Smaller Language Models Are Better Instruction Evolvers

Anonymous ACL submission

Abstract

Instruction tuning has been widely used to unleash the complete potential of large language models. Notably, complex and diverse instructions are of significant importance as they can effectively align models with various downstream tasks. However, current approaches to constructing large-scale instructions predominantly favour powerful models such as GPT-4 or those with over 70 billion parameters, under the empirical presumption that such larger language models (LLMs) inherently possess enhanced capabilities. In this study, we question this prevalent assumption and conduct an in-depth exploration into the potential of smaller language models (SLMs) in the context of instruction evolution. Extensive experiments across three scenarios of instruction evolution reveal that smaller language models (SLMs) can synthesize more effective instructions than LLMs. Further analysis demonstrates that SLMs possess a broader output space during instruction evolution, resulting in more complex and diverse variants. We also observe that the existing metrics fail to focus on the impact of the instructions. Thus, we propose Instruction Complex-Aware IFD (IC-IFD), which introduces instruction complexity in the original IFD score to evaluate the effectiveness of instruction data more accurately. Our source code is available at: https://anonymous.4open.science/r/Evolution-Analysis-8BB9

1 Introduction

Large Language Models (LLMs) have demonstrated exceptional performance in various NLP tasks and are widely integrated into a variety of applications, represented by ChatGPT and Copilot (Ouyang et al., 2022; OpenAI, 2023; Dubey et al., 2024). A key factor in unleashing the full potential of these models is high-quality instruction tuning data, which plays a crucial role in post-training and enhances their effectiveness as AI assistants. In particular, incorporating more complex and diverse instructions allows models to better align with different domains and tasks, boosting their performance in a variety of downstream applications (Zhang et al., 2023). However, generating such diverse instructions remains time-consuming and labor intensive (Zheng et al., 2024a; Zhao et al., 2024; Liu et al., 2024), which undoubtedly presents a significant challenge for the automated and scalable alignment of LLMs. Recently, a series of efforts utilizing LLMs for automatic instruction evolution have garnered sustained attention from the community. Specifically, foundational work like Self-Instruct (Wang et al., 2023) begins with a small set of seed instructions and uses a powerful supervision model to obtain a large number of synthetic instructions. Furthermore, Evol-Instruct (Xu et al., 2024a) refines and evolves existing instructions to produce more complex variants.

However, previous studies mainly favour strong LLMs like GPT-4 or those with more than 70 billion parameters to synthesize instructions, empirically assuming that larger language models inherently have superior instruction evolution capabilities. But is this really the case? Recently, Xu et al. (2024c) propose the Larger Models' Paradox, which points out that larger models do not necessarily lead to better performance when generating responses, but it overlooks the analysis of instructions. We propose that smaller language models, which require less computational demand and have lower instruction following capabilities, may provide a more efficient and effective alternative for evolving more complex and diverse instructions. To gain insight into this, we investigate the differences between smaller language models (SLMs) and larger language models (LLMs) in generating high-quality instructions. Specifically, given a set of base models and seed instructions, we are particularly interested in the following research question:

RQ1: Do SLMs Perform Better than LLMs in Evolving Instructions?

In response to this, we conduct comprehensive experiments across three distinct instruction evolution scenarios: Evol-Instruct, AutoIF (Dong et al., 2024), and Auto Evol-Instruct (Zeng et al., 2024). In these experiments, we use small (∼8B) and large (∼70B) models from the Llama-3.1 and Qwen-2 families to evolve and synthesize new instructions, while also fine-tuning various backbone models. The experimental results across all three scenarios consistently indicate that larger, more powerful LLMs do not outperform SLMs in evolving effective instructions. More interestingly, SLMs even demonstrate the capability to evolve more complex and diverse instructions. To further investigate why more powerful LLMs perform worse than SLMs in generating new instructions, we subsequently pose the second research question.

RQ2: Why do SLMs Outerperform LLMs in Evolving Instructions?

To better understand why more powerful LLMs underperform compared to SLMs in evolving instructions, we compare the top-1 token probabilities of both models during the synthetic of instructions. Our findings demonstrate that LLMs, due to their superior instruction following capabilities, tend to generate a higher proportion of high-probability top-1 tokens when evolving new instructions. This overconfidence in token generation results in a narrower output space. In contrast, SLMs can generate a wider variety of tokens, leading to more complex and diverse instructions. To further investigate what kind of instruction data is effective, we propose the third research question.

RQ3: How Do We Determine Whether An Instruction is Effective without Instruction Tuning?

Evaluations that do not require instruction tuning can more efficiently assess instruction data. Recent such evaluations often fail to account for the impact of the instructions themselves. For instance, reward models (Cai et al., 2024) are commonly used to assess the quality of responses generated based on a given instruction, yet they tend to overlook the quality of the instruction itself. Similarly, while the IFD score (Li et al., 2024) measures the influence of instructions on response generation, it neglects the effect of the instruction's inherent complexity. We introduce the Instruction Complex-Aware IFD (IC-IFD) score, which incorporates the difficulty of the instruction as a penalty term in the original IFD. We conduct extensive filtering instruction data experiments, and the results demonstrate that the IC-IFD score provides a more accurate assessment of instruction data, particularly in scenarios where the instructions exhibit higher complexity levels. In summary, our key contributions are as follows:

(1) To the best of our knowledge, we are the first to comprehensively explore the performance discrepancies between SLMs and LLMs in synthesizing instructions.

(2) Extensive experimental results demonstrate that SLMs have a broader output space, leading to evolving more complex and diverse instructions.

(3) We propose the IC-IFD score, which introduces the difficulty of the instruction as a penalty term. Comprehensive experiments show that IC-IFD can more accurately assess the effectiveness of instruction data without instruction tuning.

2 Preliminaries

(Auto) Evol-Instruct. The goal of (Auto) Evol Instruct is to refine original instructions by using artificially designed or LLM-generated evolutionary trajectories, thereby increasing their complexity and fostering the development of a more capable model. Formally, given an instruction evolution model Θ_e, a response generation model Θ_r, and an original instruction dataset $\mathcal{D} = \{(\mathcal{I}_i, \mathcal{R}_i)\}_{i=1}^n$, where $\mathcal{I}$ and $\mathcal{R}$ are instructions and responses and n represents the data size, we employ either artificially designed methods or the Θ_e-generated evolutionary trajectory $\mathcal{T}$ to obtain more complex and diverse evolutionary dataset $D_{evol} = \{(\mathcal{I}_{ei} = \Theta_e(\mathcal{I}_i|\mathcal{T}), \mathcal{R}_{ei} = \Theta_r(\mathcal{R}|\mathcal{I}_{ei}))\}_{i=1}^n$.

AutoIF. The goal of AutoIF is to automatically construct large-scale and reliable instructions from a small set of seed instructions (which can also be seen as constraints) to improve instruction following ability. In this paper, we only utilize the first several steps of AutoIF. Specifically, given a small set of seed instructions $\mathcal{I}_s$, we first prompt the supervised model Θ to construct a large number of verifiable instructions $\mathcal{I}_{new}$ based on $\mathcal{I}_s$. Subsequently, we prompt Θ to generate the corresponding verification functions f and test cases c for $\mathcal{I} = \{\mathcal{I}_s, \mathcal{I}_{new}\}$. Finally, cross-validation is performed to obtain the final scalable and reliable instructions $\mathcal{I}_{final} = \{\mathcal{I}|f(\mathcal{I}, c) = True\}$.

Model	Instruction Following (IFEval)				Math Reasoning		Code Generation	
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	GSM8K	MATH	HumanEval	MBPP
Supervised Model: Llama-3.1-70B-Instruct								
Mistral-7B-v0.3	19.59	31.77	22.74	34.65	33.89	**3.16**	24.39	6.00
DeepSeek-7B	36.23	**48.20**	41.04	52.52	**48.07**	2.96	28.66	33.00
Llama-3.2-3B	40.11	50.84	43.81	54.43	53.75	6.60	35.98	**36.00**
Llama-3-8B	33.83	46.28	36.41	49.28	63.00	7.62	43.90	36.20
Llama-3.1-8B	34.57	46.04	38.81	50.48	64.22	11.32	**51.22**	40.60
InternLM-2-7B	40.85	53.48	44.54	56.95	**68.31**	19.50	56.10	40.40
Supervised Model: Llama-3.1-8B-Instruct								
Mistral-7B-v0.3	**24.40**	**35.01**	**26.25**	**37.53**	**40.18**	2.84	**29.27**	**19.60**
DeepSeek-7B	**36.60**	48.08	**41.77**	**53.12**	47.92	**3.56**	**34.76**	**33.80**
Llama-3.2-3B	**41.59**	**53.48**	**45.66**	**57.07**	**55.12**	**7.32**	**39.02**	32.80
Llama-3-8B	**35.49**	**47.00**	**39.56**	**50.72**	**63.38**	**11.44**	**48.17**	**37.60**
Llama-3.1-8B	**38.45**	**50.96**	**43.81**	**55.28**	**67.10**	**13.12**	48.78	**41.60**
InternLM-2-7B	**43.07**	**54.80**	**47.32**	**58.39**	68.08	**20.32**	**57.93**	**40.80**

Table 1: Comparison of performance with Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct as supervised models under Evol-Instruct scenario.

