DecIF: Improving Instruction-Following through Decomposition

Tingfeng Hui[1], Pengyu Zhu[1], Bowen Ping[2], Ling Tang[1], Guanting Dong[3], Yaqi Zhang[1], Sen Su[1]*

[1]Beijing University of Posts and Telecommunications, Beijing, China
[2]Peking University, Beijing, China
[3]Renmin University of China, Beijing, China
[1](huitingfeng,susen)@bupt.edu.cn

Abstract

We propose a novel data synthesis framework, DecIF, which automatically generates accurate and diverse instruction-following data from scratch for supervised fine-tuning (SFT) and reinforcement learning (RL), leveraging large language models (LLMs) and minimal external resources. By decomposing the data synthesis pipeline into fine-grained steps, DecIF achieves meticulous quality and diversity control over generated instruction-following data. Extensive experiments across both SFT and RL demonstrate DecIF's strong capability to flexibly synthesize accurate instruction-following data for both paradigms compared to comprehensive baselines. Further analysis demonstrates the framework's robustness, scalability, and computational efficiency in instruction-following data generation, while its modular design ensures straightforward implementation and reproducibility. The source code are available at: https://github.com/HypherX/DecIF

1 Introduction

Large language models (LLMs) have demonstrated strong performance across diverse NLP tasks and are increasingly integrated into daily life through AI assistants (OpenAI, 2024; Yang et al., 2025a). As their applications expand, instruction-following, the ability to accurately adhere to complex constraints, has become one of the key research focus (Li et al., 2024b; Dong et al., 2024a,b; Chen et al., 2025). Recent efforts to align LLMs with instruction-following tasks have mainly focused on how to obtain high-quality instruction-following data. For example, AutoIF (Dong et al., 2024a) integrates manually designed constraints into instructions and uses Python code to ensure consistency between inputs and outputs. AIR (Liu et al., 2025) extracts instructions from documents and iteratively refines them to generate large-scale data. UltraIF (An et al., 2025) synthesizes instruction data using a fine-tuned UltraComposer and employs LLM-as-a-judge for response verification. VerIF (Peng et al., 2025) utilize Constraint Back-translation (Qi et al., 2025) to obtain constraints and incorporate rule-based and LLM-based rewards for RL. However, previous approach rely heavily on existing documents and resources (e.g. large datasets) for obtaining queries or constraints, limiting their flexibility.

In this paper, we propose a novel framework, DecIF, which automatically generates accurate and diverse instruction-following data from scratch with verifiable rewards, using only LLMs and minimal external resources. DecIF's pipeline consists of two primary phases: instruction construction and response construction. Specifically,

(1) **Instruction Construction**: To improve the controllability and reliability of instruction synthesis under complex constraints, we propose a modular instruction construction pipeline that avoids the instability of direct end-to-end generation (see Figure 1). Our core motivation is to ensure that constraints and their verification logic are tightly coupled and that the final instructions are both diverse and verifiable. We decompose the pipeline into two stages: *query generation* and *constraint formulation*. In the query generation stage, we progressively generate four types of meta-information: *domain*, *request*, *scenario*, and *persona* which are combined to produce rich, diverse, and difficulty-controlled queries. Specifically, domains define topical boundaries, requests specify the core intent, scenarios add contextual variety, and personas modulate the complexity level of the query. In the constraint formulation stage, we distinguish between *soft constraints*, which encode stylistic or qualitative requirements, and *hard constraints*, which define strict, rule-based conditions. For soft constraints, we first prompt LLMs to produce high-level constraint categories, then jointly generate

