

# Automatically Enhanced Instruction-following Capabilities of Large Language Models via Execution Feedback

Anonymous ACL submission

## Abstract

One core capability of large language models (LLMs) is to follow natural language instructions. However, the issue of automatically constructing high-quality training data to enhance the complex instruction-following abilities of LLMs without manual annotation remains unresolved. In this paper, we introduce AUTOIF, the first scalable and reliable method for automatically generating instruction-following training data. AUTOIF transforms the validation of instruction-following data quality into code verification, requiring LLMs to generate instructions, the corresponding code to check the correctness of the instruction responses, and unit test samples to verify the code’s correctness. Then, execution feedback-based rejection sampling can generate data for Supervised Fine-Tuning (SFT) and Reinforcement Learning from Human Feedback (RLHF) training. AUTOIF achieves significant improvements across three training algorithms, SFT, Offline DPO, and Online DPO, when applied to the top open-source LLMs, Qwen2 and Llama3, in self-alignment and strong-to-weak distillation settings.

## 1 Introduction

The instruction-following ability of large language models (LLMs) refers to their capacity to understand, interpret, and execute commands given to them in natural language (Lou et al., 2023; OpenAI et al., 2024). This ability is fundamental to contemporary LLMs as it enables them to leverage their underlying knowledge, interact intuitively with users (Ouyang et al., 2022), adapt to various requirements (Zhang et al., 2023), and perform complex tasks (Sun et al., 2024). Misunderstandings in following instructions can lead to unintended outcomes, potentially resulting in severe consequences, particularly in critical scenarios (Zhou et al., 2023; Chang et al., 2024).

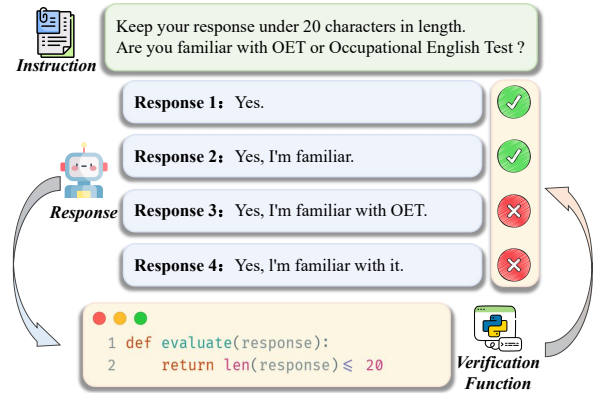


Figure 1: An example of the verification function automatically assesses the adherence of responses to the instruction’s constraints.

Although instruction following is crucial, scalable and reliable methods to enhance this capability of LLMs remain elusive. Current efforts in this field are divided into manual annotation (Wei et al., 2021; Zhou et al., 2023; Jiang et al., 2024b) and behavior imitation (Xu et al., 2023; Zhao et al., 2024). Manual annotation involves annotators designing instructions and writing corresponding responses. However, due to human cognition’s limitations, creating highly complex and diverse instructions is challenging, making the process difficult to scale. Furthermore, accurately executing complex instructions can sometimes be difficult for humans (Sun et al., 2024; Cao et al., 2024b), requiring multiple rounds of rigorous and costly validation (Wang et al., 2024a; Wei et al., 2024). On the other hand, behavior imitation aims to distill responses from more advanced LLMs (Taori et al., 2023; Peng et al., 2023) like GPT-4. This approach limits models to the capabilities of the advanced LLMs from which they are distilled. Moreover, even advanced LLMs can make mistakes, and the reliability of the distilled data cannot be guaranteed (Cui et al., 2023). Consequently, models trained with this data may have a propensity to not follow instructions accurately (Zhou et al., 2024).

In this paper, we introduce AUTOIF, the first scalable and reliable method for automatically generating instruction following training Data for Supervised Finetuning (SFT) or Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022). The core idea of AUTOIF is to use code to verify the correctness of following instructions. Intuitively, if designed properly, a significant portion of instructions, such as “Keep your response under 20 characters in length” can be verified for correctness using code, as illustrated in Fig. 1. Therefore, the key components of AUTOIF include (1) automatically generating instructions that can be verified by code, (2) automatically generating corresponding verification codes for these instructions, and (3) ensuring the reliability of the first two steps. Specifically, we start by providing AUTOIF with a small set of hand-written seed instructions. Then, LLMs, not necessarily advanced ones, generate an augmented instruction set through self-instruct (Wang et al., 2023). Next, LLMs write verification codes and unit test cases for each instruction. Only the code that compiles correctly, passes the test cases, and back-translates to the original instruction is retained. If an instruction does not have a corresponding code that can verify its correctness, it is discarded. Finally, we employ LLMs to generate responses that either pass or fail the verification code using execution feedback-based rejection sampling (Yuan et al., 2023). Responses that pass can be directly used for SFT, while pairs of passing and failing responses can be used to create chosen-rejected pairs for Direct Preference Optimization (DPO) (Rafailov et al., 2023) and other RLHF algorithms. Moreover, once the instructions and verification code are determined, this process can be conducted on-policy, continually enhancing the instruction-following capabilities.

Through extensive experiments, we have demonstrated that AUTOIF achieves significant improvements under three training algorithms—SFT, Offline DPO, and Online DPO—when applied to the top open-source LLMs, Qwen2-72B and Llama3-70B, in both self-alignment and strong-to-weak distillation settings. In the IFEval benchmark, we achieved Loose Instruction (Acc.) rates of up to 88.0% with Qwen2-72B and 90.4% with Llama3-70B, marking the first instance of surpassing 90% accuracy. In the FollowBench benchmark, these models also showed significant improvements, with increases of over 5% in the SSR metric (avg). Additionally, they enabled Qwen2-7B and Llama3-8B

to achieve average performance gains of over 4% in both benchmarks. Replacing Qwen2-72B and Llama3-70B with the more advanced GPT-4 resulted in further substantial improvements. We will open-source the SFT and DPO datasets constructed using AUTOIF on Qwen2-72B, representing the first open-source complex instruction-following dataset at a scale of tens of thousands.

## 2 Related Works

**Instruction-following capabilities** are among the most essential features of LLMs (OpenAI et al., 2024; Lou et al., 2023), which are expected to precisely follow a broad and complex set of instructions. Consequently, recent research has concentrated on evaluating LLMs’ instruction-following abilities in various contexts, such as verifiable (Zhou et al., 2023), compositional (Qin et al., 2024), format-related (Xia et al., 2024), refuting (Yan et al., 2024), and fine-grained instructions (Jiang et al., 2024b). However, a significant gap remains between open-source and proprietary closed-source LLMs. Sun et al. (2024) propose Conifer, which enhances the instruction-following capabilities of open-source LLMs through knowledge distillation from proprietary LLMs. Wang et al. (2024b) use LLMs to encode instruction metadata and augment diverse instructions from this metadata, employing proprietary LLMs for quality control. Both approaches, however, rely on proprietary LLMs for response distillation or judgment, which not only limits their potential but also subjects them to OpenAI’s terms of use<sup>1</sup>. In this work, we propose AUTOIF, a more scalable and reliable method to enhance the instruction-following capabilities of LLMs. AUTOIF uses execution feedback from self-generated verification functions to provide supervision for instructions. This allows for effective self-alignment and strong-to-weak distillation on open-source models, thereby narrowing the performance gap with proprietary LLMs.

**Learning with Execution Feedback** is a widely-used technique in automated alignment for tool use and coding (Cao et al., 2024a). These learning methods typically utilize execution feedback from tools such as code executors to provide supervision for specific tasks. For instance, Le et al. (2022) employ feedback from unit tests via code compilers to enhance code synthesis capabilities through reinforcement learning. Similarly, Chen

<sup>1</sup><https://openai.com/policies/terms-of-use>

et al. (2023) train LLMs to provide debugging suggestions as feedback to improve coding abilities. Additionally, Qiao et al. (2024) introduce Reinforcement Learning with execution feedback to enhance LLMs using execution results from tools. Building on this learning paradigm, we propose a novel scalable oversight method that enables LLMs to autonomously generate verification functions and unit tests for natural language instructions, thereby applying execution feedback to enhance their instruction-following capabilities.

### 3 AutoIF

We introduce AUTOIF, an automated, scalable, and reliable method designed to enhance the instruction-following capabilities of LLMs. In this section, we outline the preliminaries (§3.1), detail the two core components of AUTOIF (§3.2, §3.3), and discuss various training strategies that can be seamlessly integrated with AUTOIF (§3.4).