Model	Instruction Following (IFEval)				Math Reasoning		Code Generation	
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	GSM8K	MATH	HumanEval	MBPP
Supervised Model: Qwen-2-72B-Instruct								
Mistral-7B-v0.3	20.15	30.94	23.84	34.41	46.93	**3.26**	32.32	1.80
DeepSeek-7B	35.67	47.12	**39.56**	50.84	44.81	2.76	**36.59**	**34.00**
Llama-3.2-3B	39.74	51.44	43.99	55.40	53.83	**7.40**	38.41	31.00
Llama-3-8B	34.75	45.80	37.71	48.92	63.76	**10.06**	43.90	35.40
Llama-3.1-8B	**36.41**	**47.60**	39.00	50.60	65.43	10.84	**48.17**	38.40
InternLM-2-7B	41.96	53.60	43.99	55.64	65.28	17.96	56.71	40.60
Supervised Model: Qwen-2-7B-Instruct								
Mistral-7B-v0.3	**25.32**	**37.17**	**29.76**	**41.01**	**47.31**	2.20	**32.93**	**12.00**
DeepSeek-7B	**36.41**	**48.56**	39.37	**51.32**	**48.07**	**3.82**	35.37	33.20
Llama-3.2-3B	**43.81**	**55.16**	**47.87**	**58.27**	**56.56**	7.18	**39.63**	**31.40**
Llama-3-8B	**38.92**	**48.33**	**43.81**	**52.19**	**63.91**	8.66	**45.73**	**38.40**
Llama-3.1-8B	34.75	45.80	**39.93**	**51.08**	**68.76**	**14.02**	46.34	**38.60**
InternLM-2-7B	**44.12**	**55.16**	**48.62**	**58.73**	**66.87**	**19.60**	**58.54**	**41.40**

Table 2: Comparison of performance with Qwen-2-7B-Instruct and Qwen-2-72B-Instruct as supervised models under Evol-Instruct scenario.

3 RQ1: Do SLMs Perform Better than LLMs in Evolving Instructions?

In this section, we investigate the potential of SLMs in evolving complex and diverse instructions across three distinct scenarios: Evol-Instruct, AutoIF, and Auto Evol-Instruct. Through a series of comprehensive experiments and analyses, we attempt to answer the questions raised in RQ1. For clarity, we will refer to the instruction data evolved by SLMs and LLMs as SLM-INST and LLM-INST. The implementation details for the three scenarios, as well as our experimental hyperparameters, can be found in Appendix A.1.

3.1 Evol-Instruct Scenario

In this section, we primarily focus on *whether SLMs can evolve more complex and challenging instruction data compared to LLMs.*

Seed Datasets. Following (Xu et al., 2024a; Zeng et al., 2024), we utilize the following seed datasets for instruction following, mathematical reasoning and code generation: (1) Alpaca (Taori et al., 2023), (2) GSM8K Train (Cobbe et al., 2021), and (3) Code Alpaca (Chaudhary, 2023). More detailed information can be found in Appendix A.2.

Evaluation Benchmarks and Metrics. We use IFEval (Zhou et al., 2023b) to assess instruction following capability, GSM8K and MATH (Hendrycks et al., 2021b) to evaluate mathematical reasoning ability, and HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) to assess code generation performance. For detailed information, please refer to Appendix A.3.

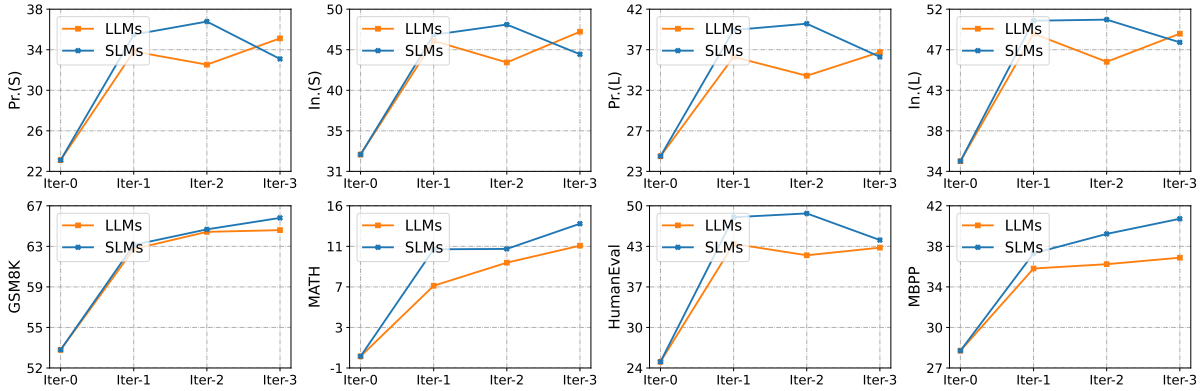

Figure 1: Comparison of performance on Llama-3-8B during three iterations of instruction evolution, using Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct as supervised models for each round under Evol-Instruct scenario.

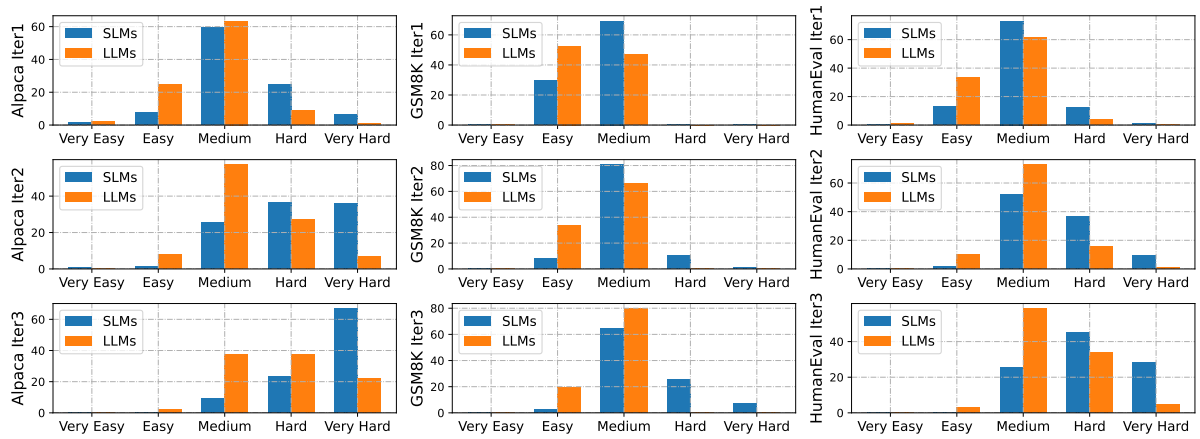

Figure 2: Distribution of difficulty levels for instructions evolved during three iterations, using Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct as supervised models for each round under Evol-Instruct scenario.

Results of Evol-Instruct. We conduct two sets of experiments, using the Llama-3.1 (Dubey et al., 2024) and Qwen-2 (Yang et al., 2024) model series for instruction evolution. This approach helps eliminate potential biases specific to each model series, ensuring the generalizability of the conclusions. Specifically, we use Llama-3.1-8B-Instruct and Qwen-2-7B-Instruct as SLMs and Llama-3.1-70B-Instruct and Qwen-2-72B-Instruct serve as LLMs for instruction evolution. To ensure that the generated responses do not influence the experimental conclusions, we consistently use Qwen-2.5-72B-Instruct (Team, 2024) as the response generator.

Table 1 and Table 2 present a comparative analysis of benchmark results for SLM-INST and LLM-INST using the Llama and Qwen model families, highlighting the following key insights[1].

(1) We find that SLM-INST outperforms LLM-INST across instruction following, mathematical reasoning, and code generation, demonstrating superior overall performance in both the Llama and Qwen model families.

(2) More complex and difficult instruction data leads to more effective improvements in instruction following capabilities (Dong et al., 2024). Our results show that SLM-INST significantly outperforms LLM-INST on IFEval, highlighting the ability of SLMs to generate more complex instructions compared to LLMs.

Impact of Evolution Iteration. Figure 1 illustrates the performance of Llama-3-8B after three rounds of evolution with the Llama-3.1 series (Detailed results can be found in Table 10). Iter 0 represents the performance of the seed instruction data and we release the following key insights.

