constraints, we randomly selected 50 instances and manually removed the constraints. For FollowBench, we selected level 0 instructions along with their corresponding level 5 counterparts. To ensure a fair comparison, we only retained instruction pairs where the core meaning of the instruction pairs remained consistent, such that the output generated from the complex instructions would still satisfy the simple versions.

We evaluate the quality of model output along the following four dimensions.

- **Engagingness:** Evaluate how captivating and interesting the text is, based on its ability to hold attention and evoke interest.

  Components: Interest (ability to sustain attention), Appeal (suitability for the audience), and Emotional/Intellectual Impact.

- **Understandability:** Evaluate the clarity and ease with which the text can be understood by the target audience.

  Components: Simplicity (absence of unnecessary complexity), Accessibility (use of language suitable for the audience), and Clarity.

- **Fluency:** Evaluate the smoothness of the writing, focusing on grammar, sentence structure, and the natural flow of language.

  Components: Grammar (correct use of language rules), Sentence Structure (variety and complexity), and Naturalness (how easily the text flows).

- **Coherence:** Evaluate the logical flow and consistency of ideas, ensuring the text's structure is logical and ideas are connected.

  Components: Logical Flow (clear progression of ideas), Transitions (smooth movement between topics or sentences), and Consistency (absence of contradictions or disjointed thoughts).

D More Results

In this section, we present additional experimental results, divided into three parts: full results on Followbench D.1, a fairer comparison where Conifer is replaced with the same backbone as ours D.2, and our experimental results on LLaMA3.2-3B D.3.

D.1 SSR results on Followbench

We report FollowBench results under the Hard Satisfaction Rate (HSR) metric in Table 1. Table 5 presents results on FollowBench under Soft Satisfaction Rate (SSR) metric. We also conducted a comparison with the ShareGPT version, which is trained exclusively on the ShareGPT dataset.

D.2 Different Backbone Comparison

Since Sun et al. (2024) did not specify the model version of Conifer, we reproduced Conifer on Mistral-7B-v0.3, which is the backbone used in Mistral$_{\text{CRAB}}$, and the results are presented in Table 6. All conclusions remain consistent with those stated in the main text.

D.3 More Backbone Models

To verify the scalability and applicability of our approach, we conducted experiments using LLaMA3.2-3B (Dubey et al., 2024) as the backbone, with all training hyperparameters consistent with those of LLaMA3-8B. As shown in Table 6, CRAB significantly improves the model's performance on complex instruction-following tasks.

Constraint Category	Description	Generator	Rate
Situation	Adding conditions, clarifying the subject or object, or defining the circumstances under which the instruction applies.	LLM	36.9
Writing Style	Specify the style requirements for the response to align with the intended message and audience.	LLM	81.3
Semantic Elements	Clearly articulate the main theme, focus, meaning, or underlying concept of the response.	LLM	99.5
Morphological	Outline specific prohibitions, such as avoiding certain words or phrases and refraining from specific formatting styles.	LLM	99.7
Multi-lingual	Specify the language(s).	LLM	94.8
Literary Devices	Identify any particular literary devices to be employed.	LLM	91.7
Grammatical Structure	Specify the grammatical structure.	LLM	99.1
Hierarchical Instructions	Establish a response hierarchy, defining the prioritization and structuring of tasks within the output.	LLM	83.1
Output Format	Depending on the required format of the output—such as Python, tables, JSON, HTML, LaTeX—impose relevant format constraints.	LLM	15.2
Paragraphs Constraints	Clearly specify the required number of paragraphs or sections in the text. Additionally, indicate any specific spacing or separators needed—such as blank lines, horizontal rules, or special symbols to enhance readability and visual appeal.	LLM	71.3
Specific Sentence	Specify a particular phrase to be included either at the beginning or end of the text, clearly indicating its exact placement.	LLM	70.1
Header Format	Specify the formatting style for titles or keywords within the Output, such as using bold, italics, or CAPITAL LETTERS.	LLM	9.5
Item Listing Details	Clearly specify the formatting for individual entries within the text. Direct the use of specific symbols for listing—such as bullet points (•), numbers (1., 2., 3., etc.), or hyphens (-).	LLM	67.7
Length Constraint	Determine the word count of the output text to establish length constraints.	Python	47.6
Word Constraint	Determine the number of words in each sentence to set word constraints.	Python	15.1
Sentence Constraint	Determine the number of sentences in each paragraph to establish sentence constraints.	Python	20.6
Character Constraint	Determine the number of characters in each word.	Python	21.4
Keyword Constraint	Determine the keywords in the output text to make the constraints more detailed.	Python	45.1
Punctuation Limitation	Specify which punctuation marks cannot be used in the output text.	Python	21.6

Table 4: Constraint types defined during the back-translation process. The Rate (%) indicates the proportion of instances in the entire dataset that generated constraints of this category.