#### 3.1 Preliminaries

**Instruction-following Capabilities.** Following instructions is one of the most crucial skills in modern LLMs. These models are expected to provide precise responses to queries containing complex instructions, which can be either atomic or compositional. To evaluate the instruction-following capability of LLMs, we define a general instruction-following requirement as a specific task. In this task, given an instruction  $I = \{i_j\}_{j=1}^N$  with  $N$  specific constraints (e.g. “Please generate text in Shakespearean style, no more than 50 tokens” contains 2 constraints) and a specific query  $x$ , an LLM  $\pi_\theta$  should generate precise response  $y \sim \pi_\theta(y | x, I)$  adhering to the constraints.

**Verifiable Instructions.** The complexity and diversity of instructions necessitate manual construction and verification for reliable supervision. This practical challenge motivates us to focus initially on instructions that can be automatically verified through programs and code executors, also known as verifiable instructions (Zhou et al., 2023). Specifically, for a given instruction  $I$  and task-specific query  $q$ , there exists a verification function  $f_I$  such that  $f_I(y)$  returns true when the model’s response  $y$  correctly follows the instruction. We demonstrate that supervision of such instructions can be self-generated through scalable oversight with LLMs and execution feedback. Extensive experiments in our work show that training on verifiable instruc-

tions significantly benefits the handling of other general instructions that are more complex but unverifiable with simple code snippets.

**Method Overview.** AUTOIF synthesizes high-quality instruction-following data through self-evolution, rejection sampling, and execution feedback. As illustrated in Fig. 2, AUTOIF integrates automated data augmentation with quality verification processes, including automatically generated verification functions and back-translation instructions. This approach enables a two-stage automated data synthesis at both the instruction (§3.2) and query levels (§3.3). Additionally, we introduce three training strategies (§3.4) and explore two experimental settings (§5) to thoroughly evaluate the effectiveness and generalization of AUTOIF.

#### 3.2 Instruction Augmentation and Verification

We first develop verifiable instructions along with corresponding evaluation functions, using rejection sampling informed by execution feedback.

**Seed Instruction Construction.** We start by hand-writing a set of seed instructions, denoted as  $D_{seed}$ , ensuring that each instruction contains only a single atomic constraint (e.g., “Answer the words that begin with B”). Detailed information on seed instructions is listed in Appx. §A.

**Self-Instruct.** Self-Instruct (Wang et al., 2023) is a straightforward and intuitive strategy for automated data augmentation that has garnered significant attention in the field of LLM reasoning (Xu et al., 2023; Zhao et al., 2023). For each instruction in  $D_{seed}$ , we use an LLM to perform  $K$  instruction rewrites, generating  $D_{aug}$ . We then combine the seed and augmented data sets to obtain an enhanced set of instructions,  $D_{ins} = D_{seed} \cup D_{aug}$ , and remove any duplicates.

**Automated Quality Cross Verification.** Previous research has shown that relying solely on model-generated augmented instructions often leads to the inclusion of low-quality samples (Mumuni and Mumuni, 2022; Xie et al., 2020; Zheng et al., 2024). Inspired by a series of tool execution studies, we employ an LLM to generate verification functions and test cases for each instruction. We use feedback from executing Python programs to ensure quality control. Given the instruction set  $D_{ins}$ , the LLM  $M$  employs a rejection sampling (Touvron et al., 2023; Yuan et al., 2023) to generate  $K$  verification functions  $f_I = \{f_i\}_{i=1}^K$  and test cases  $c_I = \{c_i\}_{i=1}^K$  for each instruction  $I$ , resulting in

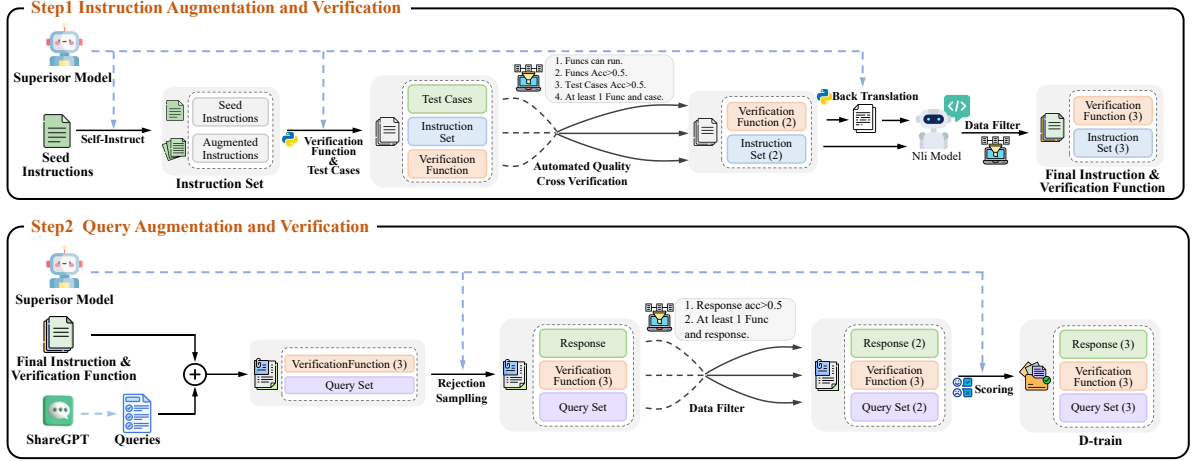


Figure 2: An Overview of AUTOIF: A Two-Stage Automated Instruction-Following Data Synthesis Method.

the set  $\{I, f_I, c_I\} \in D_{ins}$ . We then cross-validate the quality of the instructions using the verification functions and test cases, ensuring they meet the following criteria:

- The verification function  $f \in f_I$  can be successfully compiled by the Python executor.
- Each test case  $c \in c_I$  achieves an accuracy rate greater than 0.5 across all verification functions.
- Each verification function  $f \in f_I$  achieves an accuracy rate greater than 0.5 across all test cases.
- Each instruction includes at least one evaluation function and test case.

By adhering to these four conditions, we obtain the quality-filtered instruction set  $\{I^{(2)}, f_I^{(2)}\} \in D_{ins}^{(2)}$ .

**Back-translation Verification.** After the cross-validation stage, we obtained initially quality-verified verification functions and instructions. To further ensure the consistency between instructions and verification functions, we introduce back-translation. For a given pair  $\{I^{(2)}, f_I^{(2)}\} \in D_{ins}^{(2)}$ , we use the LLM  $M$  to back-translate the verification function  $f \in f_I^{(2)}$  into instruction  $I_f$ . We then treat  $I$  as the *premise* and the back-translated instruction  $I_f$  as the *hypothesis*. Using the NLI model, we identify the semantic relationship between the two instructions. The prediction can fall into one of three categories: *entailment*, *contradiction*, or *neutral*:

$$p_{\theta}(\cdot | q, q_{aug}) = \text{softmax}(\text{score}_{\theta}(I, I_f)), \quad (1)$$

where  $\text{score}_{\theta} : \mathbb{R}^{k \times \ell_I} \times \mathbb{R}^{k \times \ell_{I_f}} \rightarrow \mathbb{R}^3$  is a model dependent scoring function with parameters  $\theta$ . We filter out any instruction  $I$  labeled as *contradiction* to ensure the intent consistency. Finally we obtain the set  $\{I^{(3)}, f_I^{(3)}\} \in D_{ins}^{(3)}$

### 3.3 Query Augmentation and Verification

Once we have obtained verified instructions and verification functions, we utilize them to create training data comprising queries and responses.

**Query Reforming and Augmentation.** In the real-world application of modern chatbots, instructions are typically employed to generate constrained responses to user queries. Therefore, creating high-quality instructions is merely the initial step toward achieving effective instruction-following capabilities. To acquire authentic queries, as shown in the bottom part of Fig. 2, we randomly selected  $K$  user queries from ShareGPT (Chiang et al., 2023) for each instruction and concatenated them to construct the seed query dataset  $x, f_I^{(3)} \in D_q$ . To further enhance the diversity and complexity of the input  $x$ , we utilized the LLM to generate  $K$  responses  $y_x = \{y_i\}_{i=1}^K$ , resulting in  $\{x, f_I^3, y_x\} \in D_q$ .

**Instruction-following Verification.** Following the previous quality cross-verification process, we further employ verification functions to assess whether the augmented responses adhere to the constraints in input  $x$ . Similarly, we require each response in  $D_q$  to meet the following conditions:

- Each response must achieve an accuracy rate greater than 0.5 across all verification functions.
- Each input must include at least one verification function and one response.

Based on these rules, we obtain the set  $(x^{(2)}, f_I^{(3)}, y^{(2)}) \in D_q^{(2)}$ .