(1) We find that during the first two rounds of evolution, the SLM-INST consistently outperforms LLM-INST. Notably, in terms of the instruction

[1]More results and analyses regarding the performance of seed instruction data and the impact of temperature are provided in Appendix A.4.

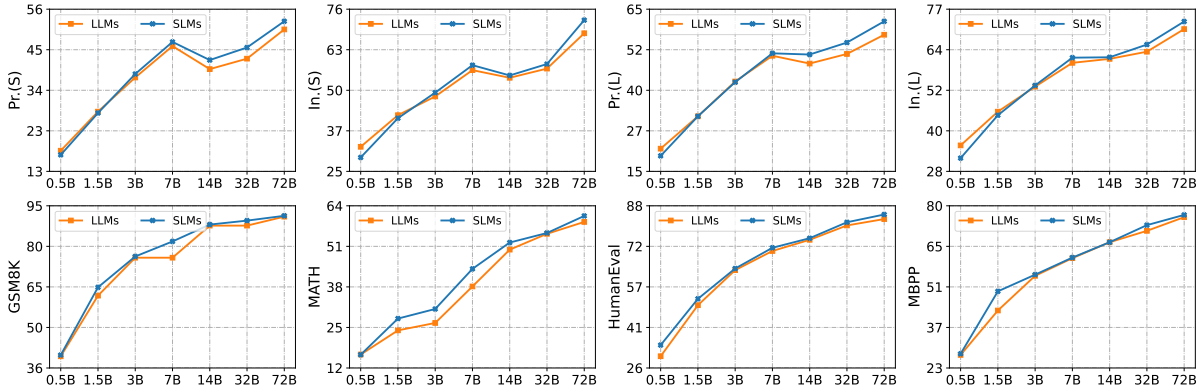

Figure 3: Comparison of performance among Qwen-2.5 series models. Detailed results can be found in Table 11.

following, LLM-INST even experiences negative growth, further proving that SLMs are superior to LLMs in generating complex instructions.

(2) The performance in the third round of evolution shows an interesting phenomenon. While the SLM-INST continues to perform well in mathematical reasoning, there is a significant drop in both instruction following and code generation. Following (Xu et al., 2024b), we use Qwen-2.5-72B-Instruct to assess the difficulty level of the evolved instructions in each round, as shown in Figure 2. We find that the difficulty of the SLM-INST in the third round is excessively high. For example, in the third-round SLM-INST for Alpaca, nearly 70% of the instructions are categorized as "very hard". Such overly complex and difficult-to-understand instructions result in a decline in performance. Further data analysis and the evaluation prompt templates can be found in Appendix A.5 and Figure 17.

(3) We find that the complexity of SLM-INST in the second iteration surpasses that of LLM-INST in the third iteration, with SLM-INST also demonstrating superior performance. This suggests that we can leverage SLMs to generate more complex and challenging instructions with fewer computational resources and evolutionary iterations, while simultaneously achieving better performance.

Scaling Experiments. To further validate whether our findings hold across models of different sizes, we train models of various sizes within the Qwen-2.5 series (ranging from 0.5B to 72B). The training details can be found in Table 7. Due to computational resource constraints, we perform full fine-tuning for models ranging from 0.5B to 7B, while applying LoRA (Hu et al., 2022) for models from 14B to 72B. To avoid introducing additional biases, we switch the response generator to Llama-3.1-70B-Instruct during the training of the Qwen-2.5 series models. As shown in Figure 3. We find that in the instruction following evaluation, SLM-INST performs slightly worse than LLM-INST on 0.5B and 1.5B models. We believe this is because the evolved instructions in Alpaca are too challenging, and smaller models with lower capabilities may struggle to understand the instructions, leading to performance discrepancies. However, in other evaluations, SLM-INST shows consistently better performance which further confirms our findings.

Finding 1

SLMs can evolve more complex and challenging instructions than LLMs.

3.2 AutoIF Scenario

In this section, we mainly concentrate on *whether SLMs can generate more diverse instruction data compared to LLMs.*

Evaluation Benchmarks and Metrics. We fully adhere to the evaluation benchmarks used in AutoIF. Specifically, we utilize IFEval and FollowBench (Jiang et al., 2024) to assess instruction following capabilities[2]. We also evaluate our models on C-Eval (Huang et al., 2023), MMLU (Hendrycks et al., 2021a), GSM8K, and HumanEval to obtain a comprehensive assessment of their capabilities. For detailed information, please refer to Appendix A.3.

Results of AutoIF. We use the Llama-3.1 series models for synthesizing instructions and we adopt Qwen-2.5-72B-Instruct for generating responses

[2]We use the Microsoft Azure OpenAI GPT-4 API.

Model	IFEval				FollowBench (HSR)						Common Abilities			
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	Level 1	Level 2	Level 3	Level 4	Level 5	Avg.	C-Eval	MMLU	HumanEval	GSM8K
Supervision Model: Llama-3.1-70B-Instruct														
Llama-3.2-3B	40.85	51.92	42.33	53.84	**61.17**	57.59	**50.55**	33.09	26.74	45.83	**41.37**	**52.65**	**29.88**	27.07
Llama-3-8B	37.71	50.00	39.19	52.04	49.64	46.60	41.56	27.05	22.37	37.44	41.87	51.14	26.83	37.76
Llama-3.1-8B	41.96	53.36	42.70	54.20	51.77	45.60	45.04	34.85	26.61	40.78	**44.50**	56.39	31.10	38.21
Qwen-2-7B	41.96	53.60	43.62	55.64	72.18	62.45	**56.43**	41.31	35.42	53.56	**81.08**	55.71	57.32	**79.68**
Qwen-2.5-7B	49.17	**60.31**	50.46	61.51	**78.88**	**73.78**	**61.50**	51.99	45.42	**62.31**	**80.46**	58.39	67.68	**85.90**
InternLM-2-7B	46.21	56.71	48.06	58.63	68.89	62.23	54.17	44.27	42.06	54.33	60.11	60.59	65.35	50.00
Supervision Model: Llama-3.1-8B-Instruct														
Llama-3.2-3B	**43.62**	**54.20**	**46.95**	**57.07**	56.95	**61.46**	50.20	**37.65**	**34.16**	**48.08**	40.56	49.08	25.00	**29.87**
Llama-3-8B	**41.04**	**51.32**	**42.88**	**53.11**	**62.99**	**54.38**	**49.29**	**32.21**	**32.21**	**46.21**	**43.49**	**55.63**	**37.20**	**45.26**
Llama-3.1-8B	**42.51**	**54.92**	**44.73**	**56.71**	**63.99**	**58.15**	**53.29**	**39.49**	**36.02**	**50.19**	43.77	**58.32**	**32.32**	**47.92**
Qwen-2-7B	**44.92**	**55.76**	**47.50**	**58.39**	**78.75**	**63.30**	52.31	**50.28**	**43.08**	**57.54**	80.11	**56.84**	**65.24**	79.53
Qwen-2.5-7B	**50.09**	59.59	**52.50**	**61.75**	77.86	70.22	59.86	**53.35**	**47.18**	61.69	79.74	**60.17**	**72.56**	84.69
InternLM-2-7B	**47.50**	**57.67**	**50.83**	**61.15**	**74.73**	**66.16**	**61.94**	**54.10**	**46.28**	**60.64**	**63.03**	**63.16**	**70.96**	**54.27**

Table 3: Comparison of performance with Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct as supervised models under AutoIF scenario.

under the AutoIF scenario. As shown in Table 3, on the IFEval and FollowBench instruction following benchmarks, the instruction data augmented by SLMs achieved better performance. Especially on FollowBench, SLM-INST even achieve nearly a 10% improvement over Llama-3-8B and Llama-3.1-8B. Meanwhile, on common abilities, SLM-INST also demonstrates competitive performance.

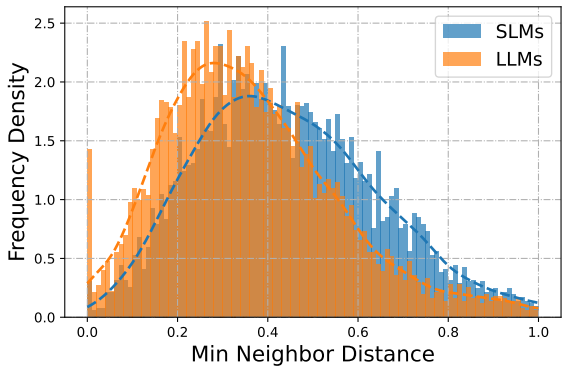

Figure 4: Distribution of Minimum Neighbor Distance for instructions generated by Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct in the AutoIF scenario.