Model	Backbone	IFEval					FollowBench (SSR)						AVG
		[S]P	[S]I	[L]P	[L]I	AVG	L1	L2	L3	L4	L5	AVG	
Llama3-ShareGPT*	Llama3	23.7	26.4	33.8	37.1	30.3	44.0	40.0	39.6	33.3	33.6	38.1	34.2
Llama3$_{\text{CRAB}}$	Llama3	39.4	50.2	43.8	54.2	46.9	57.5	52.4	51.2	47.0	45.6	50.7	48.8
Llama3$_{\text{CRAB}}$ + DPO	Llama3	40.3	52.0	47.7	58.9	49.7	64.6	55.8	54.7	52.4	54.0	56.3	53.0
Mistral-ShareGPT†	Mistral	37.5	49.3	43.4	54.9	46.3	55.7	56.6	53.6	53.4	49.7	53.8	50.0
Mistral$_{\text{CRAB}}$	Mistral	47.9	57.3	51.6	61.2	54.5	63.9	60.6	55.1	50.4	49.4	55.9	55.2
Mistral$_{\text{CRAB}}$ + DPO	Mistral	49.7	61.5	57.7	68.5	59.4	66.1	59.2	59.8	55.3	51.2	58.3	58.8

Table 5: Full results (%) on IFEval and FollowBench, where † and * indicate that the results are sourced from Sun et al. (2024) and Dong et al. (2024), respectively.

Model	Backbone	IFEval					FollowBench (HSR)						AVG
		[S]P	[S]I	[L]P	[L]I	AVG	L1	L2	L3	L4	L5	AVG	
Conifer$_{\text{SFT}}$-7B†	Mistral	45.8	57.1	50.8	62.0	53.9	54.3	49.5	49.3	40.8	30.5	44.9	49.4
Conifer$_{\text{SFT}}$-7B-v0.3	Mistral	45.8	57.0	49.7	60.8	53.3	60.6	52.2	46.7	38.8	26.5	45.0	49.1
Conifer$_{\text{DPO}}$-7B†	Mistral	48.1	59.1	52.3	63.3	55.7	60.3	53.6	48.0	47.1	41.0	50.0	52.9
Conifer$_{\text{DPO}}$-7B-v0.3	Mistral	46.4	57.2	54.9	64.6	55.8	60.1	52.5	46.6	45.7	38.6	48.7	52.2
Mistral$_{\text{CRAB}}$ + DPO	Mistral	49.7	61.5	57.7	68.5	59.4	66.1	59.2	59.8	55.3	51.2	58.3	58.8
Llama3.2 3B	Llama3.2	15.0	26.3	15.5	26.7	20.9	12.7	14.7	14.9	18.2	11.8	14.5	17.7
Llama3.2$_{\text{CRAB}}$	Llama3.2	34.9	44.6	38.1	48.1	41.4	51.7	36.4	29.1	19.7	14.3	30.2	35.8

Table 6: Experimental results (%) of the original Conifer paper, our reproduced results on Mistral 7B v0.3 (the backbone used in Mistral$_{\text{CRAB}}$) and the results of Llama3.2 as the backbone for IFEval and FollowBench. Here, † indicates that the results are from Sun et al. (2024).

