Ada-Instruct: Adapting Instruction Generators for Complex Reasoning

Wanyun Cui and **Qianle Wang**
Shanghai University of Finance and Economics
cui.wanyun@sufe.edu.cn, wql20000111@stu.sufe.edu.cn

Abstract

Instructions augmentation is a crucial step for unleashing the full potential of large language models (LLMs) in downstream tasks. Existing Self-Instruct methods primarily simulate new instructions from a few initial instructions with in-context learning. However, our study identifies a critical flaw in this approach: even with GPT4o, Self-Instruct cannot generate complex instructions of length ≥ 100, which is necessary in complex tasks such as code completion.

To address this issue, our key insight is that fine-tuning open source LLMs with only *ten examples* can produce complex instructions that maintain distributional consistency for complex reasoning tasks. We introduce Ada-Instruct, an adaptive instruction generator developed through fine-tuning. We empirically validated Ada-Instruct's efficacy across different applications. The results highlight Ada-Instruct's capacity to generate long, intricate, and distributionally consistent instructions.[1]

1 Introduction

Supervised fine-tuning (SFT) is crucial for harnessing the potential of pre-trained Large Language Models (LLMs) in downstream tasks. Addressing SFT's demand for extensive training data, recent research has employed advanced LLMs, such as ChatGPT, to generate instructions. A prevalent approach is called "Self-Instruct" (Wang et al., 2022), which involves having ChatGPT sequentially generate both instructions and answers (Sun et al., 2023; Peng et al., 2023; Taori et al., 2023; Schick and Schütze, 2021; Honovich et al., 2022; Ye et al., 2022; Meng et al., 2022, 2023). It efficiently generates substantial novel training samples from a minimal number of initial samples.

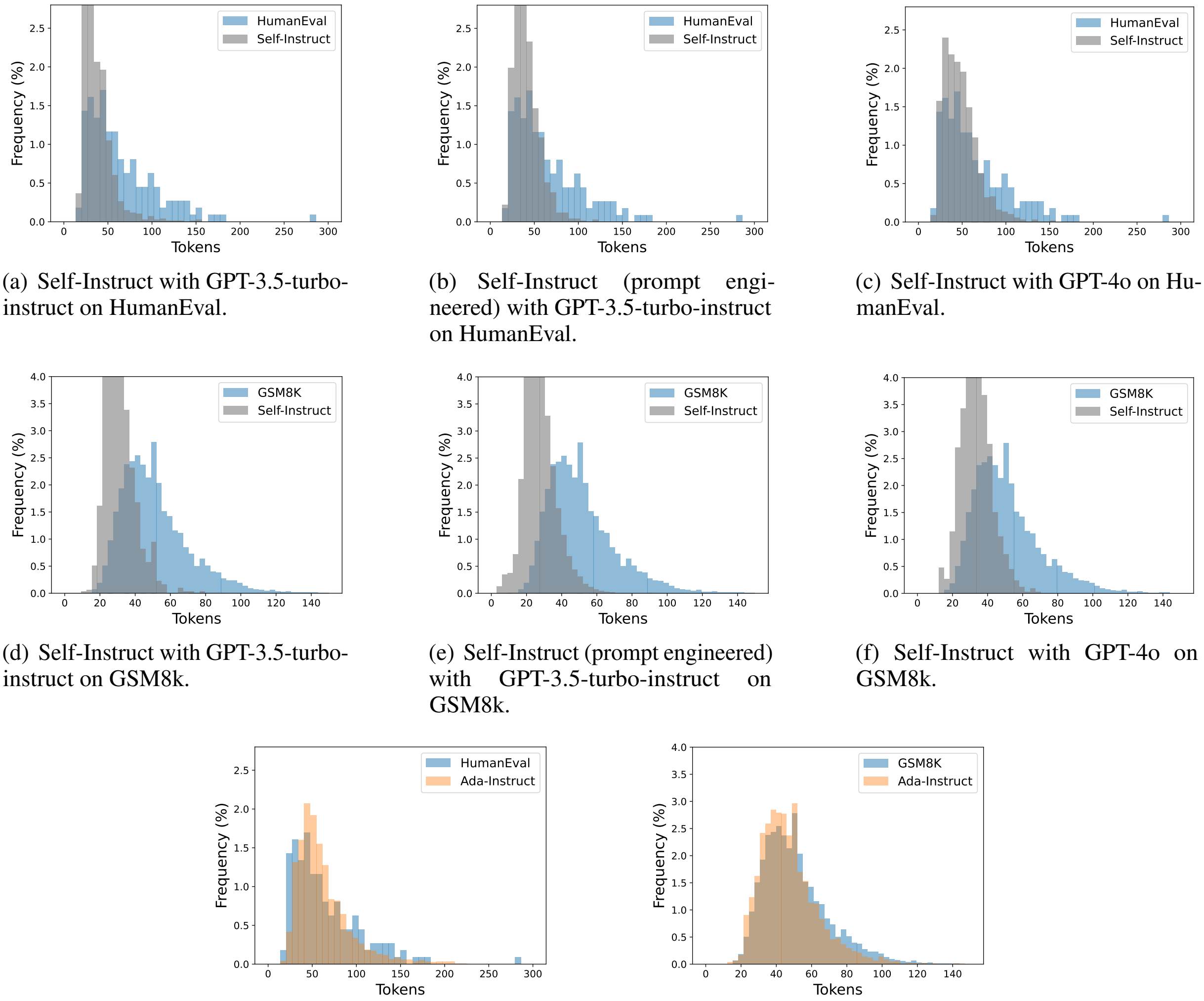
However, our observations reveal a fundamental and critical limitation of Self-Instruct — *it notably struggles to generate complex instructions.* Despite being demonstrated with long and complex examples, Self-Instruct predominantly produces disappointingly brief and overly simplistic instructions. This is evident in Figure 1(a) and Figure 1(d), where we present the length distribution of instructions by Self-Instruct on HumanEval (programming) and GSM8k (mathematics). The figures expose a glaring gap: Self-Instruct fails to produce instructions that exceed 100 and 60 tokens for HumanEval and GSM8k, respectively. This limitation significantly undermines the use of self-instruct in more complex tasks.

Is Prompt Engineering a Solution? Despite its widespread use in enhancing in-context learning, prompt engineering is not the panacea it is often made out to be (Wang et al., 2022; Sun et al., 2023; Zhou et al., 2022; Yang et al., 2023). To encourage the generation of longer and more complex instructions, we explored infusing prompts with extra requirements, such as "generate algorithms of intermediate level" (for HumanEval) and "the instructions should not be too easy" (for GSM8k). However, as shown in Figure 1(b)1(e), this approach did not effectively solve the problem of producing short instructions. A more advanced variant of prompt engineering, Evol-Instruct, employs multiturn strategies to incrementally enhance the complexity and variety of instructions. However, we will show in § 4.4.1 that Evol-Instruct is unable to generate instructions that semantically align with the target instruction distribution.

Has the Problem Been Solved by the More Advanced GPT4o? We performed additional evaluations using the advanced GPT4o model, which is equipped with superior reasoning and long-text processing capabilities. Figures 1(c)1(f) illustrate that while GPT4o outperforms gpt-3.5-turbo-instruct in terms of average output length on the HumanEval benchmark, it still falls short of generating instructions longer than 100 tokens. Similarly, on the GSM8k benchmark, GPT4o shows no marked im-

[1] Code is available at https://github.com/wangitu/Ada-Instruct

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 6967–6984
November 12-16, 2024 ©2024 Association for Computational Linguistics



(a) Self-Instruct with GPT-3.5-turbo-instruct on HumanEval.