AutoIF begins with a small set of manually crafted seed instructions, from which the model draws inspiration to generate a large number of new instructions and perform verifications to ensure their quality. Since the generated instructions have undergone multiple rounds of verification, their diversity becomes even more crucial. Following (Xu et al., 2024b), we use all-mpnet-base-v2 (Song et al., 2020) to measure similarity via minimum neighbor distance (MND) in the embedding space. Notably, a high number of samples with low MND suggests poor diversity within the dataset. Figure 4 demonstrates that SLM-INST has more samples with a larger MND, indicating higher diversity than LLM-INST.

Finding 2

SLMs can generate more diverse instructions than LLMs.

3.3 Auto Evol-Instruct Scenario

In this section, we mainly focus on *whether SLMs can automatically evolve more effective instructions compared to LLMs.*

Results of Auto Evol-Instruct. As shown in Table 4, we find that the instruction data automatically evolved by SLMs consistently performs better across the Llama series models than LLMs. In addition, we prompt the Qwen-2.5-72B-Instruct model to summarize and deduplicate keywords from the trajectories generated by SLMs and LLMs (the prompt template can be found in Figure 18). We find that the number of trajectories produced by SLMs is 6.9% higher than that of LLMs, further highlighting that SLMs can design more varied evolutionary trajectories, leading to more complex and diverse instructions.

Finding 3

SLMs can automatically evolve more effective instructions than LLMs.

4 RQ2: Why Do SLMs Outperform LLMs in Evolving Instructions?

In this section, we primarily analyze why SLMs perform better from the perspectives of model inference and real-world cases.

Model	Instruction Following (IFEval)				Math Reasoning		Code Generation	
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	GSM8K	MATH	HumanEval	MBPP
Supervised Model: Llama-3.1-70B-Instruct								
Llama-3.2-3B	36.60	48.68	39.00	51.08	53.60	7.56	35.37	33.00
Llama-3-8B	35.86	47.60	38.63	50.24	63.91	9.18	38.41	32.40
Llama-3.1-8B	36.97	47.60	40.30	51.08	66.11	11.68	40.85	**40.40**
Supervised Model: Llama-3.1-8B-Instruct								
Llama-3.2-3B	**45.47**	**57.43**	**50.28**	**61.27**	**56.48**	**8.42**	**38.41**	**34.40**
Llama-3-8B	**37.34**	**49.64**	**39.74**	**51.56**	**67.40**	**12.26**	**43.90**	**34.80**
Llama-3.1-8B	**38.08**	**49.76**	**40.48**	**52.40**	**69.52**	**15.62**	**51.22**	38.80

Table 4: Comparison of performance with Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct as supervised models under Auto Evol-Instruct scenario.

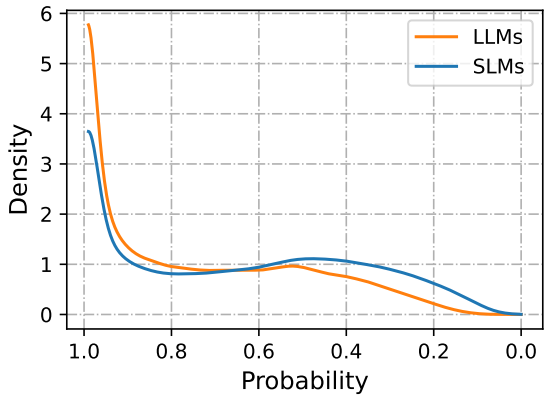

Figure 5: Comparison of output token probability distributions in the Evol-Instruct scenario.

Comparison of Token Distributions. The results of our previous experiments indicate that SLMs are capable of evolving and generating more complex and diverse instructions. We hypothesize that this is due to the superior instruction following capabilities of LLMs, which result in a narrower output space (overconfidence) when following instructions, thereby leading to less diversity and complexity in the generated new instructions. To validate this hypothesis, we employ the Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct models within the Evol-Instruct scenario to obtain the probability distributions of output tokens. By extracting the top-1 token probability at each output position, we compare the output probability distributions between SLMs and LLMs. As shown in Figure 5, we observe that the top-1 token output probability for SLMs is lower, suggesting that the output distribution of SLMs is more diverse. This supports our hypothesis that, due to their relatively weaker instruction following capabilities compared to LLMs, SLMs generate a broader output space, leading to more diverse and complex instructions. We also analyze some cases, and the detailed results can be found in Appendix A.4.

Finding 4

SLMs have a broader output space and are less likely to be overconfident than LLMs.

5 RQ3: How Do We Determine Whether An Instruction is Effective without Instruction Tuning?

In this section, we primarily discuss how to determine whether instruction data is effective without instruction tuning.

Instruction Complex-Aware IFD. As mentioned in (Xu et al., 2024c), existing evaluations typically focus on assessing responses, such as using reward models, while neglecting the impact of instructions on the data. Recently, Li et al. (2024) proposed the instruction following Difficulty (IFD) score to evaluate the quality of instructions. Specifically, the formula for IFD is as follows.

$$\text{IFD}_{\Theta}(Q, A) = \frac{L_{\Theta}(A|Q)}{L_{\Theta}(A)} \quad (1)$$

Where Q and A represent instructions and responses, and $L_{\Theta}(\cdot)$ represents the average cross entropy loss determined by a model Θ. IFD can be understood as the importance of instructions in generating responses. A lower IFD means that a sample does not require training, as the model is already able to generate the corresponding response effectively when given the instruction. However, as shown in Figure 1 and Table 15, when the difficulty of the instructions is too high, it may result in a higher IFD, but the overall performance may

fall short of expectations. Inspired by this, we introduce the difficulty level of instructions into the original IFD and propose the Instruction Complex-Aware IFD (IC-IFD). Specifically, we introduce the perplexity of the instructions into the original IFD score, resulting in the following formula.

$$\text{IC-IFD}_{\Theta}(Q, A) = \frac{L_{\Theta}(A|Q)}{L_{\Theta}(Q) \cdot L_{\Theta}(A)} \quad (2)$$

Metrics	IFEval			
	Pr.(S)	**In.(S)**	**Pr.(L)**	**In.(L)**
Original	33.09	44.72	36.41	48.32
Instruction Len.	29.94	39.69	33.83	43.53
Instruction PPL	27.91	39.69	32.35	44.36
IFD	30.87	43.53	36.04	47.60
IC-IFD	**34.01**	**46.16**	**38.82**	**50.72**

Table 5: Comparison of different metrics under 25% of Alpaca-iter3 evolved by SLMs on Llama-3-8B.

Performance of IC-IFD. To validate the effectiveness of IC-IFD, we aim to mitigate the performance degradation caused by the third round of instruction data evolved by SLMs. Specifically, we retain the top 25% of instruction data using several metrics, including instruction length (filtering out overly long instructions), instruction perplexity (PPL, filtering out instructions with excessively high PPL), IFD, and IC-IFD. As shown in Table 5, under the condition of retaining only 25% of the instruction data, IC-IFD outperforms the full dataset, while other metrics exhibit varying degrees of performance degradation, thereby demonstrating the effectiveness of IC-IFD. Further experiments on IC-IFD can be found in Appendix A.4.

6 Related Work

Instruction tuning has become an essential strategy for enhancing the capabilities of large language models (LLMs) (Ouyang et al., 2022; OpenAI, 2023). By curating high-quality datasets, we can more effectively align these models with specific objectives (Zhou et al., 2023a). Recently, some researchers have highlighted the significance of instruction data that is either manually annotated or developed with human involvement, such as ShareGPT (Chiang et al., 2023) and OpenAssistant (Köpf et al., 2023). Meanwhile, other studies concentrate on leveraging LLMs to generate high-quality datasets with minimal human effort (Xu et al., 2024a; Luo et al., 2024, 2023). Wang et al. (2023) introduces Self-Instruct, which begins with a small collection of manually crafted seed instructions and utilizes LLMs to expand these instructions, ultimately producing a large-scale instruction set that improves model abilities. Xu et al. (2024a) presents Evol-Instruct, which employs LLMs for the iterative enhancement of the original instructions through both in-depth and breadth evolution, resulting in a more complex and diverse instruction dataset. Auto Evol-Instruct (Zeng et al., 2024) further removes human involvement, enabling LLMs to autonomously design the evolution trajectory based on the original instructions. AutoIF (Dong et al., 2024) introduces a code feedback mechanism that allows LLMs to generate evaluation code for verifying whether the quality of the instructions meets the required standards. Xu et al. (2024b) only provides a single prompt to induce the model to generate a large amount of instruction data. Current research primarily focuses on utilizing larger language models, such as GPT-4 (OpenAI, 2023), for constructing complex instructions. More recently, Xu et al. (2024c) explores the performance differences of various-sized models as response generators. In contrast, we concentrate on the potential of smaller language models in evolving complex instructions. This innovation not only reduces the costs associated with instructions construction but, more importantly, offers a comprehensive evaluation and exploration, highlighting the significant capabilities inherent in smaller models and providing valuable insights for future work.