*The corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 848–867

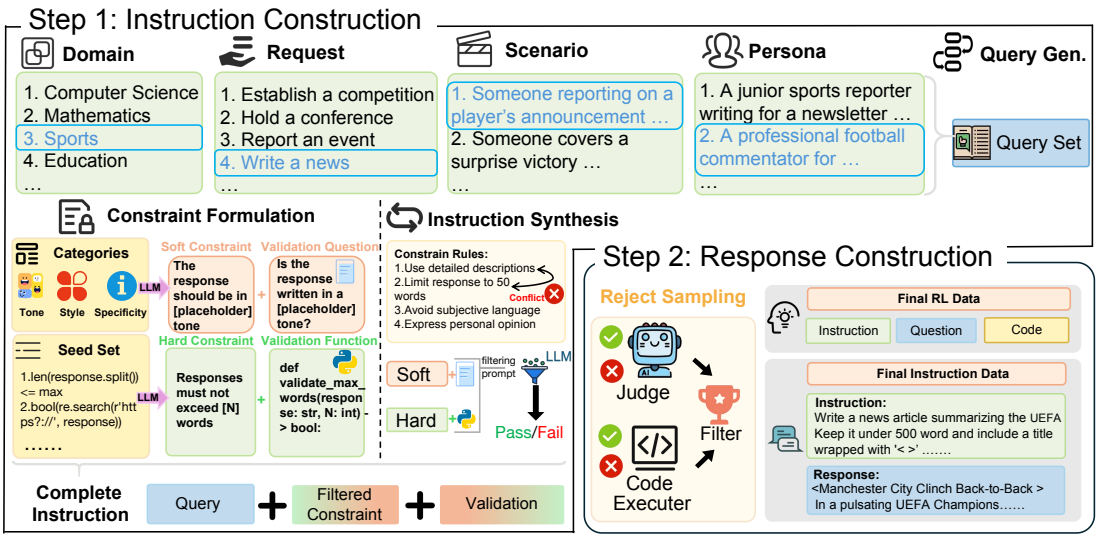

Figure 1: The overall workflow of DecIF. DecIF is decomposed into two main stages: **Instruction Construction** and **Response Construction**. The instruction construction is further decomposed into *query generation* and *constraint formulation*, while the response construction is divided into *response generation* and *reject sampling*.

template-style constraints and their corresponding validation questions. For hard constraints, we start from a small set of template constraints paired with verification functions, and expand them via Self-Instruct (Wang et al., 2023a) process that co-generates new constraint-function pairs. To ensure compatibility, we do not assess constraint combinations based on surface content alone. Instead, we leverage LLMs to perform conflict filtering guided by their associated validation questions and functions, allowing the model to reason from a verification perspective and capture subtle conflicts that would be difficult to detect otherwise. Finally, the filtered constraint combinations are merged with queries, and LLMs are prompted to instantiate all components into the final fully specified, verifiable, and natural-language instructions.

(2) **Response Construction**: To ensure the quality and verifiability of model outputs, we decompose the response construction pipeline into two stages: *response generation* and *reject sampling*. In the response generation stage, we sample multiple candidate responses for each instruction. In the reject sampling stage, we apply both natural-language validation questions and executable verification functions to assess the correctness of each response. Only instruction-response pairs that pass the corresponding validation checks are retained as high-quality SFT data. For RL data construction, we rely directly on the instruction-verification pairs. However, we first perform a filtering step to remove any examples where the functions produce runtime errors, timeouts, or where the questions are incomplete. This ensures that the reward signals used in reinforcement learning are both accurate and stable.

Extensive experiments across diverse settings and comprehensive benchmarks demonstrate that DecIF substantially enhances the instruction-following capabilities of LLMs compared to prior methods. This improvement is attributed to the increased accuracy, diversity, and verifiability of the synthesized data. Beyond basic instruction-following tasks, we evaluate DecIF across a broad spectrum of model capabilities, confirming the generalizability of our approach. To assess practical applicability, we replace the instruction-following subset in Tulu-3 (Lambert et al., 2025) with data generated by DecIF. The resulting model not only surpasses Tulu-3 in instruction-following performance but also remains competitive across other general capabilities—highlighting the compatibility of our data with diverse training pipelines. We further explore the utility of DecIF in generating general-purpose instruction data and analyze the effect of reinforcement learning configurations on instruction-following outcomes. **Our main contributions are summarized as follows:**

(1) We propose DecIF, a decomposition-guided framework for synthesizing accurate, diverse, and verifiable instruction-following data for both SFT and RL, using only LLMs and minimal external

resources.

(2) DecIF decomposes the entire synthesis pipeline into fine-grained sub-tasks, enabling precise control over diversity, complexity, and contextual richness, supporting systematic and controllable data generation.

(3) Extensive experiments demonstrate that DecIF outperforms existing method in handling complex instructions and shows strong potential for generating general-purpose data compatible with modern open-source training regimes.

2 Methodology

Previous efforts to construct instruction-following data have typically relied on real-world resources (Dong et al., 2024a; An et al., 2025; Qi et al., 2025; Peng et al., 2025; Zhang et al., 2026a), such as obtaining queries from ShareGPT (Chiang et al., 2023) or leveraging open-source datasets to extract diverse constraints. In contrast, we aim to explore a fully from-scratch approach for constructing instruction-following data, enhancing the flexibility and adaptability of the method. Detailed step-by-step cases of our data construction process are provided in Appendix B.

2.1 Instruction Construction Stage

To enhance the diversity and controllability of generated instructions, we forgo single-shot generation. Instead, we propose a compositional approach where each instruction is decomposed into two parts: a *query* component $\mathcal{Q}$ and a *constraint set* $\mathcal{C}$.