(b) Self-Instruct (prompt engineered) with GPT-3.5-turbo-instruct on HumanEval.

(c) Self-Instruct with GPT-4o on HumanEval.

(d) Self-Instruct with GPT-3.5-turbo-instruct on GSM8k.

(e) Self-Instruct (prompt engineered) with GPT-3.5-turbo-instruct on GSM8k.

(f) Self-Instruct with GPT-4o on GSM8k.

(g) Ada-Instruct on HumanEval.

(h) Ada-Instruct on GSM8k.

Figure 1: Length Distribution of Different Methods. The length is measured by the number of tokens. All methods start with the same 10 instructions. (a)(d): Self-Instruct struggles to generate complex instructions with more tokens, even being explicitly asked to do so (b)(e). (c)(f): The more advanced GPT-4o still has this issue. (g)(h): Ada-Instruct successfully produces instructions whose length is consistently aligned with the target distribution.

provement in its capacity to produce longer instructions. Consequently, the challenge of generating complex instructions remains with the more advanced GPT4o.

In this paper, we unveil a novel insight into the instruction generation capabilities. Surprisingly, we find that *even when relying solely on 10 samples, a straightforward fine-tuned model is capable of generating instructions that align with the target task distribution.* In Figure 1(g), FT models generate instructions of length $\geq$ 100 tokens for HumanEval, and in Figure 1(h), of length $\geq$ 60 tokens for GSM8k, both matching the actual distribution. In addition, the generated instructions span the target distribution (§ 4.4.1), and exhibit high diversity (§ 4.4.2).

Based on these findings, we introduce Ada-Instruct, a few-shot instruction generation procedure for downstream tasks. We fine-tune open-source LLMs using few-shot task samples for instruction generation, instead of ICL as in Self-Instruct.

In summary, our contributions include (1) We uncover a new insight into the sample generation capabilities of self-instruct, showing that it cannot generate complex instructions. (2) We introduce Ada-Instruct, a few-shot instruction generation methodology with fine-tuning. (3) We verify the effectiveness of Ada-Instruct through empirical validations, showcasing its superiority in generating complex instructions that are not only longer, but also aligned with the target distributions.

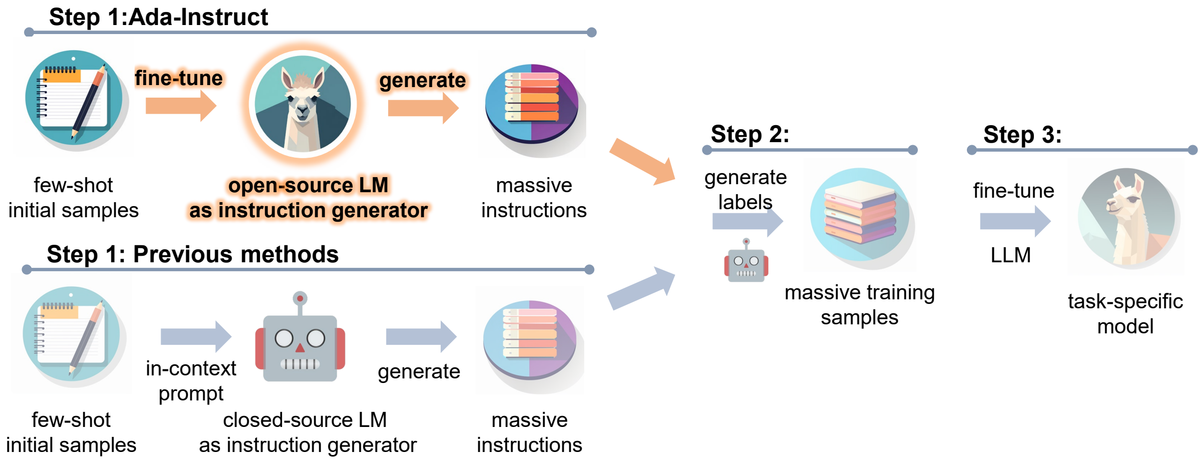


Figure 2: How Ada-Instruct works. We fine-tune LLMs as instruction generators from few-shot initial samples (step 1), while previous self-instruct methods use in-context prompting and closed-source LLMs. We then use ChatGPT to generate labels (step 2), and fine-tune a task-specific model with the labeled samples (step 3).

2 Related Work

Sample Generation via LLMs Recent works have explored the use of LLMs for sample generation, often within the self-instruction framework (Chen et al., 2023). This typically involves starting from an initial pool of instructions and having the LLMs iteratively generate new instructions along with the corresponding answers. Most prior work in the realm of instruction generation has relied on ICL (Wang et al., 2022; Taori et al., 2023; Sun et al., 2023; Xu et al., 2023; Honovich et al., 2022; Meng et al., 2022). Various studies have focused mainly on improving the self-instruct approach in different problem scenarios.

However, a limitation of this paradigm, as we have observed, is that ICL lacks the capacity to generate complex samples based solely on in-context examples. Although more intricate samples could potentially be produced using evolutionary strategies, such as Evol-Instruct (Xu et al., 2023; Luo et al., 2023a,b), these manually designed tactics risk generating samples that do not align with the target task distribution.

FewGen (Meng et al., 2023) is the only method we have identified that substitutes fine-tuning for In-Context Learning (ICL) in sample generation. However, FewGen requires sophisticated metalearning and is limited to classification tasks. In contrast, Ada-Instruct is substantially simpler and more general.

ICL vs. FT Previous exploratory studies have aimed to compare the performance of ICL and FT methodologies. Some research suggests that ICL exhibits a more robust out-of-distribution generalization compared to FT (Si et al., 2022; Awadalla et al., 2022; Utama et al., 2021). However, some recent studies (Mosbach et al., 2023) argue that these earlier comparisons may be biased. The unfairness arises from using different model architectures for comparison (e.g., GPT-3-based ICL versus RoBERTa (Liu et al., 2019)-based FT) or by basing results on small-scale models. In more equitable experimental setups, the researchers found that FT outperforms ICL (Mosbach et al., 2023), thereby supporting our strategy of using FT models for instruction generation.

3 Method

Ada-Instruct is divided into three steps: 1) Learning an instruction generator and generating massive instructions (§ 3.1), 2) generating labels with ChatGPT (§ 3.2), and 3) training LLMs for downstream tasks (§ 3.3). In the following, we dive into the details of each step. The overall workflow is shown in Figure 2.

3.1 Learning to Generate Instructions (Step 1)

The first step focuses on learning an instruction generator using a small set of samples. In most real-world scenarios, obtaining large labeled datasets for every new downstream task is infeasible. Hence, an instruction generator serves as an intermediary, converting small sets of samples into sufficient instructions for data labeling or task understanding.

Given a target downstream task T and a small set of samples $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the objective is to fine-tune an initial LLM $M(\theta)$

with parameters θ to produce instructions I that have the same distribution as the instruction X of task T and are beneficial for fine-tuning.

The goal of fine-tuning is learning to generate instructions X. Thus its objective is to optimize the parameters θ of the LLM to maximize the conditional likelihood of the target sequences given their corresponding instructions::

$$\mathcal{L}_{\text{inst}}(\theta) = -\frac{1}{n} \sum_{(x_i, y_i) \in S} \log P_M(x_i|\theta) \quad (1)$$

Here, $P_M(x_i|\theta)$ denotes the probability of observing the target instruction x_i under the current model parameters θ. θ is initialized as the pre-trained parameters. In causal language modeling, the probability of the target instruction is represented as the product of the conditional probabilities of the individual tokens in it.