7 Conclusion

In this paper, we compare the performance of SLMs and LLMs in evolving instructions. Extensive experiments demonstrate that SLMs can synthesize more effective instructions at a lower computational cost than LLMs. Through an analysis of the model output distributions, we observe that SLMs exhibit a broader output space, leading to more complex and diverse instructions. Furthermore, we introduce instruction complexity as a penalty term in the original IFD and propose IC-IFD, which allows for more accurate assessment of instruction data effectiveness without the need for instruction tuning. Our work also lays the groundwork for future research on SLMs in instruction data synthesis, offering a foundation understanding for further exploration.

Limitations

Although our work provides valuable insights that SLMs perform better in evolving instructions through comprehensive experiments, several directions are worth exploring in future research.

(1) We have only conducted experiments in instruction following, mathematical reasoning, and code generation. We have not focused on other broader domains, and there may have interesting discoveries in these areas that require future work.

(2) Our work focuses on comparing SLMs and LLMs in evolving instruction sets, rather than exploring the full potential of SLMs in synthesizing entire instruction datasets. Future research that investigates the capabilities of SLMs across the entire instruction data synthesis pipeline would be a promising and exciting direction to explore.

(3) The IC-IFD we propose is based on our observation that performance degrades with the emergence of high-difficulty instructions, which leads us to introduce instruction complexity as a penalty term in the original IFD. In the future, further exploration into how to more accurately assess the effectiveness of instruction data without instruction tuning would be valuable.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. *CoRR*, abs/2108.07732.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, Huazuo Gao, Kaige Gao, Wenjun Gao, Ruiqi Ge, Kang Guan, Daya Guo, Jianzhong Guo, Guangbo Hao, Zhewen Hao, Ying He, Wenjie Hu, Panpan Huang, Erhang Li, Guowei Li, Jiashi Li, Yao Li, Y. K. Li, Wenfeng Liang, Fangyun Lin, Alex X. Liu, Bo Liu, Wen Liu, Xiaodong Liu, Xin Liu, Yiyuan Liu, Haoyu Lu, Shanghao Lu, Fuli Luo, Shirong Ma, Xiaotao Nie, Tian Pei, Yishi Piao, Junjie Qiu, Hui Qu, Tongzheng Ren, Zehui Ren, Chong Ruan, Zhangli Sha, Zhihong Shao, Junxiao Song, Xuecheng Su, Jingxiang Sun, Yaofeng Sun, Minghui Tang, Bingxuan Wang, Peiyi Wang, Shiyu Wang, Yaohui Wang, Yongji Wang, Tong Wu, Y. Wu, Xin Xie, Zhenda Xie, Ziwei Xie, Yiliang Xiong, Hanwei Xu, R. X. Xu, Yanhong Xu, Dejian Yang, Yuxiang You, Shuiping Yu, Xingkai Yu, B. Zhang, Haowei Zhang, Lecong Zhang, Liyue Zhang, Mingchuan Zhang, Minghua Zhang, Wentao Zhang, Yichao Zhang, Chenggang Zhao, Yao Zhao, Shangyan Zhou, Shunfeng Zhou, Qihao Zhu, and Yuheng Zou. 2024. Deepseek LLM: scaling open-source language models with longtermism. *CoRR*, abs/2401.02954.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Xiaomeng Zhao, and et al. 2024. Internlm2 technical report. *CoRR*, abs/2403.17297.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

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. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality.

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. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

OpenCompass Contributors. 2023. Opencompass: A universal evaluation platform for foundation

models. https://github.com/open-compass/opencompass.

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

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The llama 3 herd of models. *CoRR*, abs/2407.21783.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacafarm: A simulation framework for methods that learn from human feedback. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

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. C-eval: A multi-level multi-discipline chinese evaluation suite for foundation models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *CoRR*, abs/2310.06825.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 4667–4688. Association for Computational Linguistics.

Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. 2023. Openassistant conversations - democratizing large language model alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and

Jing Xiao. 2024. From quantity to quality: Boosting LLM performance with self-guided data selection for instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 7602–7635. Association for Computational Linguistics.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024. What makes good data for alignment? A comprehensive study of automatic data selection in instruction tuning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2024. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Long Ouyang, Jeffrey 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 F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2020. Mpnet: Masked and permuted pre-training for language understanding. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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.

Qwen Team. 2024. Qwen2.5: A party of foundation models.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024a. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024b. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *CoRR*, abs/2406.08464.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Bill Yuchen Lin, and Radha Poovendran. 2024c. Stronger models are not stronger teachers for instruction tuning. *Preprint*, arXiv:2411.07133.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *CoRR*, abs/2407.10671.

Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. 2024. Automatic instruction evolving for large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 6998–7018. Association for Computational Linguistics.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, and Guoyin Wang. 2023. Instruction tuning for large language models: A survey. *CoRR*, abs/2308.10792.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. 2024. Wildchat: 1m chatgpt interaction logs in the wild. In *The Twelfth*

International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric P. Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. 2024a. Lmsys-chat-1m: A large-scale real-world LLM conversation dataset. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, and Yongqiang Ma. 2024b. Llamafactory: Unified efficient fine-tuning of 100+ language models. *CoRR*, abs/2403.13372.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. 2023a. LIMA: less is more for alignment. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023b. Instruction-following evaluation for large language models. *CoRR*, abs/2311.07911.

A Appendix

A.1 Experimental Details

Evolution Details of Evol-Instruct. As shown in Figure 10, 11 and 12, the instruction evolution prompts we utilized are derived from (Xu et al., 2024a; Luo et al., 2024), with minor modifications. For the Alpaca dataset, we employ four in-depth evolution methods: *deepening*, *concretizing*, *adding constraints*, and *adding reasoning steps*, in addition to one breadth-focused evolution method. However, for the GSM8K Train and Code Alpaca datasets, we exclude the breadth-focused method and use only the four in-depth methods. To ensure a fair comparison, we apply these evolution methods to each original instruction in a fixed sequence, rather than randomly selecting them as in the original Evol-Instruct. This strategy is designed to eliminate variations in the evolution of the same instruction, thereby reducing the potential for biased experimental conclusions. The results in Table1 and Table 2 are obtained after one round of evolution of the seed instructions.

Evolution Details of AutoIF. Following the approach in AutoIF, we typically employ Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct to carry out the Instruction Augmentation and Verification steps, generating 780 and 420 instructions, respectively. Due to the multiple verification steps required by AutoIF for filtering, the number of generated instructions varies. To ensure fairness, we randomly select 420 instructions from the 780 generated by the SLMs for comparison. These instructions are then concatenated with queries from ShareGPT to create a dataset of 6,720 instruction data for subsequent training.

Evolution Details of Auto Evol-Instruct. We compare the performance of Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct in automatically designing evolutionary trajectories for evolving instructions. Using the prompt template from Auto Evol-Instruct (Zeng et al., 2024) (refer to Figure 15), we prompt the models to design evolutionary trajectories and evolve instructions autonomously. To avoid introducing additional bias, we exclude the optimization stage from Auto Evol-Instruct. The experimental setup and evaluation benchmarks are consistent with those in Section 3.1. Since the models occasionally fail to adhere to the specified output format, leading to instruction extraction errors, we perform random sampling on the larger sets of evolved instructions from both models to ensure consistent quantities of instruction data. We also use Qwen-2.5-72B-Instruct to generate the responses. Finally, for the Alpaca, GSM8K, and Code Alpaca datasets, we conduct automatic evolution and sampling, resulting in 40,483, 6,200, and 15,533 instruction data points, respectively.

Implementation Details For a fair comparison, all of our experiments maintain consistent data volumes. During the construction of the instruction data, we leverage the vLLM framework (Kwon et al., 2023) for acceleration using a temperature of 0.7 and a top_p value of 0.95. For training the models, we utilize the LLaMA-Factory framework (Zheng et al., 2024b) with a global batch size of 64, a cutoff length of 2048, and a learning rate of 2e-5, following a cosine learning rate schedule over 3 epochs. No checkpoint selection is performed; instead, all models are evaluated using the final saved checkpoint. All experiments are carried out on 8 $\times$ NVIDIA Tesla A100 GPUs.

Base Models. In the Evol-Instruct scenario, we fine-tune Llama series models (Dubey et al., 2024) including Llama-3.2-3B, Llama-3.1-8B, Llama-3-8B, DeepSeek-7B (Bi et al., 2024), Mistral-7B-

v0.3 (Jiang et al., 2023), and InternLM-2-7B (Cai et al., 2024) models. In the AutoIF scenario, we also use Llama series models as in Evol-Instruct, as well as Qwen series (Yang et al., 2024) models including Qwen-2.5-7B and Qwen-2-7B, along with InternLM-2-7B. For Auto Evol-Instruct, we evaluate the performance of the Llama series models.