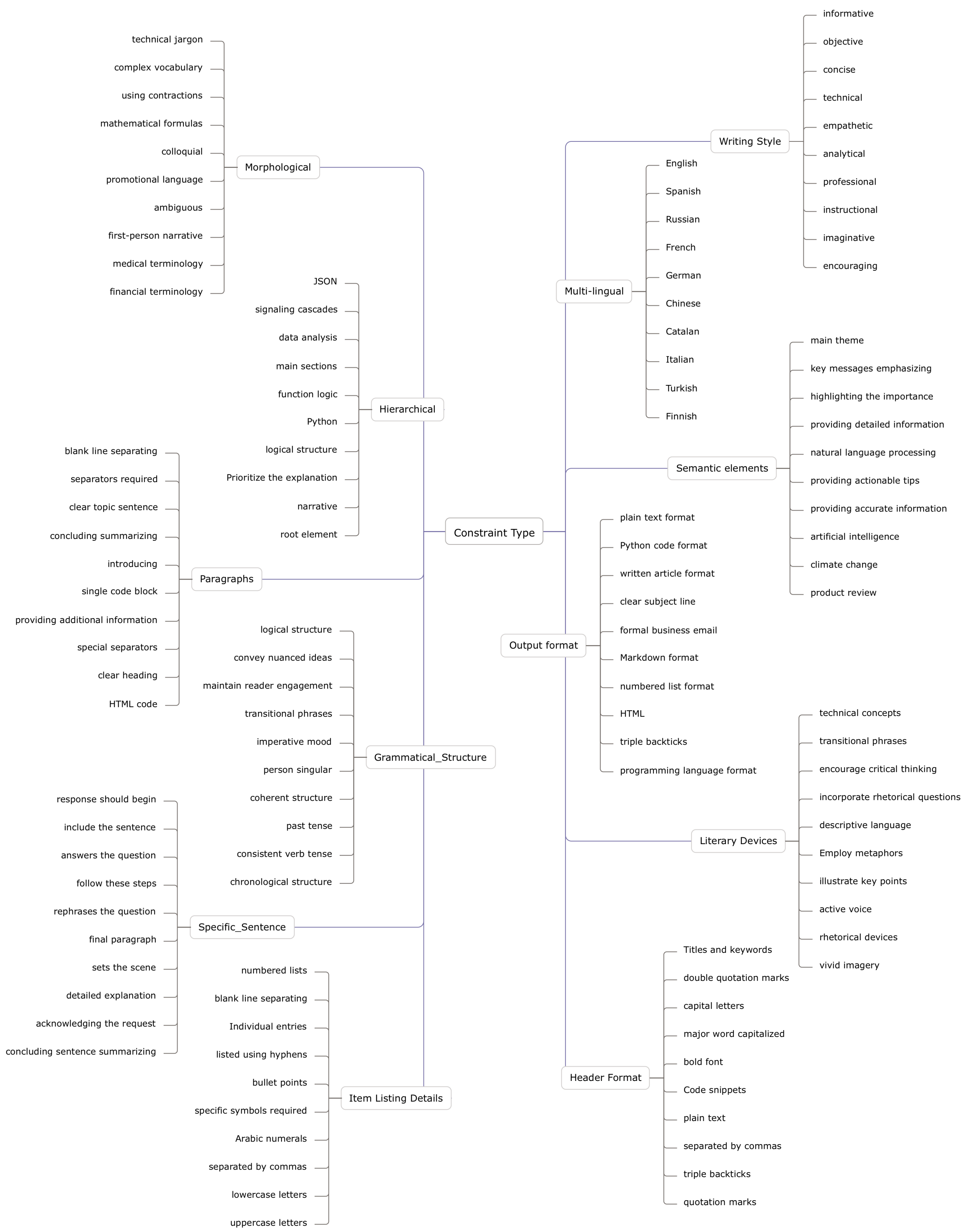


Figure 7: The Top-10 subcategories for 12 major constraint types (excluding "situation") generated by the LLM after constraint back-translation, as the "situation" type is closely tied to the input and lacks clustering tendencies.

Instruction

What is an easy J Perm PLL algorithm that wouldn't be too complicated for a beginner at speed solving?

Output

One easy J Perm PLL algorithm that is not too complicated for a beginner at speed solving is the following:

R' U R' U' J R U' R' U' J' U2 R U

This algorithm only has 10 moves and does not require any complicated finger tricks. Here is a breakdown of the algorithm:

1. R' U R' U' - This is a common setup move for J Perm PLL algorithms. It sets up the J shape in the corner of the cube.
2. J R U' R' U' - This is the J Perm algorithm itself. It swaps the two adjacent edges and the two adjacent corners in the J shape.
3. J' U2 R U - This is the inverse of the setup move and completes the PLL algorithm.

Remember to practice this algorithm slowly at first, and then gradually increase your speed. With enough practice, you should be able to execute this algorithm quickly and efficiently.

Additional Constraints

Situation: What is a simple, 10-move or fewer J Perm PLL algorithm for speed solving that a beginner can learn and execute efficiently, without requiring advanced finger tricks or complex sequences?

Length Constraint: Submit an response that contains at least 160 words.

Word Per Sentence: Restrict each sentence to 25 words maximum.

Characters Per Word: Each word in the response must consist of at least 15 characters.

Keyword Constraint: In your reply, include the terms c̈omplicated finger tricks,̈ P̈erm PLL,̈ P̈erm PLL algorithmẗo emphasize the critical components and provide clarity.

Punctuation Limitation: Avoid using question marks in your response.

Desired Writing Style: The response should be written in a clear and concise manner, with a tone that is informative and helpful, suitable for a beginner in speed solving.

Semantic Elements: The main theme of the response should focus on providing a simple J Perm PLL algorithm for beginners, highlighting its ease of use and simplicity.

Morphological Constraints: Avoid using overly technical jargon or complex mathematical notation, and refrain from using formatting styles that may confuse the reader.

Multi-lingual Constraints: The response should be written in English.

Specific Literary Devices: Use a step-by-step breakdown of the algorithm to enhance clarity and facilitate understanding.

Specific Grammatical Structure: The response should be written primarily in simple sentences, with occasional use of compound sentences to provide additional explanations.

Hierarchical Instructions: The response should prioritize explaining the algorithm, followed by a breakdown of the algorithm, and finally providing practice tips.

Paragraphs Constraints: The response should consist of three paragraphs, with a blank line separating each paragraph.

Specific Sentence: The response should start with a sentence that introduces the algorithm, and end with a sentence that encourages practice.

Item Listing Details: The breakdown of the algorithm should use numbers (1., 2., 3., etc.) to list the steps, with a hyphen and a space before each step description.

Table 7: An example from OpenAssistant of CRAB after constraint back-translation and before combination.