Generating Massive Instructions: After fine-tuning, the instruction generator is used to generate a large volume of instructions. The templates in this step are provided in Appendix G.1. These instructions serve as the basis for the subsequent phases for generating high-quality samples.

Filtering Duplicate Instructions: As massive instructions are generated from the LLM trained by a few samples, one issue is whether these instructions are duplicated. We assume that if two instructions are highly similar, using the two instructions to fine-tune the final LLM will be less effective. To further ensure the uniqueness of generated instructions, a simple filtering mechanism is used. This mechanism uses a pre-trained sentence embedding model to calculate the semantic similarity between generated instructions. If the semantic similarity between two instructions is above a predetermined threshold, the latter instruction is filtered out to avoid redundancy. In this paper, we use MPNet (Song et al., 2020) to compute the semantic similarities.

3.2 Label Generation (Step 2)

In the second step, we leverage a high quality closed-source LLM, ChatGPT [2], to generate labels for the instructions produced in step 1. Using ChatGPT alleviates the need for extensive manual labeling, providing a cost-efficient and time-effective way to accumulate labeled data based on the instructions generated in step 1 (Gilardi et al., 2023).

Given the set of instructions $I = \{x_1, x_2, \dots, x_m\}$, the objective here is to generate their corresponding labels $y_1, y_2, \dots, y_m$. For each instruction I in the set, ChatGPT generates a corresponding response, transforming I into a new training set $\mathbb{S} = \{(x_1, y_1), \dots, (x_m, y_m)\}$.

3.3 Training LLMs for Downstream Tasks (Step 3)

The final step utilizes the complete training samples S' obtained from Step 2 to train LLMs for the target downstream tasks.

The objective function is also a casual language modeling loss over the given samples, adjusted to fit the labels of the new set of samples $\mathbb{S}$ from Step 2. A new LLM $\mathbb{M}(\theta)$ is used for fine-tuning with the pre-trained parameter initialization:

$$\mathcal{L}_{\text{task}}(\theta) = -\frac{1}{m} \sum_{(x_i, y_i) \in \mathbb{S}} \log P_{\mathbb{M}}(y_i|x_i;\theta) \quad (2)$$

4 Experiments

In our experiments, we evaluate the effectiveness of Ada-Instruct in code completion (§ 4.1), mathematics (§ 4.2), and commonsense reasoning (§ 4.3). We further analyze its distributional consistency with the target task, assessing (1) **Semantic Consistency** (§ 4.4.1): the alignment of generated examples with the target distribution, and (2) **Diversity** (§ 4.4.2): the variety in instructions from 10 initial samples. We also address the concern regarding whether fine-tuning an open-source model could result in diminished performance, considering that open-source models are often perceived as less qualified compared to closed-source models (§ 4.5). All experiments ran on a single node with 8 x A100 80GiB GPUs.

4.1 Code Completion

Setup: We utilize two widely recognized benchmarks: HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021). For both benchmarks, our experiments began with an initial set of 10 samples. Specifically for MBPP, these samples were randomly extracted from its development set. For HumanEval, which does not have a development set, we selected 10 representative problems from LeetCode and the MBPP development set. This selection was aimed at closely mirroring the difficulty level as in HumanEval. These chosen samples were then appropriately formatted to align with HumanEval's query structure. We developed

[2] We use `gpt-3.5-turbo-instruct` in this paper

Model	**Initial Data**	**SFT Data**	**Size**	**HumanEval**	**MBPP**
Base model	-	-	13B	43.3	49.0
SOTA baselines					
PaLM	-	-	540B	26.2	36.8
PaLM-Coder	-	-	540B	36.0	47.0
PaLM 2-S	-	-	-	37.6	50.0
StarCoder$_{\text{Python}}$	-	-	15.5B	33.6	52.7
StarCoder$_{\text{Prompted}}$	-	-	15.5B	40.8	49.5
Code-Cushman$_{001}$	-	-	12B	33.5	45.9
GPT-3.5	-	-	-	48.1	52.2
GPT-4	-	-	-	**67.0**	-
Self-Instruct baselines					
Self-Instruct$_{\text{HE}}$	10	10k	13B	47.0 (+8.5%)	-
Self-Instruct$_{\text{MBPP}}$	10	10k	13B	-	51.2 (+4.5%)
Evol-Instruct	20k	78k	13B	64.0(+47.8%)	55.6(+13.5%)
Ada-Instruct$_{\text{HE}}$	10	10k	13B	**65.2** (+50.6%)	-
Ada-Instruct$_{\text{MBPP}}$	10	10k	13B	-	**55.6** (+13.5%)

Table 1: Results of pass@1 (%) on HumanEval and MBPP, showcasing relative improvements over the base model. Results related to Code LLAMA are from (Rozière et al., 2023). Results of other baselines and from (Luo et al., 2023b). We follow (Rozière et al., 2023) to adopt a greedy decoding strategy in Ada-Instruct.

two models based on the instructions generated for HumanEval and MBPP, named Ada-Instruct$_{\text{HE}}$ and Ada-Instruct$_{\text{MBPP}}$, respectively. We use Code LLAMA-Python (13B) (Rozière et al., 2023) as our base model.

Baselines: The primary baseline is Self-Instruct. We ensure that it utilized an identical set of initial samples and the same quantity of SFT samples for a fair comparison. We denote two models built on the two generated instruction sets as Self-Instruct$_{\text{HE}}$ and Self-Instruct$_{\text{MBPP}}$, respectively. Another vital baseline was Evol-Instruct (WizardCoder (Luo et al., 2023b)), selected to evaluate the impact of sophisticated multi-turn prompt engineering techniques. We use the WizardCoder-Python-13B version, which also uses Code LLAMA-Python (13B) as the base model. Furthermore, our analysis included comparisons with leading-edge models in the field, such as PaLM (Chowdhery et al., 2022), PaLM-Coder (Chowdhery et al., 2022), PaLM 2-S (Anil et al., 2023), StarCoder (Li et al., 2023), and GPTs (OpenAI, 2023), to establish a comprehensive comparison with the current state-of-the-art.

Main Results: Effect of Ada-Instruct: We show the results in Table 1. Compared to state-of-the-art baselines, Ada-Instruct maintains a significant advantage in effectiveness. Its pass@1 rate is second only to GPT-4. Compared to the base model (Code LLAMA-Python), Ada-Instruct exhibits a notable improvement in performance. This enhancement is particularly significant on HumanEval, where the relative increase reaches 50.6%, even when initiated with as few as 10 samples. This substantial boost underscores the adaptability of Ada-Instruct, illustrating its ability to adapt LLMs to downstream tasks. The results lend evidence to Ada-Instruct's efficacy in optimizing language models for specific tasks.

Comparison with Self-Instruct We compared the performance of Ada-Instruct with Self-Instruct baselines. It is clear that with the same initial samples and the same amount of SFT data, Ada-Instruct significantly surpasses Self-Instruct in effectiveness. Ada-Instruct also shows superior performance compared to WizardCoder, which uses multi-turn prompting. Notably, WizardCoder requires 20k initial samples and 78k SFT data, which is considerably more than the sample size used by Ada-Instruct. These comparisons validate the superiority of Ada-Instruct over Self-Instruct in terms of effectiveness. We will further elaborate in Sec 4.4 that the instructions generated by Ada-Instruct exhibit greater semantic consistency, diversity, and coverage compared to those produced by Self-Instruct and Evol-Instruct.