**Query Quality Verification.** Additionally, we observe that concatenated instructions and queries often conflict. For instance, a high-quality response to the query “help me write a news article” is unlikely to comply with the instruction “please limit

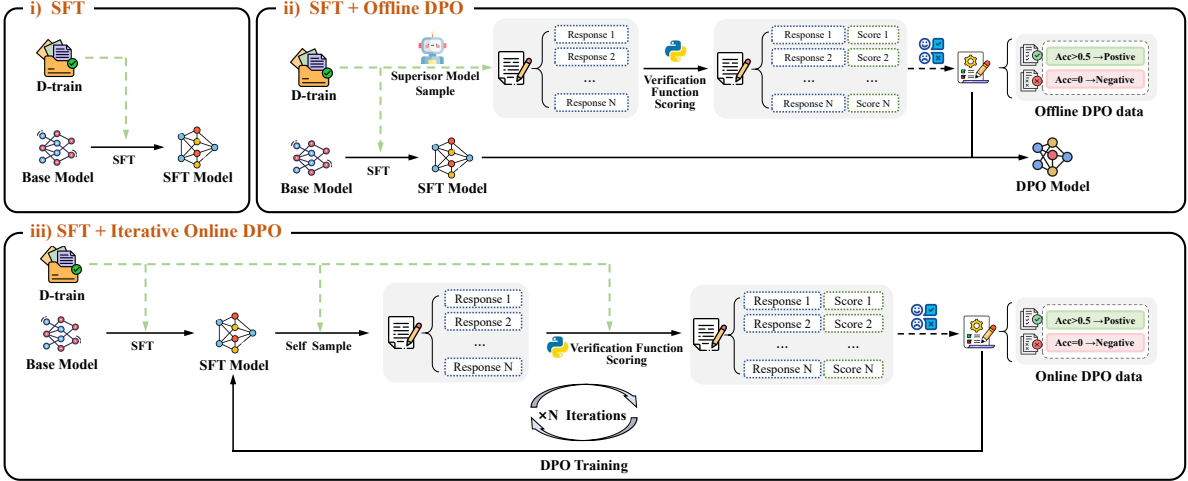


Figure 3: Different training strategies that can be adapted with synthetic dataset generated by AUTOIF.

your answer to two words”. Such high-level semantic inconsistencies are challenging for a simple NLI model to discern. Therefore, we employ the LLM  $M$  to assign matching scores between the instruction and query in input  $x^{(2)}$  and the corresponding responses  $y^{(2)}$ , on a scale from 1 to 10. We then filter out samples with a score lower than 8, constructing the final training set  $D_{\text{train}} = \{x_i, y_i, fI_i\}_{i=1}^N$ .

### 3.4 Training Strategies

AUTOIF offers multifaceted supervision for the instruction-following task, making it adaptable to various training strategies. To thoroughly evaluate the effectiveness of AUTOIF, we propose the following training approaches:

**Supervised Fine-tuning (SFT).** Given  $(x_i, y_i) \in D_{\text{final}}$ , we apply the standard Supervised Fine-tuning (SFT) objective on the base model  $P$  with parameters  $\theta$ :  $\mathcal{L}(\theta) = \sum_{(x_i, y_i) \in D_{\text{train}}} \log \mathbb{P}_{\theta}(y_i | x_i)$ , where  $x_i$  denotes the  $i$ -th input, consisting of a concatenated instruction and user query.

**SFT + Offline DPO.** In the process of AUTOIF, multiple scales of quality filtering are utilized, naturally generating a substantial number of positive and negative sample pairs. This motivates us to obtain pairwise preference data  $(x, y_w, y_l)$ . Our preference data mining is divided into two parts:

- **Instruction Level:** During the automated quality cross-verification stage, we first extract positive samples  $c_w$  from cases with an accuracy rate higher than 0.5 on all verification functions and negative samples  $c_l$  from cases with an accuracy rate of 0. We then construct pairwise preference data for each instruction:  $D_{\text{ins}}^{\text{pref}} \rightarrow (I, c_w, c_l)$ .
- **Query Level:** In the query quality verification process, we similarly extract positive samples

$y_w$  from responses with an accuracy rate higher than 0.5 on all verification functions and negative samples  $y_l$  from responses with an accuracy rate of 0. We then construct query preference data:  $D_{\text{query}}^{\text{pref}} \rightarrow (x, y_w, y_l)$ .

Finally, we merge the two parts of the data:  $D_{\text{pref}} = D_{\text{ins}}^{\text{pref}} \cup D_{\text{query}}^{\text{pref}}$ . To further explore the potential of pairwise preference data  $(x, y_w, y_l) \in D_{\text{pref}}$ , we first perform vanilla SFT on the base model  $\pi_{\theta}$  to obtain an SFT model  $\pi_{\theta}^{\text{SFT}}$  as equation 3.4. Then, we apply Direct Preference Optimization (DPO) (Rafailov et al., 2024) on our SFT model, which can be formulated as follows:

$$\mathcal{L}_{\text{DPO}}(\pi_{\theta}^{\text{SFT}}; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} [\log \sigma(\beta \log \frac{\pi_{\theta}^{\text{SFT}}(y_w | x)}{\pi_{\theta}^{\text{SFT}}(y_l | x)} - \beta \log \frac{\pi_{\text{ref}}(y_l | x)}{\pi_{\text{ref}}(y_w | x)})], \quad (384)$$

where the reference model  $\pi_{\text{ref}}$  is set to  $\pi_{\theta}^{\text{SFT}}$  initially and remains fixed throughout training.  $\beta$  is a hyperparameter and  $\sigma$  is the sigmoid function.  $\mathcal{L}_{\text{DPO}}$  aims to maximize the log probability of preferred  $y_w$  relative to the dispreferred  $y_l$ .

**SFT + Iterative Online DPO.** Online training enables real-time, iterative optimization of model weaknesses. It relies on high-quality, lightweight reward models to provide continuous supervision feedback. In the case of AUTOIF, verification functions serve as rigorous filtering standards, akin to reward models, delivering immediate feedback on model responses across training iterations. Following offline DPO, we conduct initial SFT on the base model  $\pi_{\theta}$  to derive an SFT model  $\pi_{\theta}^{\text{SFT}}$  with initial instruction-following capabilities. As depicted in Fig. 3, we set the generation temperature to 0.8 and allow the SFT model to generate

404  $K$  responses through self-sampling for each training  
405 sample, forming a response set  $\{R_1, \dots, R_k\}$ .  
406 Then, we employ corresponding verification functions  
407 to assess  $K$  responses, thereby constructing  
408 the online DPO dataset  $D_{\text{online}}^{\text{pref}} = (x, y_w, y_l)$  based  
409 on average pass rates across all functions. Finally,  
410 leveraging  $D_{\text{online}}$ , we sequentially perform DPO  
411 training on  $\pi_{\theta}^{\text{SFT}}$ . Importantly, our iterative on-  
412 line optimization process progressively unlocks en-  
413 hanced instruction-following capabilities.

## 414 4 Experiment

### 415 4.1 Experimental Setup

416 **Datasets & Baselines.** We conduct experi-  
417 ments using two LLMs from the Qwen2 series  
418 (Qwen2-7B and Qwen2-72B-Instruct) and two  
419 from the Llama3 series (Llama3-8B and Llama3-  
420 70B-Instruct). Please note that the AUTOIF method  
421 proposed in this work has been employed in the  
422 open-source Qwen2-Instruct model. Thus, the ver-  
423 sion of Qwen2-Instruct we utilized represents an  
424 early iteration during internal development, rather  
425 than the final open-source model. The training  
426 datasets are respectively generated from Qwen2-  
427 72B-Instruct and Llama3-70B-Instruct, with de-  
428 tailed statistics provided in Tab. 5. We demonstrate  
429 the effectiveness of AUTOIF by evaluating the  
430 instruction-following capabilities of models fine-  
431 tuned with self-generated datasets using AUTOIF.  
432 Additionally, we include strong open and closed-  
433 source LLM baselines such as Mixtral-8x22B and  
434 GPT-4. For more details, refer to Appx. §B.

435 **Settings.** we explore two experimental setups: (1)  
436 **Strong-to-Weak Distillation** involves aligning a  
437 less powerful model with a stronger, well-aligned  
438 model by mimicking its generated responses. In  
439 AUTOIF, we can utilize a strong model such as  
440 Qwen2-72B-Instruct for data synthesis. Subse-  
441 quently, we train a less powerful model like Qwen2-  
442 7B-Instruct using this synthesized data to achieve  
443 strong-to-weak alignment. (2) **Self-Alignment:**  
444 Following several self-alignment works (Chen  
445 et al., 2024; Yuan et al., 2024), we utilize the LLM  
446 to perform the AUTOIF process for synthesizing  
447 data, and then train the same model using this syn-  
448 thesized data.