Hyperparameter	Value
Learning Rate	2×10^{-5}
Number of Epochs	3
Number of Devices	8
Per-device Batch Size	1
Gradient Accumulation Steps	8
Learning Rate Scheduler	cosine
Warmup Ratio	0.03
Max Sequence Length	2048

Table 6: Hyperparameters utilized in Evol-Instruct, AutoIF and Auto Evol-Instruct scenarios.

More Hyperparameter Details. We provide the detailed hyperparameters for supervised fine-tuning in Table 6. Except for IFEval and FollowBench, which are evaluated using their respective repositories, all other evaluations are conducted using the OpenCompass (Contributors, 2023) framework, and vLLM is adopted for inference acceleration throughout the evaluation process to enhance computational efficiency and expedite the assessment procedures.

A.2 Detailed Information of Seed Datasets

In Evol-Instruction and Auto Evol-Instruct scenarios, we utilize the following seed datasets for instruction following, mathematical reasoning, and code generation: (1) Alpaca, a dataset that contains about 52K instruction following data points, (2) GSM8K Train, a dataset that includes nearly 7K high-quality, linguistically diverse grade school math word problems; and (3) Code Alpaca, a code generation dataset comprising approximately 20K samples. Table 8 presents the statistical information of the seed datasets.

In the AutoIF scenario, we follow the setup described in the AutoIF paper, using the seed instructions provided by the authors and the queries from ShareGPT to construct the instructions.

Hyperparameter	Value
General Hyperparameters	
Number of Epochs	2
Number of Devices	8
Per-device Batch Size	1
Gradient Accumulation Steps	8
Learning Rate Scheduler	cosine
Warmup Ratio	0.03
Max Sequence Length	2048
LoRA Hyperparameters	
LoRA Rank	8
LoRA Alpha	8
LoRA Target	all module
LoRA Dropout	0.0
Qwen-2.5-0.5B and 1.5B	
Learning Rate	1×10^{-5}
Qwen-2.5-3B and 7B	
Learning Rate	7×10^{-6}
Qwen-2.5-14B, 32B and 72B	
Learning Rate	5×10^{-5}

Table 7: Hyperparameters utilized for fine-tuning Qwen-2.5 series models.

A.3 Detailed Information of Evaluations

To evaluate the instruction following capabilities of our models, we employ several benchmarks, including IFEval and FollowBench. IFEval consists of 25 types of verifiable instructions across approximately 500 prompts, while FollowBench is a fine-grained, constraint-based instruction following a benchmark with five difficulty levels. It includes diverse open-ended instructions that require evaluation by strong LLMs. We report both strict and loose accuracy metrics at the prompt and instruction levels, and for FollowBench, we specifically report the Hard Satisfaction Rate (HSR).

In addition to instruction following benchmarks, we assess the models on other tasks. For mathematical reasoning, we use GSM8K and MATH. GSM8K consists of grade school math problems, while MATH presents more challenging mathematical problems. We report accuracy scores for both datasets. For code generation, we evaluate the models using HumanEval and MBPP, reporting the pass@1 metrics. We also evaluate our models on C-Eval, and MMLU to provide a comprehen-

	Seed Data	
	Dataset	**Datasize**
Instruction Following	Alpaca	51,983
Mathematical Reasoning	GSM8K Train	7,473
Code Generation	Code Alpaca	20,022

Table 8: Statistics of seed instruction data used in the Evol-Instruct and Auto-Evol-Instruct scenarios.

sive assessment of the models' capabilities across various domains.

A.4 More Experimental Results

Seed Instruction Data Results. Table 9 presents the experimental results for the seed instruction datasets used in Evol-Instruct and Auto Evol-Instruct scenarios. We observe that the performance of models trained on these seed data is sub-optimal. We argue that the quality of these seed data is no longer adequate to further improve the performance of the current advanced base models.

Detailed Results of Multi-Iteration Evolution. Table 10 presents the detailed results of different evolved iterations which are referred to Figure 1.

Detailed Results of Scaling Experiments. Table 11 presents the detailed results of the model scaling experiment shown in Figure 3.

The Impact of Temperatures. To explore the impact of temperature on the evolutionary instruction data, we compare Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct under different temperatures. Specifically, we evolve the Code Alpaca data under greedy decoding (with a temperature of 0) and at five different temperatures ranging from 0.1 to 0.9, and uniformly use Qwen-2.5-72B-Instruct to generate the corresponding responses. As shown in Table 12, the results of training on Llama-3.2-3B indicate that the SLMs perform consistently better than LLMs under all temperatures, which further validates the universality of our conclusion.

More Results of IC-IFD. To further validate the broad applicability of IC-IFD, beyond high-difficulty instruction data, we use the IC-IFD and IFD metrics to filter 5%, 10%, and 15% of the original Alpaca dataset for training the Llama-3-8B and Llama-3.2-3B models. We fine-tune the models on the IC-IFD and IFD-filtered data and evaluate their performance using instructions from AlpacaFarm (Dubois et al., 2023). The generated

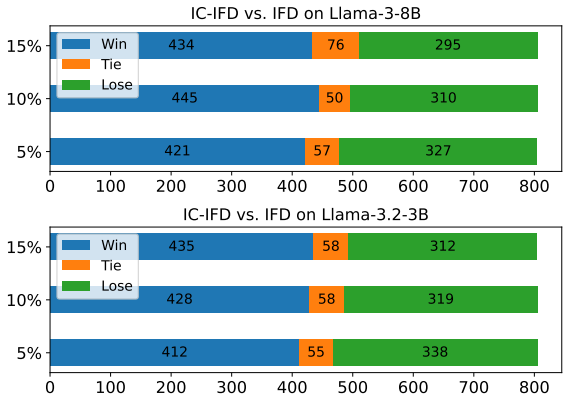

Figure 6: Performance comparison of three data selection ratios on Alpaca dataset between IC-IFD and IFD.

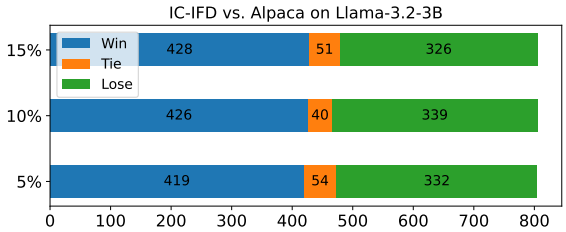

Figure 7: Performance comparison of three data selection ratios on Alpaca dataset between IC-IFD and full dataset.

responses are then assessed using GPT-4 to determine the win-tie-lose ratio (the evaluation prompt template can be found in Figure 20). As shown in Figure 6, we observe that IC-IFD consistently outperforms IFD across all three data ratio settings for both models. Furthermore, we compare the performance of models trained on IC-IFD-filtered data with those trained on the full Alpaca dataset. As shown in Figure 7, models trained on IC-IFD-filtered data also perform better than those trained on the full dataset, further demonstrating the effectiveness of the proposed IC-IFD.

Case Study. We compare the evolution of SLMs and LLMs across two specific in-depth cases. As illustrated in Figure 8, we observe that in the "adding constraints" evolution trajectory, the evolved instructions of SLMs incorporate two additional constraints: lack of time for exercise and inability to limit diet, while the evolved instructions of LLMs only add the condition that the requirements must be feasible. Similarly, in the "deepening" evolution trajectory, as shown in Figure 9, the evolved instructions of SLMs are significantly more challenging, containing numerous in-depth conditions,

Model	Instruction Following (IFEval)				Math Reasoning		Code Generation	
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	GSM8K	MATH	HumanEval	MBPP
Seed instruction data								
Mistral-7B-v0.3	17.01	26.86	19.04	29.14	27.07	0.12	10.20	8.80
DeepSeek-7B	22.00	34.05	23.48	35.73	44.05	0.56	25.61	33.80
Llama-3.2-3B	22.55	34.17	25.88	37.65	46.40	0.56	28.05	32.20
Llama-3-8B	23.11	32.97	24.77	35.13	53.68	0.22	25.00	28.60
Llama-3.1-8B	27.54	38.13	28.65	39.21	56.41	7.56	29.88	31.80
InternLM-2-7B	32.72	45.08	35.30	48.08	61.87	10.28	42.07	40.00

Table 9: Results of seed instruction data.