Generalization Abilities for Multiple Tasks To validate its generalization ability, we also adapt Ada-Instruct to target a domain of multiple tasks rather than a single task. This is achieved by expanding the initial sample pool to include initial samples from different tasks. We conducted a direct experiment: We used an initial sample set comprising 10 initial HumanEval samples and 10 initial MBPP samples. Using these 20 initial samples, our Ada-Instruct framework generated 10k instructions in total. We then trained a domain model, termed Ada-Instruct$_{\text{Program}}$. For comparison, we also tested the performance of Self-Instruct using the same 20 initial samples and the same amount of SFT samples, denoted as Self-Instruct$_{\text{Program}}$. As shown in Table 2, it is evident that Ada-Instruct still achieves a significant performance improvement in the target domain with just 20 initial samples, surpassing the results of Self-Instruct.

Effect on Unseen Tasks We also assessed the generalization capability on unseen tasks within the code completion domain. Specifically, we tested two scenarios:

Model	Initial Data	SFT Data	HumanEval	MBPP
Base model	-	-	43.3	49.0
Self-Instruct$_{\text{Program}}$	20	10k	51.8$_{(+19.6\%)}$	47.8$_{(-2.5\%)}$
Ada-Instruct$_{\text{Program}}$	20	10k	**62.8**$_{(+45.0\%)}$	**54.0**$_{(+10.2\%)}$

Table 2: Results of pass@1 (%) on multiple code completion tasks.

Model	Training Data	Evaluation Task	Pass@1
Base model	-	HumanEval	43.3
Self-Instruct	10k HumanEval	HumanEval	47.0
Ada-Instruct	**10k MBPP**	**HumanEval**	**60.4**
Base model	-	MBPP	49.0
Self-Instruct	10k MBPP	MBPP	51.2
Ada-Instruct	**10k HumanEval**	**MBPP**	**52.4**

Table 3: Results of pass@1 (%) on unseen code completion tasks.

1. Utilize 10 initial HumanEval instructions and generate 10k SFT instructions. Then evaluate the fine-tuned model on MBPP.

2. Utilize 10 initial MBPP instructions and generate 10k SFT instructions. Then evaluate the fine-tuned model on HumanEval.

As presented in Table 3, Ada-Instruct demonstrates robust generalization abilities on unseen tasks, even outperforms self-instruct which was trained on the target task.

4.2 Math

Setup: We evaluated Ada-Instruct on two benchmarks: GSM8k (Cobbe et al., 2021) (easier) and MATH (Hendrycks et al., 2021) (harder). We randomly sampled 10 instructions from the training set of each benchmark as the initial samples. We require that the 10 MATH samples not be related to drawing scripts. We developed two models based on the instructions generated for each benchmark, named Ada-Instruct$_{\text{GSM8k}}$ and Ada-Instruct$_{\text{MATH}}$, respectively. The base model used here was LLAMA 2.

Baselines: We employed Self-Instruct as the baseline. The models developed using initial instructions from GSM8k and MATH are respectively denoted as Self-Instruct$_{\text{GSM8k}}$ and Self-Instruct$_{\text{MATH}}$. We have omitted the comparison with Evol-Instruct, as its implementation in WizardMath (Luo et al., 2023a) already incorporates GSM8k and MATH as part of their training datasets.

Model	Initial Data	SFT Data	Size	GSM8k	MATH
Base model	8	-	13B	28.7	3.9
SOTA Models					
Falcon	-	-	40B	19.6	2.5
Baichuan-chat	-	-	13B	23.9	-
Vicuna v1.3	-	-	13B	27.6	-
GPT3	-	-	175B	34.0	5.2
Text-davinci-002	-	-	175B	40.7	19.1
Chinchilla	-	-	70B	43.7	-
LLAMA 2	-	-	34B	42.2	6.2
LLAMA 2	-	-	70B	56.8	13.5
GPT-3.5	-	-	-	57.1	-
PaLM 2	-	-	540B	80.7	34.3
GPT-4	-	-	-	**92.0**	**42.5**
Self-Instruct Baselines					
Self-Instruct$_{\text{GSM8k}}$	10	10k	13B	30.8 (+7.3%)	-
Self-Instruct$_{\text{MATH}}$	10	10k	13B	-	5.8 (+48.7%)
Ada-Instruct$_{\text{GSM8k}}$	10	10k	13B	**48.7** (+69.7%)	-
Ada-Instruct$_{\text{MATH}}$	10	10k	13B	-	**8.8** (+125.6%)

Table 4: Results on GSM8k and MATH, demonstrating relative improvements over the base model (LLAMA 2). For the base model, we follow (Touvron et al., 2023) to deploy 8-shot in-context learning. Results of baselines are from (Luo et al., 2023a). The decoding strategy of Ada-Instruct was sourced from (Luo et al., 2023a).

Effect: In Table 4, we observed a significant performance enhancement of Ada-Instruct in comparison with the base model. Ada-Instruct demonstrated a relative improvement of 69.7% and 125.6% on GSM8k and MATH, respectively, compared to the base model (LLAMA 2-13B). This surpassed the performance of LLAMA 2-34B and achieved state-of-the-art results in few-shot instruction generation models.

Comparison with Self-Instruct: In Table 4, we also compare the performance of Ada-Instruct and Self-Instruct. The settings for both Self-Instruct and Ada-Instruct are kept consistent. Ada-Instruct markedly surpasses Self-Instruct.

4.3 Commonsense Reasoning

Setup: We evaluated the effectiveness of Ada-Instruct on CommonsenseQA (Talmor et al., 2019), a benchmark for commonsense reasoning. We randomly selected 10 samples from the training set to serve as initial samples. We choose LLAMA 2-13B as our base model.

Model	Initial Data	SFT Data	Size	Accuracy
Base Models				
LLAMA 2 (0-shot)	-	-	13B	59.0*
LLAMA 2 (1-shot)	1	-	13B	62.8*
LLAMA 2 (7-shot)	7	-	13B	67.3
LLAMA 2 (10-shot)	10	-	13B	68.1*
SOTA Models				
GPT-NeoX	-	-	20B	60.4
BLOOM	-	-	176B	64.2
OPT	-	-	66B	66.4
BloombergGPT	-	-	51B	65.5
ChatGPT	-	-	-	74.0
Self-Instruct Baselines				
Self-Instruct	10	10k	13B	71.4*$_{(+21.0\%)}$
Evol-Instruct	52k	250k	13B	64.0*$_{(+8.5\%)}$
Ada-Instruct	10	10k	13B	**75.5*** $_{(+28.0\%)}$