449 **Evaluation.** We evaluate our methods using two  
450 instruction-following benchmarks: IFEval (Zhou  
451 et al., 2023) and FollowBench (Jiang et al., 2024b).  
452 IFEval comprises 25 types of verifiable instruc-

453 tions across about 500 prompts. While IFEval  
454 also focuses on verifiable instructions, extensive  
455 n-gram probing confirms no overlap between the  
456 IFEval test set and our training sets, thus eliminat-  
457 ing any contamination concerns. We report strict  
458 and loose accuracy metrics at both prompt and in-  
459 struction levels for IFEval. FollowBench is a fine-  
460 grained constraint-following benchmark with five  
461 levels of difficulty. It contains diverse and open-  
462 ended instructions requiring evaluation by strong  
463 LLMs, such as GPT-4, which can fully examine  
464 the generalization of AUTOIF to more general in-  
465 structions not verifiable by simple code executions.  
466 At the same time, we also evaluate our models on  
467 C-Eval (Huang et al., 2023), MMLU (Hendrycks  
468 et al., 2021), GSM8k (Cobbe et al., 2021), and  
469 HumanEval (Chen et al., 2021a) to obtain a com-  
470 prehensive assessment of capabilities.

### 471 4.2 Main Results

472 Tab. 1 reports the main results. Overall, AUTOIF  
473 substantially enhances instruction-following perfor-  
474 mance across all models, configurations (strong-to-  
475 weak distillation & self-Alignment), and training  
476 methodologies (SFT, Offline & Online DPO) on  
477 two benchmarks. These results decisively establish  
478 the superiority of our approach. Furthermore, we  
479 have identified the following insights:

480 **On-policy Learning is More Effective.** Compar-  
481 ing Online DPO and Offline DPO, the model-  
482 generated online data through self-supervision  
483 demonstrates superior performance compared to  
484 offline data (Qwen2-7B, IFEval: 1.7% $\uparrow$ , Follow-  
485 bench: 2.6% $\uparrow$ ). This confirms that on-policy itera-  
486 tive execution feedback can effectively target and  
487 enhance the model’s weaknesses.

488 **Larger models yield greater improvements.**  
489 FollowBench provides a more comprehensive  
490 instruction-following assessment than IFEval. Sig-  
491 nificantly, base models with larger parameters typi-  
492 cally improve Followbench more than smaller mod-  
493 els (Qwen2 72B: 4.6% $\uparrow$ , Llama3 70B: 5.6% $\uparrow$ ).  
494 This underscores that models with robust founda-  
495 tional capabilities coupled with AUTOIF, can fur-  
496 ther unlock powerful instruction-following align-  
497 ment potential.

498 **General abilities are not declined.** Improving  
499 instruction following abilities without compromis-  
500 ing other capabilities is crucial. AUTOIF notably  
501 preserves general abilities (MMLU, C-Eval), math-  
502 ematical reasoning (GSM8k), and coding (Hu-

Model	IFEval				FollowBench (SSR)						C-Eval	MMLU	GSM8k	HumanEval
	Pr (S)	Pr. (L)	Ins. (S)	Ins. (L)	Level 1	Level 2	Level 3	Level 4	Level 5	Avg				
<i>Baselines (&lt; 10B)</i>														
Qwen2-7B	37.7	43.6	49.4	53.4	55.6	53.5	53.7	49.9	48.6	52.3	74.4	64.4	71.1	58.1
Qwen2-7B (ShareGPT)	30.9	33.5	42.4	45.2	56.1	52.7	50.8	45.2	47.9	50.5	70.2	59.8	59.4	52.4
Llama3-8B	24.6	26.1	38.1	39.7	10.0	10.3	10.5	14.3	12.7	11.6	24.2	38.8	4.5	0.6
Llama3-8B (ShareGPT)	23.7	26.4	33.8	37.1	44.0	40.0	39.6	33.3	33.6	38.1	70.2	59.8	59.4	52.4
Mistral-7B	23.3	24.6	38.4	39.6	40.1	39.7	37.9	35.7	36.7	38.0	38.2	47.6	20.5	38.4
<i>Baselines (&gt; 10B)</i>														
Qwen2-72B-Instruct	77.1	80.4	84.4	86.9	70.2	66.6	63.5	58.1	56.3	62.9	83.8	80.8	87.9	73.8
Llama3-70B-Instruct	77.8	83.8	84.2	88.8	60.7	60.5	61.1	61.7	60.3	60.9	60.2	80.5	92.6	78.7
Mixtral-8x22B	41.8	47.3	55.2	60.0	63.9	60.0	58.2	56.2	55.3	58.7	-	-	-	-
GPT-4 <sup>†</sup>	76.9	79.3	83.6	85.4	84.7	77.6	76.2	77.9	73.3	77.9	-	-	-	-
GPT-3.5 Turbo <sup>†</sup>	-	-	-	-	80.3	71.2	74.2	69.6	67.1	72.5	-	-	-	-
<b>Supervision Model: Qwen2-72B</b>														
<i>Strong-to-Weak</i>														
Qwen2-7B-SFT	40.7(+3.0)	44.5(+0.9)	51.3(+1.9)	55.4(+2.0)	60.2(+4.6)	53.7(+0.2)	54.3(+0.6)	49.9(+0.0)	48.6(+0.0)	53.3(+1.0)	73.9(+0.0)	64.4(+0.0)	74.1(+3.0)	58.3(+0.2)
w/ Offline DPO	41.2(+3.5)	44.7(+1.2)	51.4(+2.0)	56.2(+2.8)	61.4(+5.8)	54.5(+1.0)	54.3(+0.6)	51.2(+1.3)	48.6(+0.0)	54.0(+1.7)	75.1(+0.7)	64.5(+0.1)	72.9(+1.8)	59.5(+1.4)
w/ Online DPO	44.0(+6.3)	46.6(+3.0)	55.0(+5.6)	57.9(+4.5)	61.4(+5.8)	56.8(+3.3)	57.8(+4.1)	55.4(+5.5)	51.6(+3.0)	56.6(+4.3)	76.0(+1.6)	64.8(+0.4)	72.3(+1.2)	58.2(+0.1)
<i>Self-Alignment</i>														
Qwen2-72B-Instruct	80.2(+3.1)	82.3(+1.9)	86.1(+1.7)	88.0(+1.1)	76.2(+6.0)	69.8(+3.2)	67.0(+3.5)	61.6(+3.5)	62.8(+6.5)	67.5(+4.6)	84.9(+1.1)	81.2(+0.4)	88.2(+0.3)	75.0(+1.2)
w/ Online DPO														
<b>Supervision Model: Llama3-70B</b>														
<i>Strong-to-Weak</i>														
Llama3-8B-SFT	28.7(+4.1)	40.3(+14.2)	41.4(+3.3)	52.2(+12.05)	46.6(+36.6)	46.2(+35.9)	45.9(+35.4)	37.6(+23.3)	41.0(+28.3)	43.5(+31.9)	34.5(+10.3)	45.6(+6.8)	33.2(+28.7)	38.2(+37.6)
w/ Offline DPO	27.9(+3.3)	41.6(+15.5)	40.5(+2.4)	54.1(+14.4)	51.9(+41.9)	51.3(+41.0)	50.1(+39.6)	45.3(+31.0)	47.9(+34.8)	49.2(+37.6)	36.2(+12.0)	45.3(+6.5)	31.9(+27.4)	38.5(+37.9)
w/ Online DPO	28.8(+4.2)	43.1(+17.0)	42.2(+4.1)	56.0(+16.3)	54.6(+44.6)	52.1(+41.8)	50.0(+39.5)	49.0(+34.7)	43.7(+31.0)	49.9(+38.3)	38.2(+14.0)	45.1(+6.3)	32.5(+28.0)	38.4(+37.8)
<i>Self-Alignment</i>														
Llama3-70B	80.2(+2.4)	85.6(+1.8)	86.7(+2.5)	90.4(+1.6)	71.0(+10.3)	67.2(+6.7)	66.2(+5.1)	64.6(+2.9)	63.5(+3.2)	66.5(+5.6)	61.6(+1.4)	80.7(+0.2)	92.7(+0.1)	78.7(+0.0)
w/ Online DPO														

Table 1: The main results on two instruction-following and four general benchmarks. Pr. and Ins. stand for prompt and instruction levels, respectively. S and L represent strict and loose metrics for IFEval. The highest accuracy for each setup is highlighted in green. Results marked with <sup>†</sup> are directly sourced from the original benchmarks.