Model	Instruction Following (IFEval)				Math Reasoning		Code Generation	
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	GSM8K	MATH	HumanEval	MBPP
Supervised Model: Llama-3.1-70B-Instruct								
Iteration 1	33.83	46.28	36.41	49.28	63.00	7.62	43.90	36.20
Iteration 2	32.53	43.76	34.20	46.16	64.59	10.04	42.07	36.60
Iteration 3	35.12	47.36	36.97	49.28	64.75	11.82	43.29	37.20
Supervised Model: Llama-3.1-8B-Instruct								
Iteration 1	35.49	47.00	39.56	50.72	63.38	11.44	48.17	37.60
Iteration 2	36.78	48.20	40.30	50.84	64.82	11.48	48.78	39.40
Iteration 3	33.09	44.72	36.41	48.32	65.88	14.12	44.51	40.80

Table 10: Detailed performance of different evolved iterations on Llama-3-8B refer to Figure 1.

which is absent in the evolved instructions of LLMs. Overall, from actual cases, SLMs can evolve more complex and diverse instructions under the same constraints or trajectories, achieving more effective instructions at a lower computational cost.

A.5 Further Analysis

Difficulty Scores of Evol-Instruct. We utilize the prompt template shown in Figure 19 to prompt Qwen-2.5-72B-Instruct for evaluating the complexity scores of the three-round data in the Evol-Instruct scenario. As shown in Table 13, we find that in each round, SLM-INST consistently outperforms LLM-INST in terms of complexity scores. Interestingly, SLM-INST Iter 2 is even more difficult than LLM-INST Iter 3, as demonstrated by the experiment in Figure 1, where the overall performance of SLM-INST Iter 2 is superior to that of LLM-INST Iter 3.

Quality Score Evaluated by Reward Model. We also utilize InternLM-2-7B-Reward as the reward model to evaluate the average scores of the evolved instructions of both SLMs and LLMs. Specifically, given the evolved prompt templates (as shown in Figure 10 and 12), we then use the reward model to evaluate the rewards of the evolved instructions generated by SLMs and LLMs respectively and obtain the mean reward of the instruction set. As shown in Table 14, we find that the overall scores of the instructions evaluated by the reward model are approximately in line with its performance during the training stage. However, on some datasets, it could not accurately reflect the quality of the instructions. Moreover, using the reward model cannot directly assess the quality of the instructions. Instead, it requires the meta-instructions used when constructing the instructions. Therefore, the reward model cannot be well applied to the evaluation of instructions.

Comparison of IFD and IC-IFD. We analyze the third-round evolved Alpaca dataset for both SLMs and LLMs. Specifically, we compute the IFD and IC-IFD scores for each sample in both datasets and compare their average scores. As shown in Table 15, we evaluate the average performance of IFEval on the two datasets using Llama-3-8B. We find that when the instruction difficulty level is too high, the IFD score tends to increase. However, the performance of the fine-tuned models does not align with expectations. In contrast, the IC-IFD score effectively captures the influence of instruction complexity, offering a more accurate data quality assessment.

A.6 Prompt Templates

Prompt Templates of Evol-Instruct. Figure 10 shows the in-depth evolution prompt template for instruction evolution used in the Evol-Instruct sce-

Model	Instruction Following (IFEval)				Math Reasoning		Code Generation	
	Pr.(S)	In.(S)	Pr.(L)	In.(L)	GSM8K	MATH	HumanEval	MBPP
Supervised Model: Llama-3.1-70B-Instruct								
Qwen-2.5-0.5B	18.48	32.73	22.00	35.85	40.26	16.32	30.49	27.60
Qwen-2.5-1.5B	28.84	42.67	31.98	46.04	62.32	24.06	50.00	43.20
Qwen-2.5-3B	37.89	48.56	42.70	53.60	76.12	26.44	63.41	55.40
Qwen-2.5-7B	46.21	56.83	50.64	60.79	76.12	38.14	70.73	61.60
Qwen-2.5-14B (LoRA)	40.11	54.43	48.24	61.99	87.79	49.94	75.00	67.20
Qwen-2.5-32B (LoRA)	42.88	57.31	51.20	64.15	87.79	55.02	80.49	71.20
Qwen-2.5-72B (LoRA)	50.63	68.43	57.12	70.98	91.05	58.83	82.93	76.00
Supervised Model: Llama-3.1-8B-Instruct								
Qwen-2.5-0.5B	17.38	29.38	19.78	32.01	40.71	16.26	34.76	28.00
Qwen-2.5-1.5B	28.47	41.73	31.98	44.96	65.35	27.84	52.44	49.94
Qwen-2.5-3B	38.82	49.76	42.51	53.96	76.57	30.92	64.02	55.80
Qwen-2.5-7B	47.32	58.39	51.39	62.35	82.03	43.78	71.95	61.80
Qwen-2.5-14B (LoRA)	42.51	55.16	51.02	62.47	88.17	52.22	75.61	67.20
Qwen-2.5-32B (LoRA)	45.84	58.75	54.71	66.31	89.61	55.28	81.71	73.20
Qwen-2.5-72B (LoRA)	52.79	72.56	61.25	73.27	91.36	60.75	84.67	76.80

Table 11: Detailed performance among Qwen-2.5 series models refer to Figure 3.

Temperature	HumanEval	MBPP	HumanEval	MBPP
	Supervised Model: Llama-3.1-70B-Instruct		*Supervised Model: Llama-3.1-8B-Instruct*	
greedy	37.20	33.40	**39.63**	**36.40**
0.1	36.59	36.40	**37.80**	**37.60**
0.3	38.41	35.20	**39.63**	**37.80**
0.5	35.98	33.40	**37.80**	**35.80**
0.7	35.98	**36.00**	**39.02**	32.80
0.9	34.76	33.00	**40.24**	**35.80**

Table 12: Performance among different temperatures on Llama-3.2-3B under code generation scenario.

	Alpaca	GSM8K Train	Code Alpaca
Seed Instruction	27.63	34.05	26.01
LLM-INST Iter1	52.89	39.88	46.75
SLM-INST Iter1	**66.35**	**48.85**	**58.86**
LLM-INST Iter2	68.16	47.14	65.02
SLM-INST Iter2	**77.62**	**63.48**	**73.37**
LLM-INST Iter3	75.73	54.00	72.85
SLM-INST Iter3	**82.44**	**72.12**	**79.19**

Table 13: Scores of difficulty levels for instructions evolved during three iterations, using Llama-3.1-8B-Instruct and Llama-3.1-70B-Instruct as supervised models for each round under Evol-Instruct scenario.

Iteration	Average Reward		
	Alpaca	GSM8K	Code Alpaca
Supervised Model: Llama-3.1-70B-Instruct			
Iteration 1	1.54	0.74	1.10
Iteration 2	**1.68**	0.73	**1.19**
Iteration 3	**1.56**	0.69	**1.14**
Supervised Model: Llama-3.1-8B-Instruct			
Iteration 1	**1.59**	**1.01**	**1.23**
Iteration 2	1.54	**0.79**	0.96
Iteration 3	1.42	**0.97**	1.03

Table 14: Comparison of average rewards among different iteration evolution instruction data.

nario, derived from (Xu et al., 2024a) and slightly modified. Figures 11 and 12 demonstrate the four in-depth methods and one in-breadth evolved prompt template we adopt.

Prompt Templates of AutoIF. We utilize the prompt templates consistent with those in (Dong et al., 2024). Figures 13 and 14 represent the prompts used in the two stages: Self-Instruct Seed Instructions and Verification Funcs and Cases Generation.

Prompt Templates of Auto Evol-Instruct. As shown in Figure 15, we utilize the prompt templates consistent with those in (Zeng et al., 2024) under Auto Evol-Instruct scenario.

Prompt Templates of Response Generation. We use the prompt template shown in Figure 16 to generate the corresponding responses for all instructions. We adopt the data organization format

Case Study (Adding Constraints)

Original Instruction:
Give three tips for staying healthy.

LLMs Evolution Instruction:
Provide three actionable tips for staying healthy, considering a moderate lifestyle, and explain how they can be incorporated into a daily routine.

SLMs Evolution Instruction:
Consider the following additional constraint:
Assuming a person has a busy schedule with limited time for exercise and a restrictive diet, provide three evidence-based tips for maintaining overall health and well-being.

Figure 8: Comparison of cases between LLMs and SLMs under adding constraints strategy.

Case Study (Deepening)

Original Instruction:
Weng earns $12 an hour for babysitting. Yesterday, she just did 50 minutes of babysitting. How much did she earn?

LLMs Evolution Instruction:
Weng earns $12 an hour for babysitting, but her rate is prorated to the nearest 10-minute interval. Yesterday, she did 50 minutes of babysitting. How much did she earn, and would her earnings change if she had worked for the full hour?

SLMs Evolution Instruction:
Consider a scenario where Weng earns a variable hourly wage for babysitting, which is $12 plus a 10% bonus for every hour worked on a weekday and a 15% bonus for every hour worked on a weekend. Yesterday, she worked 50 minutes of babysitting, which falls on a Saturday. Additionally, she has a 5% additional ones on her total earnings if she completes the babysitting job within the scheduled time. How much did she earn, considering the bonuses and additional earns?