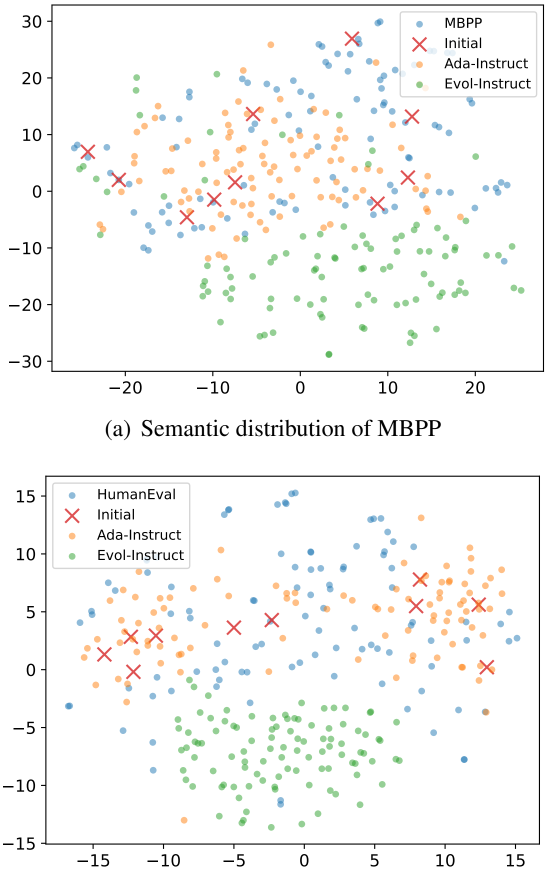
Table 5: Results on CommonsenseQA. Results related to LLAMA 2 are from (Touvron et al., 2023). Results of other baselines are from (Wu et al., 2023). *: results are tested on the dev set.

Baselines: We compare with Self-Instruct with the same initial samples and the same amount of SFT data. We also compare with Evol-Instruct with the implementation of WizardLM (Xu et al., 2023)). For a fair comparison, we used the WizardLM-13B-V1.2 version, which also employs LLAMA2-13B as its base model.

Results: Based on the results presented in Table 5, we observe a substantial improvement in performance attributed to Ada-Instruct. Ada-Instruct also demonstrated superior performance compared to both Self-Instruct and Evol-Instruct.

4.4 Analysis of Distributional Consistency

We have already illustrated in Figure 1 that Ada-Instruct is capable of generating instructions whose length distribution aligns with the target task. We will now proceed to further analyze their semantic consistency. Given that we only used 10 initial samples, our investigation particularly focuses on two critical concerns: (1) the extent to which the generated instructions encompass the entire distribution of the target task, rather than merely echoing these initial examples (§ 4.4.1), and (2) the diversity of the generated instructions, specifically examining whether they demonstrate a broad spectrum of variation (§ 4.4.2).



(a) Semantic distribution of MBPP

(b) Semantic distribution of HumanEval

Figure 3: Semantic distribution of generated instructions by t-SNE. Ada-Instruct shows better semantic distribution consistency than Evol-Instruct.

4.4.1 Semantic Distribution

We plot the semantic distribution of the initial instructions and the generated instructions. Additionally, we plot the distribution of the target task for comparison, to verify whether the generated instructions align with the target distribution. For comparison, we also plot the distribution of instructions by Evol-Instruct. We represent the semantics of the instructions using *text-embedding-ada-002* API from OpenAI and visualized their distribution using t-SNE (Van der Maaten and Hinton, 2008).

Figure 3 shows that the generated instructions exhibit a consistent distribution with the target task. The instructions of Ada-Instruct are not confined to the vicinity of the ten initial samples but demonstrate the capability to expand to broader regions, aligning with the actual instruction distribution of the target task. In contrast, the Evol-Instruct distribution shows noticeable deviations from the target instruction distribution. Such gaps are not unusual - Evol-Instruct, which is based on multi-turn prompt

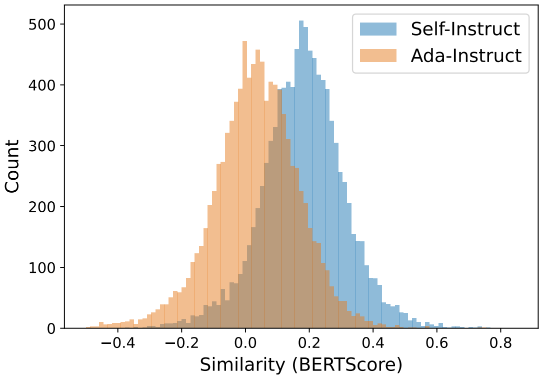


Figure 4: Similarity score distribution. Ada-Instruct generally has lower similarity scores than Self-Instruct, indicating that it has high diversity.

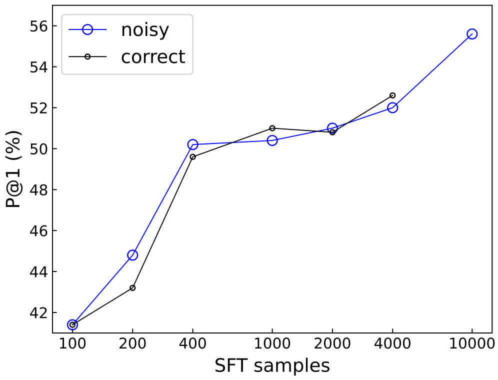


Figure 5: All generated instructions (noisy) vs correct instructions only on MBPP. The correctness is verified by test cases generated from gpt-3.5-turbo-instruct. Using noisy instructions does not cause a significant performance decline.

engineering, can generate long and complex instructions. However, crafting prompts manually without learning makes it difficult to fit the intended distribution. Ada-Instruct is capable of learning to adapt to the downstream instruction distribution. which is essential for instruction generation. These observations validate both Ada-Instruct's distributional consistency with respect to semantics, and the motivation of adapting LLMs as instruction generators for intended tasks.

4.4.2 Diversity

Given that our instruction generator was trained from merely 10 examples, another concern is whether the generated instructions are sufficiently diverse or if they overfit to a limited number of training samples. To address this, we assessed the diversity of the generated samples. Specifically, we randomly sampled 10000 pairs of generated samples for MBPP and calculated their similarity scores. A high similarity score for a pair of instructions indicates redundancy. Therefore, for a more diverse set of generated samples, we desire a lower similarity score distribution. We compared the diversity of instructions generated by Ada-Instruct and by Self-Instruct.

We followed the approach used in a previous work (Honovich et al., 2022) to employ BERTscore (Zhang et al., 2019) to measure the similarity between instruction pairs. The visualization of the results can be seen in Figure 4. The samples from Ada-Instruct exhibited lower similarity between pairs. This indicates that Ada-Instruct produces instructions with greater diversity. Given that the expressive capacity of the base model for Ada-Instruct (Code LLAMA) is evidently weaker than that of ChatGPT, this underscores the effectiveness of Ada-Instruct in generating diverse instructions.

4.5 The Impact of Instruction Quality

Ada-Instruct typically employs fine-tuning on open-source models, whereas Self-Instruct often uses closed-source models (like ChatGPT) for generating instructions. It is important to note that, as of now, the quality of open-source models generally lags behind that of closed-source models. Therefore, a concern with Ada-Instruct is that the quality of individual instructions might be lower, particularly for complex tasks. In this subsection, we investigate the actual impact on instruction quality.

We take MBPP as the object and examine how a decline in instruction quality affects the results. Specifically, we analyze the impact of using potentially erroneous instructions generated by Ada-Instruct (denoted as noisy samples) compared to using correct instructions. To determine the correctness of the instructions, given that MBPP samples include both code and use cases, we test whether the generated code passes through these cases successfully. Instructions that do so are considered correct samples. Among all noisy samples generated, we found that 46.9% are correct. We sampled different scales of generated noisy samples and correct samples, respectively, and compared the effects of training models on them in Figure 5.