Model	IFEval		FollowBench (SSR)
	Prompt(L)	Instruction(L)	Avg
Qwen2-7B	43.6	53.4	52.3
<i>Supervision Model: Qwen2-72B</i>			
+SFT	44.5(+0.9)	55.4(+2.0)	53.3(+1.0)
+SFT & Offline DPO	44.7(+1.1)	56.2(+2.8)	54.0(+1.7)
+SFT & Online DPO	46.6(+3.0)	57.9(+4.5)	56.6(+4.3)
<i>Supervision Model: GPT-4</i>			
+SFT	52.9(+9.3)	62.6(+9.2)	55.1(+2.8)
+SFT & Offline DPO	59.3(+15.7)	68.9(+15.5)	54.4(+2.1)
+SFT & Online DPO	59.5(+15.9)	69.4(+16.0)	55.7(+3.4)

Table 2: Ablation study on supervision models.

Model	IFEval		FollowBench (SSR)
	Prompt(L)	Instruction(L)	Avg
<i>Supervision Model: Qwen2-72B</i>			
Qwen2-7B-SFT-OnlineDPO	46.6	57.9	56.6
w/o Back-translation	-0.8	-1.7	-0.7
w/o Query Quality Verification	-1.4	-2.4	-1.3
w/o Cross Verification	-1.6	-3.0	-1.5
w/o All Quality Process	-2.2	-3.8	-2.6

Table 3: Ablation study on specific components of AUTOIF.

maneval) performance across all training setups. Surprisingly, there are even slight performance gains in on-policy settings. We attribute this preservation largely to incorporating ShareGPT data during data synthesis, highlighting AUTOIF’s capability to strike a balance across diverse abilities and excel in broad applicability.

### 4.3 Quality Ablation Study

**Ablation on Supervision Model.** Tab. 2 presents the results of replacing the supervision model Qwen72B with GPT-4. We observe that in AUTOIF, a stronger supervision model (GPT-4) demonstrates more effective strong-to-weak distillation alignment, particularly evident with a performance gain of over 15% in the loose prompt in IFEval. This is reasonable because AutoIF requires the supervision model to perform several tasks, such as text augmentation (instruction, query,

and response rewriting), code generation (verification function), and quality assessment (scoring). This implies that a supervision model with stronger fundamental abilities can synthesize higher-quality data when using AUTOIF.

### Quality Control on Instructions and Responses.

In Fig. 4, we examine how varying pass rate thresholds of verification functions (indicative of data quality) affect the amount of SFT data and instruction-following performance. As the pass rate threshold increases, the amount of SFT data decreases at the instruction level, while model performance consistently improves. This suggests that the quality of instructions is a crucial factor influencing IF performance. At the query level, the SFT data amount also decreases with higher pass rate thresholds. Notably, performance peaks at a pass rate of 0.8 and declines beyond 1. This observation aligns with our expectations, indicating a trade-off

Setup	Benchmark	Train Size	Test Size	Rephrase	Percentage↓	N-gram↓
ShareGPT	IFEval	25K	542	0	0.01%	4.8%
	Followbench	25K	820	1	0.01%	2.3%
Qwen 72B	IFEval	10K	542	2	0.01%	3.5%
	Followbench	12K	820	1	0.01%	0.9%
Llama3 72B	IFEval	15K	542	0	0.01%	2.9%
	Followbench	17K	820	1	0.01%	1.2%
GPT4	IFEval	25K	542	0	0.01%	3.6%
	Followbench	25K	820	1	0.01%	1.5%

Table 4: Contamination analysis on SFT data generated by different supervision models using AUTOIF.

Supervision	Total Data	SFT Data	DPO Data	Pass Rate	MBPP (Code)	IFEval
Llama3-70b	85K	15K	6k	26%	70.4	43.1
Qwen2-72b	123K	10K	4K	28%	73.9	44.7
GPT4	210k	25K	15K	34%	87.5	59.3

Table 5: Data statistics and efficiency. Pass Rate denotes the samples with a query quality score above 8 (%).

between data quality and quantity.

**Ablation on Specific Components.** To investigate the effectiveness of various modules in AUTOIF, we conduct an ablation study, as presented in Tab. 3. we use *w/o* to denote the variant *without* a specific module. The results reveal the following: (1) The performance of AUTOIF declines when any quality filtering process is removed, indicating that all components are highly effective. (2) The most significant performance drop occurs when the *Cross Verification* of instructions is removed, highlighting its importance over query quality verification. This underscores that a high-quality instruction set is fundamental to the AUTOIF process. (3) Eliminating the overall quality filtering process results in a more substantial performance drop than removing any single component, suggesting that quality filtering at both the instruction and query levels provides a mutually reinforcing effect.

#### 4.4 Analyses

**Contamination Analysis.** We evaluate the contamination of the training dataset generated by AUTOIF on IFEval and FollowBench. Specifically, we employ contamination detectors from LM-Sys (Yang et al., 2023), which utilize advanced chatbots to identify potentially rephrased contaminated test samples. Additionally, we report contamination findings detected by traditional n-gram contamination algorithms. As shown in Tab. 4, both contamination rates are lower than those of the ShareGPT dataset we used. This allows us to confidently assert that there is no contamination between the self-generated training samples and the test sets. More cases can be viewed in Appx. §D, **Data Efficiency.** Tab. 5 explores the relationship

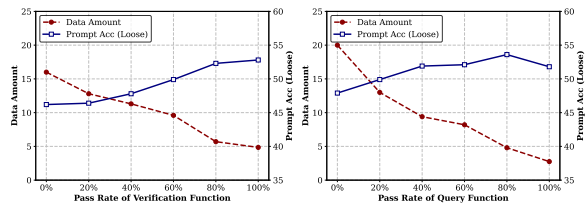


Figure 4: Quality ablation on instructions and queries.

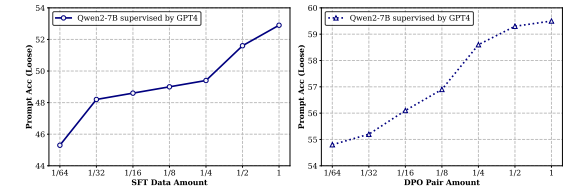


Figure 5: Scaling analysis on SFT data and DPO pairs

between model coding ability, data quality pass rate, and instruction-following capability. Surprisingly, we observe consistency in the supervision model across all three metrics. This indicates that the execution feedback resulting from the supervision model’s coding ability substantially influences data synthesis quality and the final capability.

**Scaling Analysis on SFT & DPO Data.** Fig. 5 presents the scaling analysis of SFT and DPO data using GPT-4 as the supervision model. The results demonstrate that even with just 1/64 of AUTOIF-generated SFT/DPO data, Qwen2-7B achieves impressive performance, particularly with 1/64 DPO data reaching nearly 55% in loose prompt accuracy, an increase of 11.4% pts. This strongly verifies the high quality of AUTOIF-generated data. Further analysis reveals that IF capability steadily improves with an increase in data quantity, a scaling trend confirmed by numerous reasoning studies (Yuan et al., 2023; Muennighoff et al., 2024).

## 5 Conclusion

In this paper, we propose AUTOIF, a scalable and automated method to enhance the instruction-following abilities of LLMs. It uses self-instruct and rejection sampling to enhance the supervisory signals of seed instructions and relies on self-generated execution feedback for quality filtering. We introduce three training strategies and two alignment settings to comprehensively analyze AUTOIF. Experiments demonstrate that our method significantly improves performance across all settings in both IFEval and Followbench, with the first LLM achieving over 90% loose prompt accuracy.



## 608 Limitations

609 In this paper, we propose AUTOIF, a system for  
610 automated instruction augmentation and quality  
611 filtering, capable of scaling to over 10,000 instruc-  
612 tions. While our focus is not on the construction  
613 of cross-instructions, the excellent results achieved  
614 in two instruction-following benchmarks demon-  
615 strate the generalizability of our method in han-  
616 dling complex instruction-following tasks. Addi-  
617 tionally, we believe a more direct strategy would  
618 involve combining multiple simple instructions into  
619 cross-instructions, and subsequently enhancing and  
620 quality-filtering them using AUTOIF. This way has  
621 the potential to further amplify the effectiveness of  
622 our method. Therefore, we consider automating  
623 and scaling cross-instruction tasks as a key direc-  
624 tion for future research.

## 625 Ethic Consideration

626 In this paper, we have fully presented the seed in-  
627 struction set used by AUTOIF in the Appendix. All  
628 concatenated queries are sourced from the publicly  
629 available ShareGPT dataset and have undergone  
630 multiple steps of quality filtering. Therefore, our  
631 method strives to minimize potential safety and eth-  
632 ical risks as much as possible. However, during the  
633 rejection sampling process, malicious prompts can  
634 lead the model to produce harmful or inappropriate  
635 outputs, which is a shared problem. Ensuring the  
636 quality of generated content in a safe and control-  
637 lable manner is crucial. The application of these  
638 techniques should be guided by ethical considera-  
639 tions, with safeguards in place to prevent misuse  
640 and reduce the likelihood of producing harmful  
641 outcomes.