Figure 9: Comparison of cases between LLMs and SLMs under deepening strategy.

Datasets	IFD (%)	IC-IFD (%)	Performance
SLMs (Alpaca iter 3)	**83.04**	35.89	40.64
LLMs (Alpaca iter 3)	82.03	**37.05**	**42.18**

Table 15: Comparison of IFD and IC-IFD on third-round evolved Alpaca datasets from SLMs and LLMs.

from Llama-Factory, and therefore, when generating responses, we classify them into two types based on the presence of an input.

Prompt Templates of Data Analysis. Figure 17 and 19 show the prompt templates used to assess the difficulty levels and scores of instructions. Figure 18 displays the prompt template used to analyze the evolutionary trajectories automatically generated by the model.

Prompt Templates of Evaluation. Figure 20 shows the prompt template used to assess the win-tie-lose rates on AlpacaFarm.

In-Depth Evolution Prompt

I want you to act as a Prompt Rewriter.
Your objective is to rewrite a given prompt into a more complex version to make those famous AI systems (e.g., ChatGPT and GPT-4) a bit harder to handle.
But the rewritten prompt must be reasonable and must be understood and responded to by humans.
You SHOULD complicate the given prompt using the following method:
{METHOD}
You should try your best not to make the #Rewritten Prompt# become verbose, #Rewritten Prompt# can only add 10 to 20 words into #The Given Prompt#.
You MUST only generate the new prompt without #The Given Prompt# and #Rewritten Prompt#.
#The Given Prompt#:
{INSTRUCTION}
#Rewritten Prompt#:

Figure 10: In-depth evolution prompt template utilized in Evol-Instruct scenario.

In-depth Evolution Method

Adding Constraints:
Please add one more constraint/requirement into #The Given Prompt#
Deepening:
If #The Given Prompt# contains inquiries about certain issues, the depth and breadth of the inquiry can be increased.
Concretizing:
Please replace general concepts with more specific concepts.
Adding Reasoning Steps:
If #The Given Prompt# can be solved with just a few simple thinking processes, you can rewrite it to explicitly request multiple-step reasoning.

Figure 11: Four in-depth methods utilized in Evol-Instruct scenario.

In-breadth Evolution Prompt

I want you to act as a Prompt Creator.
Your goal is to draw inspiration from the #Given Prompt# to create a brand new prompt.
This new prompt should belong to the same domain as the #Given Prompt# but be even more rare.
The LENGTH and complexity of the #Created Prompt# should be similar to that of the #Given Prompt#.
The #Created Prompt# must be reasonable and must be understood and responded to by humans or modern AI chatbots.
You MUST only generate the new prompt without any other words or special symbols.
#Given Prompt#:
{INSTRUCTION}
#Created Prompt#:

Figure 12: In-breadth evolution prompt template utilized in Evol-Instruct scenario.

Self-Instruct Seed Instructions Prompt

You are an expert for writing instructions. Please provide 50 different instructions that meet the following requirements:
- Instructions are about the format but not style of a response
- Whether instructions are followed can be easily evaluated by a Python function
Here are some examples of seed instructions we need:
{SEED_INSTRUCTIONS}
Do not generate instructions about writing style, using metaphor, or translation. Here are some examples of instructions we do not need:
- Incorporate a famous historical quote seamlessly into your answer
- Translate your answer into Pig Latin
- Use only words that are also a type of food
- Respond with a metaphor in every sentence
- Write the response as if you are a character from a Shakespearean play
Please generate one instruction per line in your response and start each line with '- '.
Do NOT repeat the seed instructions.

Figure 13: Prompt template of Self-Instruct Seed Instructions in AutoIF scenario.

Verification Funcs and Cases Generation

You are an expert for writing evaluation functions in Python to evaluate whether a response strictly follows an instruction.
Here is the instruction: {INSTRUCTION}
Please write a Python function named `evaluate` to evaluate whether an input string `response` follows this instruction. If it follows, simply return True, otherwise return False.
Please respond with a single JSON that includes the evaluation function in the key `func`, and a list of three test cases in the key `cases`, which includes an input in the key `input` and an expected output in the key `output` in (true, false).
Here is an example of output JSON format: {{"func": "JSON_STR(use only \\n instead of \n)", "cases": [{{"input": "str", "output": "str"}}]}}.

Figure 14: Prompt template of Verification Funcs and Cases Generation in AutoIF scenario.

Auto Evol-Instruct Prompt

You are an Instruction Rewriter that rewrites the given #Instruction# into a more complex version. Please follow the steps below to rewrite the given #Instruction# into a more complex version.
Step 1: Please read the #Instruction# carefully and list all the possible methods to make this instruction more complex (to make it a bit harder for well-known AI assistants such as ChatGPT and GPT4 to handle). Please do not provide methods to change the language of the instruction!
Step 2: Please create a comprehensive plan based on the #Methods List# generated in Step 1 to make the #Instruction# more complex. The plan should include several methods from the #Methods List#.
Step 3: Please execute the plan step by step and provide the #Rewritten Instruction#. #Rewritten Instruction# can only add 10 to 20 words into the #Instruction#.
Step 4: Please carefully review the #Rewritten Instruction# and identify any unreasonable parts. Ensure that the #Rewritten Instruction# is only a more complex version of the #Instruction#. Just provide the #Final Rewritten Instruction# without any explanation.
Please reply strictly in the following format:
Step 1 #Methods List#:
Step 2 #Plan#:
Step 3 #Rewritten Instruction#:
Step 4 #Finally Rewritten Instruction#:
#Instruction#: {INSTRUCTION}

Figure 15: Prompt template of Auto Evol-Instruct scenario.

Response Generation Prompt

When input is provided:
Given the following instruction and input, please provide a comprehensive and accurate response.
Instruction: {INSTRUCTION}
Input: {INPUT}
Response:

When no input is provided:
Given the following instruction, please provide a comprehensive and accurate response.
Instruction: {INSTRUCTION}
Response:

Figure 16: Prompt template of response generation.

Evaluate Difficulty Level

Instruction
You first need to identify the given user intent and then label the difficulty level of the user query based on the content of the user query.

User Query
{QUERY}

Output Format
Given the user query, in your output, you first need to identify the user intent and the knowledge needed to solve the task in the user query. Then, rate the difficulty level of the user query as 'very easy', 'easy', 'medium', 'hard', or 'very hard'.

Remember, only generate the difficulty level without any other words or symbols.
Output

Figure 17: Prompt template of evaluating the difficulty levels.

Extract Keywords of Trajectory

You are tasked with generating a concise summary for the given trajectory of instruction evolution. Please follow the steps below:
Step 1: Carefully read the given trajectories of instruction evolution and identify the key concept or process it describes.
Step 2: Create a short and simple phrase that accurately summarizes the core idea of the trajectory. The summary should be succinct and focus on the essence of the evolution process. Ensure that the phrase does not contain any unnecessary symbols, punctuation, or formatting. It should be just a brief, clear description of the method. Please ignore the numerical labels or special identifiers at the beginning of the methods.
Provide only the summary phrase without any further explanation or additional information.
Trajectory of Instruction Evolution: {TRAJECTORY}

Figure 18: Prompt template of extracting the keywords from evolution trajectories.

Evaluate Difficulty Score

Instruction
You first need to identify the given user intent and then label the difficulty score of the user query based on the content of the user query.

User Query
{QUERY}

Output Format
Given the user query, in your output, you first need to identify the user intent and the knowledge needed to solve the task in the user query. Then, rate the difficulty score of the user query from 0 to 100.

Remember, only generate the difficulty score without any other words or symbols.
Output

Figure 19: Prompt template of evaluating the difficulty scores.

Evaluation Prompt

[User]
{USER_QUERY}
[Assistant 1]
{ASSISTANT 1 QUERY}
[Assistant 2]
{ASSISTANT 2 QUERY}
[System Information]
We would like to request your feedback on the two dialogues shown above between two AI assistants. Focus on the AI's responses. The AI's responses should perfectly align with the user's needs. Additionally, the responses should be concise and to the point, avoiding unnecessary details or excessive information, while still being as comprehensive as possible in addressing the user's query. The answers must maintain good logical flow, use precise technical terms, and be factually accurate and objective.
Based on the above criteria, compare the performance of Assistant 1 and Assistant 2. Determine which one is "better than," "worse than," or "equal to" the other. First, compare their responses and analyze which aligns better with the stated requirements.
On the last line, output a single label only, selecting from one of the following:
'Assistant 1 is better than Assistant 2'
'Assistant 1 is worse than Assistant 2'
'Assistant 1 is equal to Assistant 2'

Figure 20: Prompt template of evaluating the win-tie-lose rates.