We observed that the effects on the originally generated noisy samples are comparable to those based on correct samples, echoing a similar finding in (Honovich et al., 2022). This indicates that the difference in effectiveness between noisy sam-

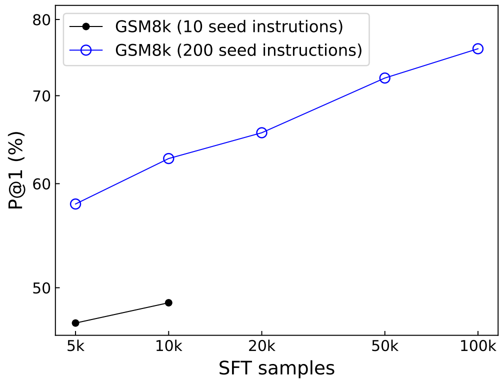


Figure 6: Impact of increasing both the number of seed samples and the number of SFT samples. Both x and y axes are presented on a log scale.

ples produced by open-source LLMs and those produced by closed-source LLMs might not be a significant concern in sample generation. Even for complex tasks like programming, the impact of using noisy instructions generated by Ada-Instruct appears to be minimal. This confirms Ada-Instruct's adaptability in handling instructional noise.

4.6 Scaling Up the Instructions

We further validate the efficacy of Ada-Instruct by increasing both the number of seed samples (for example, 200 seed instructions) and the scale of SFT samples. Figure 6 illustrates our experimental results on GSM8k. A larger set of seed instructions leads to improved performance. Under the condition of 200 seed instructions, the P@1 and the number of SFT samples exhibit a clear scaling law, with room for further improvement. This evidence substantiates that Ada-Instruct's performance significantly improves as the instruction size increases.

5 Conclusion

We unveil novel insights into the capabilities of instruction generation, demonstrating that the conventional ICL-based Self-Instruct fails to generate long and complex instructions. In contrast, we reveal the proficiency of fine-tuning in generating task-aligned instructions, even with a limited number of initial samples. We introduced Ada-Instruct, a novel few-shot instruction generation methodology that leverages the fine-tuning of open-source LLMs, diverging significantly from the prevalent self-instruct strategies based on in-context learning with closed-source LLMs. Ada-Instruct ensures the generation of coherent, high-quality, and diverse instructions that align well with the target task distribution, presenting a groundbreaking solution to the challenges of data sparsity and diversity in instruction generation.

6 Limitations

There are a few limitations worth noting:

- Reliance on closed-source LLMs for labeling: In the current implementation of Ada-Instruct, the labeling step relies on a closed-source LLM (e.g. ChatGPT). The performance and reliability of the labeling step are subject to the capabilities and limitations of the chosen closed-source LLM.

- Limited evaluation on more tasks: The experiments in this paper primarily focus on code completion, mathematical reasoning, and commonsense reasoning tasks. Further evaluation on a wider range of tasks is helpful to comprehensively assess the generalizability and effectiveness of Ada-Instruct.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Anas Awadalla, Mitchell Wortsman, Gabriel Ilharco, Sewon Min, Ian Magnusson, Hannaneh Hajishirzi, and Ludwig Schmidt. 2022. Exploring the landscape of distributional robustness for question answering models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 5971–5987.

Jiaao Chen, Derek Tam, Colin Raffel, Mohit Bansal, and Diyi Yang. 2023. An empirical survey of data augmentation for limited data learning in nlp. *Transactions of the Association for Computational Linguistics*, 11:191–211.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

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

Fabrizio Gilardi, Meysam Alizadeh, and Maël Kubli. 2023. Chatgpt outperforms crowd-workers for text-annotation tasks. *arXiv preprint arXiv:2303.15056*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Or Honovich, Thomas Scialom, Omer Levy, and Timo Schick. 2022. Unnatural instructions: Tuning language models with (almost) no human labor. *arXiv preprint arXiv:2212.09689*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. 2023. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023a. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023b. Wizardcoder: Empowering code large language models with evol-instruct. *arXiv preprint arXiv:2306.08568*.

Yu Meng, Jiaxin Huang, Yu Zhang, and Jiawei Han. 2022. Generating training data with language models: Towards zero-shot language understanding. *Advances in Neural Information Processing Systems*, 35:462–477.

Yu Meng, Martin Michalski, Jiaxin Huang, Yu Zhang, Tarek Abdelzaher, and Jiawei Han. 2023. Tuning language models as training data generators for augmentation-enhanced few-shot learning. In *International Conference on Machine Learning*, pages 24457–24477. PMLR.

Marius Mosbach, Tiago Pimentel, Shauli Ravfogel, Dietrich Klakow, and Yanai Elazar. 2023. Few-shot fine-tuning vs. in-context learning: A fair comparison and evaluation. *arXiv preprint arXiv:2305.16938*.

OpenAI. 2023. Gpt-4 technical report. *arXiv*, pages 2303–08774.

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

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Timo Schick and Hinrich Schütze. 2021. Generating datasets with pretrained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6943–6951.

Chenglei Si, Zhe Gan, Zhengyuan Yang, Shuohang Wang, Jianfeng Wang, Jordan Lee Boyd-Graber, and Lijuan Wang. 2022. Prompting gpt-3 to be reliable. In *The Eleventh International Conference on Learning Representations*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems*, 33:16857–16867.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. 2023. Principle-driven self-alignment of language models from scratch with minimal human supervision. *arXiv preprint arXiv:2305.03047*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. Commonsenseqa: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158.

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. `https://github.com/tatsu-lab/stanford_alpaca`.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Prasetya Utama, Nafise Sadat Moosavi, Victor Sanh, and Iryna Gurevych. 2021. Avoiding inference heuristics in few-shot prompt-based finetuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9063–9074.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language model with self generated instructions. *arXiv preprint arXiv:2212.10560*.

Shijie Wu, Ozan Irsoy, Steven Lu, Vadim Dabravolski, Mark Dredze, Sebastian Gehrmann, Prabhanjan Kambadur, David Rosenberg, and Gideon Mann. 2023. Bloomberggpt: A large language model for finance. *arXiv preprint arXiv:2303.17564*.

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*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Jiacheng Ye, Jiahui Gao, Qintong Li, Hang Xu, Jiangtao Feng, Zhiyong Wu, Tao Yu, and Lingpeng Kong. 2022. Zerogen: Efficient zero-shot learning via dataset generation. *arXiv preprint arXiv:2202.07922*.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. 2019. Bertscore: Evaluating text generation with bert. In *International Conference on Learning Representations*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2022. Large language models are human-level prompt engineers. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

A Quality Analysis

To assess the quality of the generated instructions, we evaluated whether the generated instructions are coherent and logically sound. For this evaluation, we used ChatGPT as an annotator. We randomly sampled 200 generated instructions for MBPP and CommonsenseQA. We first tell ChatGPT the task description of MBPP and CommonsenseQA, and then ask ChatGPT, “Do you think this instruction is coherent and logically sound? Yes or No.” As a baseline, we also evaluated the quality of the real samples from the corresponding data sets as the upper quality limit.

As can be seen in Table 6, the quality of the generated instructions is comparable to that of the real samples, suggesting that the generated samples possess sufficient accuracy. Although a small fraction of incorrect samples still exist, we investigated the impact of such errors in Section 4.5.

MBPP			CommonsenseQA		
Generated	Real Samples	Ratio	Generated	Real Samples	Ratio
80.5%	93.0%	86.6%	62.0%	65.0%	95.4%

Table 6: Quality of generated instructions, evaluated by ChatGPT. We compare with the real instructions, showing that their quality are close.