## 642 References

643 Josh Achiam, Steven Adler, Sandhini Agarwal, Lama  
644 Ahmad, Ilge Akkaya, Florencia Leoni Aleman,  
645 Diogo Almeida, Janko Altenschmidt, Sam Altman,  
646 Shyamal Anadkat, et al. 2023. Gpt-4 technical report.  
647 *arXiv preprint arXiv:2303.08774*.

648 Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang,  
649 Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei  
650 Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin,  
651 Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu,  
652 Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren,  
653 Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong  
654 Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang  
655 Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian  
656 Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi  
657 Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang,

658 Yichang Zhang, Zhenru Zhang, Chang Zhou, Jin-  
659 gren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023.  
660 [Qwen technical report](#). *Preprint*, arXiv:2309.16609.

661 Boxi Cao, Keming Lu, Xinyu Lu, Jiawei Chen, Mengjie  
662 Ren, Hao Xiang, Peilin Liu, Yaojie Lu, Ben He, Xian-  
663 pei Han, Le Sun, Hongyu Lin, and Bowen Yu. 2024a.  
664 [Towards scalable automated alignment of llms: A  
665 survey](#). *Preprint*, arXiv:2406.01252.

666 Boxi Cao, Keming Lu, Xinyu Lu, Jiawei Chen, Mengjie  
667 Ren, Hao Xiang, Peilin Liu, Yaojie Lu, Ben He,  
668 Xianpei Han, et al. 2024b. [Towards scalable auto-  
669 mated alignment of llms: A survey](#). *arXiv preprint  
670 arXiv:2406.01252*.

671 Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu,  
672 Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi,  
673 Cunxiang Wang, Yidong Wang, et al. 2024. [A sur-  
674 vey on evaluation of large language models](#). *ACM  
675 Transactions on Intelligent Systems and Technology*,  
676 15(3):1–45.

677 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming  
678 Yuan, Henrique Ponde de Oliveira Pinto, Jared Ka-  
679 plan, Harri Edwards, Yuri Burda, Nicholas Joseph,  
680 Greg Brockman, Alex Ray, Raul Puri, Gretchen  
681 Krueger, Michael Petrov, Heidy Khlaaf, Girish Sas-  
682 try, Pamela Mishkin, Brooke Chan, Scott Gray,  
683 Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz  
684 Kaiser, Mohammad Bavarian, Clemens Winter,  
685 Philippe Tillet, Felipe Petroski Such, Dave Cum-  
686 mings, Matthias Plappert, Fotios Chantzis, Eliza-  
687 beth Barnes, Ariel Herbert-Voss, William Hebgren  
688 Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie  
689 Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain,  
690 William Saunders, Christopher Hesse, Andrew N.  
691 Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan  
692 Morikawa, Alec Radford, Matthew Knight, Miles  
693 Brundage, Mira Murati, Katie Mayer, Peter Welinder,  
694 Bob McGrew, Dario Amodei, Sam McCandlish, Ilya  
695 Sutskever, and Wojciech Zaremba. 2021a. [Evaluat-  
696 ing large language models trained on code](#).

697 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming  
698 Yuan, Henrique Ponde de Oliveira Pinto, Jared Ka-  
699 plan, Harri Edwards, Yuri Burda, Nicholas Joseph,  
700 Greg Brockman, Alex Ray, Raul Puri, Gretchen  
701 Krueger, Michael Petrov, Heidy Khlaaf, Girish Sas-  
702 try, Pamela Mishkin, Brooke Chan, Scott Gray,  
703 Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz  
704 Kaiser, Mohammad Bavarian, Clemens Winter,  
705 Philippe Tillet, Felipe Petroski Such, Dave Cum-  
706 mings, Matthias Plappert, Fotios Chantzis, Eliza-  
707 beth Barnes, Ariel Herbert-Voss, William Hebgren  
708 Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie  
709 Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain,  
710 William Saunders, Christopher Hesse, Andrew N.  
711 Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan  
712 Morikawa, Alec Radford, Matthew Knight, Miles  
713 Brundage, Mira Murati, Katie Mayer, Peter Welinder,  
714 Bob McGrew, Dario Amodei, Sam McCandlish, Ilya  
715 Sutskever, and Wojciech Zaremba. 2021b. [Evaluat-  
716 ing large language models trained on code](#). *Preprint*,  
717 arXiv:2107.03374.

718	Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. <a href="#">Teaching large language models to self-debug</a> . <i>Preprint</i> , arXiv:2304.05128.	772
719		773
720		774
721	Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024. <a href="#">Self-play fine-tuning converts weak language models to strong language models</a> . <i>Preprint</i> , arXiv:2401.01335.	775
722		776
723		777
724		
725	Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. 2023. <a href="#">Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality</a> .	778
726		779
727		
728		
729		
730		
731	Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. <a href="#">Training verifiers to solve math word problems</a> . <i>Preprint</i> , arXiv:2110.14168.	780
732		781
733		782
734		783
735		784
736		785
737	Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. <a href="#">Ultrafeedback: Boosting language models with high-quality feedback</a> . <i>arXiv preprint arXiv:2310.01377</i> .	786
738		787
739		788
740		
741		
742	Tri Dao. 2023. <a href="#">Flashattention-2: Faster attention with better parallelism and work partitioning</a> . <i>CoRR</i> .	789
743		
744	Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. <a href="#">Measuring massive multitask language understanding</a> . <i>Preprint</i> , arXiv:2009.03300.	790
745		791
746		792
747		793
748	Yuzhen Huang, Yuzhuo Bai, Zhihao Zhu, Junlei Zhang, Jinghan Zhang, Tangjun Su, Junteng Liu, Chuancheng Lv, Yikai Zhang, Jiayi Lei, Yao Fu, Maosong Sun, and Junxian He. 2023. <a href="#">C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models</a> . <i>Preprint</i> , arXiv:2305.08322.	794
749		795
750		796
751		797
752		798
753		799
754	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. <a href="#">Mistral 7b</a> . <i>arXiv preprint arXiv:2310.06825</i> .	800
755		801
756		802
757		803
758		804
759	Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024a. <a href="#">Mixtral of experts</a> . <i>arXiv preprint arXiv:2401.04088</i> .	805
760		806
761		807
762		808
763		809
764	Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024b. <a href="#">Follow-bench: A multi-level fine-grained constraints following benchmark for large language models</a> . <i>Preprint</i> , arXiv:2310.20410.	810
765		811
766		812
767		813
768		814
769		815
770	Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C. H. Hoi. 2022. <a href="#">Coder1: Mastering code generation through pretrained models and deep reinforcement learning</a> . <i>Preprint</i> , arXiv:2207.01780.	816
771		817
		818
		819
		820
		821
		822
		823
		824
		825
		826
		827
		828
		829
		830

831	Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeesh Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O’Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pocrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nicolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. <a href="#">Gpt-4 technical report</a> . <i>Preprint</i> , arXiv:2303.08774.	
878	Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.	
886	Baolin Peng, Chunyuan Li, Pengcheng He, Michel Galley, and Jianfeng Gao. 2023. Instruction tuning with gpt-4. <i>arXiv preprint arXiv:2304.03277</i> .	
889	Shuofei Qiao, Honghao Gui, Chengfei Lv, Qianghuai Jia, Huajun Chen, and Ningyu Zhang. 2024. <a href="#">Making language models better tool learners with execution feedback</a> . <i>Preprint</i> , arXiv:2305.13068.	
	Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. <a href="#">Infobench: Evaluating instruction following ability in large language models</a> . <i>Preprint</i> , arXiv:2401.03601.	893 894 895 896 897
	Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model.	898 899 900 901
	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. <i>Advances in Neural Information Processing Systems</i> , 36.	902 903 904 905 906
	Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. KDD ’20.	907 908 909 910
	Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. 2024. <a href="#">Conifer: Improving complex constrained instruction-following ability of large language models</a> . <i>Preprint</i> , arXiv:2404.02823.	911 912 913 914 915
	Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.	916 917 918 919
	Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. <a href="#">Llama 2: Open foundation and fine-tuned chat models</a> . <i>Preprint</i> , arXiv:2307.09288.	920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942
	Xinru Wang, Hannah Kim, Sajjadur Rahman, Kushan Mitra, and Zhengjie Miao. 2024a. Human-llm collaborative annotation through effective verification of llm labels. In <i>Proceedings of the CHI Conference on Human Factors in Computing Systems</i> , pages 1–21.	943 944 945 946 947
	Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khoshabi, and Hannaneh	948 949

950	Hajishirzi. 2023. <a href="#">Self-instruct: Aligning language models with self-generated instructions</a> . <i>Preprint</i> , arXiv:2212.10560.	1004
951		1005
952		1006
953	Zifeng Wang, Chun-Liang Li, Vincent Perot, Long T. Le, Jin Miao, Zizhao Zhang, Chen-Yu Lee, and Tomas Pfister. 2024b. <a href="#">Codeclm: Aligning language models with tailored synthetic data</a> . <i>Preprint</i> , arXiv:2404.05875.	1007
954		1008
955		1009
956		1010
957		1011
958	Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. <i>arXiv preprint arXiv:2109.01652</i> .	1012
959		1013
960		1014
961		1015
962		
963	Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Hu, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, et al. 2024. Long-form factuality in large language models. <i>arXiv preprint arXiv:2403.18802</i> .	1016
964		1017
965		1018
966		1019
967		
968	Congying Xia, Chen Xing, Jiangshu Du, Xinyi Yang, Yihao Feng, Ran Xu, Wenpeng Yin, and Caiming Xiong. 2024. <a href="#">Fofu: A benchmark to evaluate llms' format-following capability</a> . <i>Preprint</i> , arXiv:2402.18667.	1020
969		1021
970		1022
971		1023
972		1024
973	Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. Unsupervised data augmentation for consistency training. <i>Advances in neural information processing systems</i> , 33:6256–6268.	1025
974		1026
975		1027
976		1028
977	Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. <a href="#">Wizardlm: Empowering large language models to follow complex instructions</a> . <i>Preprint</i> , arXiv:2304.12244.	
978		
979		
980		
981		
982	Jianhao Yan, Yun Luo, and Yue Zhang. 2024. <a href="#">Refutebench: Evaluating refuting instruction-following for large language models</a> . <i>Preprint</i> , arXiv:2402.13463.	
983		
984		
985		
986	Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E. Gonzalez, and Ion Stoica. 2023. <a href="#">Rethinking benchmark and contamination for language models with rephrased samples</a> . <i>Preprint</i> , arXiv:2311.04850.	
987		
988		
989		
990	Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024. <a href="#">Self-rewarding language models</a> . <i>Preprint</i> , arXiv:2401.10020.	
991		
992		
993		
994	Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. <a href="#">Scaling relationship on learning mathematical reasoning with large language models</a> . <i>Preprint</i> , arXiv:2308.01825.	
995		
996		
997		
998		
999	Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. 2023. <a href="#">Instruction tuning for large language models: A survey</a> . <i>arXiv preprint arXiv:2308.10792</i> .	
1000		
1001		
1002		
1003		
	Yingxiu Zhao, Bowen Yu, Binyuan Hui, Haiyang Yu, Fei Huang, Yongbin Li, and Nevin L. Zhang. 2023. <a href="#">A preliminary study of the intrinsic relationship between complexity and alignment</a> .	
	Yingxiu Zhao, Bowen Yu, Binyuan Hui, Haiyang Yu, Minghao Li, Fei Huang, Nevin L Zhang, and Yongbin Li. 2024. Tree-instruct: A preliminary study of the intrinsic relationship between complexity and alignment. In <i>Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)</i> , pages 16776–16789.	
	Chenyu Zheng, Guoqiang Wu, and Chongxuan Li. 2024. Toward understanding generative data augmentation. <i>Advances in Neural Information Processing Systems</i> , 36.	
	Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. <i>Advances in Neural Information Processing Systems</i> , 36.	
	Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. <a href="#">Instruction-following evaluation for large language models</a> . <i>Preprint</i> , arXiv:2311.07911.	

## Appendix

### A Seed Instructions

Fig. 6 illustrates our hand-written seed instructions.

### B Implementation Details

To better motivate researchers to reproduce the results, I report the detailed experimental details:

In the SFT phase, we perform full fine-tuning on Qwen2-7B and Llama3-8B with a learning rate of  $7e-6$ , using a linear scheduler with 20 warm-up steps. All models are trained with DeepSpeed ZeRO Stage 3 (Rasley et al., 2020) and Flash-Attention 2 (Dao, 2023). We use a global batch size of 128, a weight decay of 0.1, and train for 3 epochs, saving checkpoints every 200 steps. Mixed precision training with bf16 is used, and the maximum context length is set to 8192 tokens. For Qwen2-72B and Llama3-70B, the global batch size is 512.

In the DPO phase, the learning rate is set to  $5e-7$  with a cosine scheduler and a 0.1 warm-up ratio. We use DeepSpeed ZeRO Stage 3 and Flash-Attention 2 for efficiency, with a global batch size of 64. Training utilizes a sigmoid loss function with a beta value of 0.3 and spans 2 epochs, with checkpoints every 200 steps. Mixed precision training with bf16 is employed, and the maximum context length is 4096 tokens.

We run all our experiments on NVIDIA A100 and H800 GPUs. Specifically, we train Qwen2-7 and LLaMa-3-8B on 8 A100 GPUs, while Qwen2-72B-Instruct and LLaMa-3-70B-Instruct on 64 H800 GPUs. Notably, we use an in-house version of Qwen2-7B without any targeted optimizations on instruction-following capabilities. For evaluations, we report pass@1 results with greedy decoding for HumanEval and zero-shot accuracy for GSM8K. We report averaged performance from five randomly seeded experiments.

### C Details of AUTOIF

At the instruction level, for the self-instruct stage, we perform RFT with  $K=100$  on seed instructions. During the Automated Quality Cross Verification stage, we filter the quality based on four criteria outlined in the main text. For NLI filtering, we use mDeberta as our filtering model<sup>2</sup>, and filter out

<sup>2</sup>The NLI model is available at <https://huggingface.co/MoritzLaurer/mDeBERTa-v3-base-xnli-multilingual-nli-2mil7>

only samples predicted as "Contradiction" (approximately 15%).

At the query level, we randomly select 16 ShareGPT samples for each instruction and perform Response Rejection Sampling with  $K=8$ . For instruction following verification, we adhere to the two standards mentioned in the text. Finally, for query quality verification, we filter for consistency using a threshold of 8.

### D Case Study of Data Combination

We used n-gram 13 to evaluate the overlap between each test sample and the SFT training samples. It is unnecessary to evaluate DPO data since the inputs for DPO data are derived from SFT data. In Tab. 4, all our data combination metrics (both model-based and rule-based evaluation) are lower than those of ShareGPT, confirming that our method has no data combination with the test set. We also present the top 5 training-test sample overlaps in n-gram for both IF Eval and Followbench in Fig. 7.

### E Case Study of AUTOIF

In Tab. 6, we illustrate the data format of our AUTOIF, including the query, response (verification funcns  $Acc>0.8$ ) and verification function.

### F Baselines & Datasets

We give introductions to the LLM baselines for our instruction following.

**Llama3** (Meta, 2024), developed by MetaAI, is the latest iteration of the Llama series, featuring significant upgrades. Compared to Llama 2, Llama 3 expands its training dataset, context length, and vocabulary, resulting in improved performance across various tasks. Enhancements in contextual understanding and language generation further distinguish Llama 3.

**Qwen2** (Bai et al., 2023), developed by Alibaba, includes five sizes: Qwen2-0.5B, Qwen2-1.5B, Qwen2-7B, Qwen2-57B-A14B, and Qwen2-72B. Trained on high-quality data in Chinese, English, and 27 other languages, Qwen2 excels in multilingual capabilities and shows strong performance in coding and mathematics. Additionally, it supports extended context lengths of up to 128K tokens (Qwen2-72B-Instruct), making it ideal for long texts and complex tasks.