B Impact of Length on Performance

Ada-Instruct’s ability to generate longer instructions that align well with the target distribution contributes to its performance improvement. To directly validate the benefits of longer instructions experimentally, we selected HumanEval as the target task. We randomly sampled two sets of 5k instructions:

1. From all instructions generated by Ada-Instruct.

2. Only from instructions with lengths less than 90 (based on Figure 1, self-instruct rarely generates instructions longer than 90 tokens).

Length	HumanEval
Length < 90	57.9
Full Length	61.0

Table 7: Comparison of pass@1 (%) results on HumanEval using two distinct sets of 5k instructions.

As shown in Table 7, instructions sampled from the set that includes longer examples yield a higher pass@1 score.

C Training Details

When fine-tuning in Step 1, we train the models for 40 epochs with 10% warm-up steps for all tasks. We use a batch size of 10, a learning rate of 1e-6, a weight decay of 1e-2, a cosine learning rate scheduler, and bf16 precision for all tasks except for MATH. We find MATH much harder than other tasks, so we apply a lower learning rate of 8e-7 to better adapt to the task. For all tasks under consideration, we adopt the first checkpoint at which the loss value resides within the range of 0.2 to 0.4 to avoid overfitting. This checkpoint is selected from the 25th, 30th, 35th, and 40th training epochs.

In Step 1 of the generation process, the temperature is set to 1 for all tasks. To enhance diversity, we utilized top-k sampling. Specifically, for simpler MBPP and CSQA, we set $k = 100$, while for more complex HumanEval, GSM8K, and MATH, we set $k = 80$.

When fine-tuning in Step 3, for all tasks except HumanEval and CommonsenseQA, we train the LLMs for 3 epochs with a batch size of 256, a learning rate of 2e-5, a weight decay of 1e-2 and bf16 precision. We use a cosine scheduler with 10% warm-up steps. For HumanEval, we adopt a lower learning rate of 1e-5. For CommonsenseQA, we adopt 2 training epochs and a lower learning rate of 1e-5, given that the data points in this task are much shorter than those in other tasks. Similarly to (Rozière et al., 2023), we adopt a cosine scheduler with 15% warm-up steps and set the final learning rate to be 25% of the peak learning rate. We do not apply loss masking to the instruction for all tasks except for CommonsenseQA, as the output for CommonsenseQA consists of only a few tokens.

D Case Study

In Table 8, we present the instructions generated by Ada-Instruct on HumanEval. We observe that the instructions generated by Self-Instruct are predominantly short. Although Evol-Instruct can generate longer instructions by iteratively adding constraints, these instructions tend to be unnatural and do not align well with the distribution of the downstream tasks. In contrast, Ada-Instruct is capable of producing longer instructions that align well with the target task.

E Licenses for Artifacts

We list the artifacts used in this paper and their licenses below:

- (Touvron et al., 2023), llama2
- (Xu et al., 2023; Luo et al., 2023a,b), llama2
- (Wang et al., 2022), Apache-2.0 license

This work is consistent with their intended use.

F Evaluation Strategies

F.1 Prompts for Downstream Tasks

HumanEval:

```
[INST] You are an expert Python
programmer, complete the function
below based on its docstring and
the given test cases:
{Question}
Your code should start with a
[PYTHON] tag and end with a
[/PYTHON] tag. [/INST]
```

MBPP:

```
[INST] You are an expert Python
programmer, and here is your task:
{Question}
Your code should pass these tests:

{Test Cases}
Your code should start with a
[PYTHON] tag and end with a
[/PYTHON] tag. [/INST]
```

GSM8k and MATH:

```
[INST] You are expert at solving
math problems that require
multi-step reasoning, and here is
your task:
{Question} [/INST] Let's think
step by step.
```

CommonsenseQA:

```
[INST] You are expert at
commonsense reasoning, and here is
your task: {Question}
```

Model	Instruction
Self-Instruct	Given a list of words, create a dictionary to count the number of occurrences of each word.
Evol-Instruct	Create a program that can filter out words of a string that contain a specific character and have a length greater than 3. Additionally, if the character is a vowel, the program should replace it with the next vowel in the vowel sequence. The program should then output the modified string, while maintaining the original word order. Additionally, you need to handle cases where the string contains special characters or numbers. If a word contains any special characters or numbers, it should be excluded from the output.
Ada-Instruct	You are given an array of meeting time ranges in any order. Each meeting time ranges[i] = [start_i, end_i] means that you need attend a meeting during the time range [start_i, end_i). Return the minimum number of conference rooms required.

Table 8: Comparison of Generated Instructions for HumanEval: Instructions from Self-Instruct are overly simplistic. Instructions from Evol-Instruct, while longer, exhibit unnaturalness and lack alignment with the target distribution. In contrast, Ada-Instruct successfully generates longer instructions that are consistent with the target distribution (algorithmic problems).

```
A. {Text of Label A}
B. {Text of Label B}
C. {Text of Label C}
D. {Text of Label D}
E. {Text of Label E} [/INST] The
answer is:
```

F.2 Decoding Strategies

For code completion tasks, to ensure comparable evaluations, we follow (Rozière et al., 2023) and report the pass@1 scores of our models within the settings of greedy decoding and zero-shot.

For math tasks, to ensure comparable evaluations, we follow (Luo et al., 2023a) and report the pass@1 scores of our models within the settings of greedy decoding, zero-shot, and chain-of-thought.

For CommonsenseQA, the absence of an available test set necessitates the evaluation of our model on the development set. This evaluation is carried out within a framework adapted from (Hendrycks et al., 2020), and is executed in a zero-shot and answer-only manner. To ensure an equitable comparison, we also evaluate other LLAMA 2 base models in this setting.

G Fine-Tuning Data Formats for Ada-Instruct

G.1 Step 1

HumanEval:

```
 [INST] You are an expert Python
 programmer, complete the function
 below based on its docstring and
 the given test cases:
{Question}
 Your code should start with a
 [PYTHON] tag and end with a
 [/PYTHON] tag. [/INST] [PYTHON]
 # pass
 [/PYTHON]
```

MBPP:

```
 [INST] You are an expert Python
 programmer, and here is your task:
{Question}
 Your code should pass these tests:

{Test Cases}
 Your code should start with a
 [PYTHON] tag and end with a
 [/PYTHON] tag. [/INST] [PYTHON]
 # pass
 [/PYTHON]
```

GSM8k and MATH:

```
 [INST] You are expert at solving
 math problems that require
```

```
multi-step reasoning, and here is
your task:
{Question} [/INST] Let’s think
step by step.
```

CommonsenseQA:

```
[INST] You are expert at
commonsense reasoning, and here is
your task: {Question}
A. {Text of Label A}
B. {Text of Label B}
C. {Text of Label C}
D. {Text of Label D}
E. {Text of Label E} [/INST]
```

G.2 Step 3

HumanEval:

```
[INST] You are an expert Python
programmer, complete the function
below based on its docstring and
the given test cases:
{Question}
Your code should start with a
[PYTHON] tag and end with a
[/PYTHON] tag. [/INST] [PYTHON]
{Output}
[/PYTHON]
```

MBPP:

```
[INST] You are an expert Python
programmer, and here is your task:
{Question}
Your code should pass these tests:

{Test Cases}
Your code should start with a
[PYTHON] tag and end with a
[/PYTHON] tag. [/INST] [PYTHON]
{Output}
[/PYTHON]
```

GSM8k and MATH:

```
[INST] You are expert at solving
math problems that require
multi-step reasoning, and here is
your task:
{Question} [/INST] Let’s think
step by step.
{Output}
```

CommonsenseQA:

```
[INST] You are expert at
commonsense reasoning, and here is
your task: {Question}
A. {Text of Label A}
B. {Text of Label B}
C. {Text of Label C}
D. {Text of Label D}
E. {Text of Label E} [/INST] The
answer is: {Output}
```

H Prompts for Self-Instruct

To encourage the generation of high quality and diverse instruction, we use the following prompts in the Self-Instruct baseline.