## Seed Instructions

1. Answer with words that begin with the letter 'B'	13. Use only military lingo	25. Use only the first half of the alphabet (A-M)
2. Construct the reply as if it's a telegram STOP	14. Respond with a haiku (5-7-5 syllable structure)	26. Use only questions to form your reply
3. Use only palindromes	15. Write the response in future tense only	27. Use only words that start and end with the same letter
4. Use words that end with '-ing'	16. Use only monosyllabic words	28. Write the response in Morse code
5. Write the response backward	17. Answer with words in alphabetical order	29. Use only words that are colors
6. Use only words with double letters (e.g., "bookkeeper")	18. Write the response as a limerick	30. Use only the second half of the alphabet (N-Z)
7. Use only onomatopoeia	19. Use no adjectives or adverbs	31. Answer with each sentence decreasing in word count
8. Answer with a single sentence that is exactly 100 words long	20. Respond with a six-word story	32. Respond with a list of bullet points
9. Use no words containing the letter 'E'	21. Include at least three rhyming pairs	33. Answer with a sequence of puns
10. Translate your answer into emojis	22. Write the response in iambic pentameter	34. Answer with emoji only
11. Use only the 1000 most common English words	23. Use alliteration throughout your answer	35. Use only words that have an X in them
12. Incorporate a famous movie quote seamlessly into your answer	24. Answer in the form of a sonnet (14 lines with 10 syllables each)	36. Answer with each word starting with the next letter of the alphabet

Figure 6: Examples of our seed instructions

### Case study

On IFEVAL			On Follow Bench		
N-gram	Train data	Test data	N-gram	Train data	Test data
8.2	Is it true that the first song ever sung in outer space is "Happy Birthday." Your answer must contain one of the following phrases: My answer is yes. My answer is no. My answer is maybe.	Is it true that AI is dangerous for humankind? Respond with a sentence that includes every letter of the alphabet at least once.	8.0	You are a doctor. Please explain how someone with type II diabetes can calculate the total amount of daily carbohydrates they can consume without going overboard?	You are a Russian physics professor. Create a ridiculous problem set in the course Quantum Mechanics 1. Write the response as a series of conditional statements.
8.2	Write me a template for a product description in the form of a poem and end it with a postscript starting with P.P.S.	Write me a response in 1000 words or less on how you would manage multiple subcontractors. Use only words that are the name of a body part.	7.3	How did US states get their names? Please respond in the writing style of Shakespeare.	How do I properly offboard users in Microsoft 365 with PowerShell? Answer with each sentence being a statement.
8.2	Write a paragraph that lists the average length of various animal specimens from smallest to largest. Your response should contain less than 17 sentences.	Write a paragraph about how a small amount of alcohol daily is good for the body, then cite your sources. Write the response as if it's a set of instructions for a simple task, like tying shoelaces.	6.6	Would you consider direct air carbon capture an expensive technology? Please provide one reason to support your opinion.	Would you write me a Unity code for a simple Flappy Bird-like game? Answer with words that have a homophone.
8.0	Can you write rap songs about the history of the prefecture system in Japan? Give exactly two different responses separated by 6 asterisk symbols *****.	Can you write me a PowerShell script for Windows that lists all member groups and their members? Write the response as a series of book titles.	5.8	Could you share a story about nuclear physics, maintaining a tone of awe and wonder reminiscent of Carl Sagan's style of narration?	Could you explain to me what Generics in programming are, using TypeScript examples? Use alliteration and consonance throughout your answer.
8.0	What is a lattice? Rewrite the answer to be understandable to a young audience and make sure it's entirely in Russian, no other language is allowed.	What is a good product to start selling on TikTok? It needs to be able to generate catchy videos on TikTok. Answer with words that are all the same length.	5.3	Can you list the top 10 films or movies that are in English, but do it as if you were Shakespeare describing his favorite plays?	Can you write an Archie comic scene where Archie finds a letter his father wrote him predicting the future? Translate your answer into ASCII art

Figure 7: Case Study of data combination on IFEval and Followbench

**Mistral-7B** (Jiang et al., 2023), released by Mistral AI in September 2023, leverages grouped query attention (GQA) combined with sliding window attention (SWA) to efficiently process sequences of any length, enhance inference speed, and improve throughput. It outperforms many 13B models across various tasks.

**Mixtral-8x7B** (Jiang et al., 2024a) developed by Mistral AI, is the first open-source MOE large model. It is a sparse mixture of experts network and, like Mistral 7B, employs the GQA mechanism. With a smaller parameter count compared to Llama2 70B and GPT-3.5, it outperforms them across numerous tasks.

**GPT Series** GPT-3.5 (OpenAI, 2022) and GPT-4 (Achiam et al., 2023), developed by OpenAI, are advanced models in the GPT series that use a three-stage reinforcement learning with human feedback (RLHF) algorithm. This enhances their

instruction-following capabilities and minimizes harmful content generation. GPT-3.5 excels in text completion, translation, and summarization. Building on these strengths, GPT-4 further refines the RLHF algorithm, enhancing performance on complex instructions and making it suitable for applications ranging from academic research to industrial use.

In addition to the two Instruction-Following benchmarks introduced in the main text, we also provide a detailed overview of datasets covered in the experiments

**ShareGPT** refers to the multi-turn chatting histories used by Vicuna (Chiang et al., 2023). ShareGPT includes 86K human queries and responses from ChatGPT and other chatbots. We randomly select 2w samples to train Llama3-8B and Qwen2-7B to obtain our baseline models: Llama3-8B(ShareGPT) and Qwen2-7B(ShareGPT).<sup>3</sup>

<sup>3</sup>We use the version from <https://huggingface>.

1157 **GSM8K** (Cobbe et al., 2021) is a mathemati-  
1158 cal dataset designed to evaluate the mathematical  
1159 problem-solving abilities of language models. It  
1160 consists of 8,000 diverse grade school-level math  
1161 word problems, which require understanding and  
1162 manipulating mathematical concepts to arrive at a  
1163 correct solution. It comprises high-quality grade  
1164 school math problems, with 7,473 training samples  
1165 and 1,319 testing samples.

1166 **HumanEval** (Chen et al., 2021b) includes 164  
1167 unique programming challenges, each paired with  
1168 approximately 9.6 test cases on average. To provide  
1169 a more comprehensive evaluation of the functional  
1170 accuracy of code generated by large language mod-  
1171 els, HumanEval+ substantially increases the num-  
1172 ber of test cases to an average of 774.8 per problem.  
1173 In this paper, we report the Pass@1 result when  
1174 applying greedy decoding.

1175 **MMLU** (Hendrycks et al., 2021) is a benchmark  
1176 designed to assess pretraining knowledge in mod-  
1177 els using zero-shot and few-shot evaluations. It in-  
1178 cludes 57 subjects across STEM, humanities, social  
1179 sciences, and more, with difficulty levels ranging  
1180 from elementary to advanced professional. MMLU  
1181 tests both world knowledge and problem-solving  
1182 skills, covering traditional disciplines like math-  
1183 ematics and history, as well as specialized areas  
1184 such as law and ethics.

1185 **C-Eval** (Huang et al., 2023) consists of multiple-  
1186 choice questions categorized into four difficulty  
1187 levels: middle school, high school, college, and  
1188 professional. The questions cover 52 varied disci-  
1189 plines, including humanities, science, and engineer-  
1190 ing. Additionally, there is C-Eval Hard, a subset of  
1191 particularly challenging topics within C-Eval that  
1192 demand advanced reasoning skills. We perform an  
1193 in-depth evaluation of leading language models on  
1194 C-Eval, testing both English and Chinese-focused  
1195 models.

---

co/datasets/anon8231489123/ShareGPT\_Vicuna\_  
unfiltered cleaned raw dataset, and follow Vicuna  
preprocess.

Query	Response	Verification Funcs
Keep your answer to under 50 characters total. what is autoarima in python.	AutoARIMA automates ARIMA model selection.	<pre>def evaluate(response: str) -&gt; bool:     return len(response) &lt;=         50</pre>
Refrain from using any words that contain 'S'. i need a name for my Dungeons and Dragons world.	EternaRealm	<pre>def evaluate(response):     forbidden = 'sS'     for char in response:         if char in             forbidden:                 return False     return True</pre>
Keep your response under twenty words without sacrificing clarity. Let's play a game shall we?.	ure, let's play a game! What game do you have in mind? Please keep instructions simple and clear.	<pre>def evaluate(response: str) -&gt; bool:     return len(response.         split()) &lt;= 20 and         len(response) &gt; 0</pre>
Compose your answer using exactly 20 words. Different Normal Forms.	Normal forms in databases: 1NF ensures atomic columns, 2NF eliminates non-key dependencies, 3NF removes transitive dependencies, BCNF enforces determinant restriction.	<pre>def evaluate(response: str) -&gt; bool:     return len(response.         split()) == 20</pre>
Word your response in a five-line verse with a strict AABBA rhyme. 1. Write short notes on Decision trees..	Decision trees, so clear and bright, Branch out to split data's might, With nodes of questions, true or false, They sort through cases, young or old, like a versatile horse. From root to leaves, paths decide their course.	<pre>def evaluate(response: str) -&gt; bool:     lines = response.split(         '\n')     if len(lines) != 5:         return False     rhymes = [line[-1] for         line in lines]     rhyme_pattern = rhymes         [0] == rhymes[1] ==             rhymes[2] !=                 rhymes[3] == rhymes                     [4]     return rhyme_pattern</pre>

Table 6: Examples of AutoIF's data formats.