H.1 Prompts For gpt-3.5-turbo-instruct

HumanEval:

```
You are asked to come up with a
set of 20 diverse instructions on
code completion task. These
instructions will be given to a
Codex model and we will evaluate
the Codex model for generating
codes that follow the
instructions.

Here are the requirements:
1. The instructions are designed
for testing the Python programming
capability to solve Python
problems. Each instruction should
describe a Python problem with
function definition, docstring,
and test cases.
2. The instructions should
incorporate as many Python
concepts as possible, as well as
being diverse and comprehensive.
3. The instructions should not be
too easy. Each Python problem
```

```
should be solved using built-in
libraries or data structures with
algorithm of intermediate level.
4. The instructions should at
least 1 to 2 sentences long.
Either an imperative sentence or a
question is permitted.
5. The output should be an
appropriate response to the
instruction, and should take full
account of requirements and test
cases in the instruction.
6. The instructions must not
appear in mainstream evaluation
datasets for code generation, e.g.
HumanEval, MBPP, DS1000 and so on.

List of 20 tasks:
###
1. {Example 1}
###
2. {Example 2}
###
3. {Example 3}
###
4.
```

MBPP:

```
You are asked to come up with a
set of 20 diverse instructions on
code completion task. These
instructions will be given to a
Codex model and we will evaluate
the Codex model for generating
codes that follow the
instructions.

Here are the requirements:
1. The instructions are designed
for testing the Python programming
capability to solve basic Python
problems. Each instruction should
have a clear and distinct
solution.
2. The instructions should
incorporate as many Python
concepts as possible, as well as
being diverse and comprehensive.
```

```
3. The instructions should not be
too complicated or too easy. Each
Python problem should be solved
using built-in libraries or data
structures with algorithm of
intermediate level.
4. The instructions should at
least 1 to 2 sentences long.
Either an imperative sentence or a
question is permitted.
5. The output should be an
appropriate response to the
instruction, and should take full
account of requirements and test
cases in the instruction.
6. The instructions must not
appear in mainstream evaluation
datasets for code generation, e.g.
HumanEval, MBPP, DS1000 and so on.

List of 20 tasks:
###
1. {Example 1}
###
2. {Example 2}
###
3. {Example 3}
###
4.
```

GSM8k:

```
You are asked to come up with a
set of 20 diverse instructions on
math problem solving task. These
instructions will be given to a
math model and we will evaluate
the math model for generating
solutions that follow the
instructions.

Here are the requirements:
1. The instructions are designed
for testing the math capability to
solve math problems that require
multi-step reasoning. Each
instruction should be accompanied
by a detailed reasoning path and a
final answer.
```

```
2. The instructions should include
diverse types of grade school math
problems, as well as being diverse
and comprehensive.
3. The instructions should not be
too complicated or too easy. Each
math problem should take between 2
and 8 steps to solve, and
solutions primarily involve
performing calculations using
basic arithmetic operations (+ - /
*) to reach the final answer.
4. The instructions should at
least 1 to 2 sentences long.
Either an imperative sentence or a
question is permitted.
5. The output should be an
appropriate response to the
instruction that is in the form of
reasoning followed by the final
answer.
6. The instructions must not
appear in mainstream evaluation
datasets for math, e.g. GSM8K,
MATH and so on.

List of 20 tasks:
###
1. {Example 1}
###
2. {Example 2}
###
3. {Example 3}
###
4.
```

MATH:

```
You are asked to come up with a
set of 20 diverse instructions on
math problem solving task. These
instructions will be given to a
math model and we will evaluate
the math model for generating
solutions that follow the
instructions.

Here are the requirements:
1. The instructions are designed
for testing the math capability to
solve math problems that require
multi-step reasoning. Each
instruction should be accompanied
by a detailed reasoning path and a
final answer.
2. The instructions should
describe math problems in LaTex
that require knowledge such as
calculus, algebra, number theory,
counting and probability, etc.
3. The instructions should be
challenging, diverse and
comprehensive. Each math problem
should take multiple steps of
complex reasoning maybe with some
advanced mathematical knowledge
and tools to solve.
4. The instructions should at
least 1 to 2 sentences long.
Either an imperative sentence or a
question is permitted.
5. The output should be an
appropriate response to the
instruction that is in the form of
reasoning followed by the final
answer. Both the reasoning and
answer should be in the form of
LaTex. The final answer should be
placed in ”$\boxed{}$”.
6. The instructions must not
appear in mainstream evaluation
datasets for math, e.g. GSM8K,
MATH and so on.

List of 20 tasks:
###
1. {Example 1}
###
2. {Example 2}
###
3. {Example 3}
###
4.
```

H.2 Prompts For gpt-4o

HumanEval:

```
user: You are asked to come up
with a set of 10 diverse
instructions on code completion
```

task. These instructions will be given to a Codex model and we will evaluate the Codex model for generating codes that follow the instructions.

Here are the requirements:
1. The instructions are designed for testing the Python programming capability to solve Python problems. Each instruction should describe a Python problem with function definition, docstring, and test cases.
2. The instructions should incorporate as many Python concepts as possible, as well as being diverse and comprehensive.
3. The instructions should not be too easy. Each Python problem should be solved using built-in libraries or data structures with algorithm of intermediate level.
4. The instructions should at least 1 to 2 sentences long. Either an imperative sentence or a question is permitted.
5. The output should be an appropriate response to the instruction, and should take full account of requirements and test cases in the instruction.
6. The instructions must not appear in mainstream evaluation datasets for code generation, e.g. HumanEval, MBPP, DS1000 and so on.

assistant: ###
1. {Example 1}
###
2. {Example 2}
###
3. {Example 3}
###

user: Continue to generate the remaining 7 instructions. The order number of each instruction must be preceded by "###".

user: You are asked to come up with a set of 10 diverse instructions on math problem solving task. These instructions will be given to a math model and we will evaluate the math model for generating solutions that follow the instructions.

Here are the requirements:
1. The instructions are designed for testing the math capability to solve math problems that require multi-step reasoning. Each instruction should be accompanied by a detailed reasoning path and a final answer.
2. The instructions should include diverse types of grade school math problems, as well as being diverse and comprehensive.
3. The instructions should not be too complicated or too easy. Each math problem should take between 2 and 8 steps to solve, and solutions primarily involve performing calculations using basic arithmetic operations (+ - / *) to reach the final answer.
4. The instructions should at least 1 to 2 sentences long. Either an imperative sentence or a question is permitted.
5. The output should be an appropriate response to the instruction that is in the form of reasoning followed by the final answer.
6. The instructions must not appear in mainstream evaluation datasets for math, e.g. GSM8K, MATH and so on.

assistant: ###
1. {Example 1}
###
2. {Example 2}
###
3. {Example 3}

GSM8k:

###

user: Continue to generate the remaining 7 instructions. The order number of each instruction must be preceded by "###".