Query Generation. We generate the query set $\mathcal{Q}$ via a four-stage pipeline, where each stage adds an essential element of the instruction context. The process is as follows:

(1) **Domain Generation:** Define a set of domains $\mathcal{D} = \{d_1, \ldots, d_N\}$, where each d represents a real-world context (e.g., *Math*, *Sports*).

(2) **Request Generation:** For each domain $d \in \mathcal{D}$, we leverage an LLM θ to generate a set of requests $\mathcal{R}(d) = \{r_1, \ldots, r_{M(d)}\}$ describing typical tasks within d.

(3) **Scenario Generation:** For each request $r \in \mathcal{R}(d)$, generate a set of scenarios $\mathcal{S}(r) = \{s_1, \ldots, s_{K(r)}\}$, each representing a different situation or context.

(4) **Persona Assignment:** For each scenario $s \in \mathcal{S}(r)$, generate a set of personas $\mathcal{P}(s) = \{p_1, \ldots, p_{L(s)}\}$, which specify user perspective or difficulty level.

Finally, each query q is uniquely defined by a tuple (d, r, s, p), where $d \in \mathcal{D}$, $r \in \mathcal{R}(d)$, $s \in \mathcal{S}(r)$, and $p \in \mathcal{P}(s)$. The overall query set is $\mathcal{Q} = \{q_1, \ldots, q_T\}$, analogous to the construction in (Ge et al., 2024)..

Constraint Formulation. In parallel to query generation, we construct a set of constraints $\mathcal{C}$ designed to complement and guide the queries. Unlike prior methods (An et al., 2025; Qi et al., 2025; Zhang et al., 2026b) that rely heavily on constraints extracted from extensive external corpora, our approach focuses on synchronously generating both constraints and their associated validation methods using LLMs, thereby minimizing dependence on external resources and ensuring tight alignment between constraints and their verification logic.

We classify constraints into two distinct categories following (Peng et al., 2025): *soft constraints* ($\mathcal{C}_{\text{soft}}$), which represent stylistic or qualitative guidelines validated through model-based heuristics (e.g., tone or politeness), and *hard constraints* ($\mathcal{C}_{\text{hard}}$), which enforce strict and deterministic rules verifiable by explicit checks or scripts (e.g., length limitations or keywords requirements). Formally, the constraint set is expressed as:

$$\begin{cases} \mathcal{C}_{\text{soft}} = \{c_1^s, c_2^s, \ldots, c_{N_s}^s\}, \\ \mathcal{C}_{\text{hard}} = \{c_1^h, c_2^h, \ldots, c_{N_h}^h\}, \\ \mathcal{C} = \mathcal{C}_{\text{soft}} \cup \mathcal{C}_{\text{hard}}. \end{cases} \tag{1}$$

where c_i^s and c_j^h denote the i-th soft constraint and the j-th hard constraint, respectively, and N_s (N_h) is the number of soft (hard) constraints. The generation process for each constraint category is designed to yield both constraints and their corresponding validation mechanisms in a tightly integrated, template-based manner.

(1) **Soft Constraint Generation.** We first prompt the LLM θ to generate a set of high-level constraint categories $\mathcal{T} = \{t_1, \ldots, t_P\}$ that capture stylistic or content-related properties. For each $t \in \mathcal{T}$, θ is then prompted to simultaneously produce (i) a template-style soft constraint (e.g., the response should be in [placeholder] tone") and (ii) a corresponding template-style validation question (e.g., Is the response written in a [placeholder] tone?"). This joint generation process ensures that each constraint $c_i^s \in \mathcal{C}_{\text{soft}}$ is aligned with a specific verification question $v_i^s \in \mathcal{V}^s = \{v_1^s, \ldots, v_{N_s}^s\}$.

(2) **Hard Constraint Generation.** We begin with a small seed set of hard constraints $\mathcal{C}_{\text{seed}}^h = \{c_1^h, \ldots, c_M^h\}$ and their corresponding verification

Method	#Data	IFEval				Multi-IF			FollowBench		LiveBench
		Pr (S)	In (S)	Pr (L)	In (L)	Turn 1	Turn 2	Turn 3	HSR	SSR	Score
Directly utilize the open source dataset											
ShareGPT	10k	39.00	50.24	42.70	53.96	40.25	28.51	23.13	40.52	57.97	31.00
Evol-Instruct	10k	39.37	50.48	42.14	53.60	33.45	21.36	14.34	29.78	49.13	26.90
Conifer	13k	44.36	56.24	48.98	60.55	44.61	25.52	17.95	45.83	59.98	41.10
AIR	10k	43.99	55.16	47.32	58.51	42.00	23.56	18.94	44.97	61.54	38.50
Condor	20k	55.64	64.99	58.60	67.15	50.74	30.73	23.05	48.10	61.92	41.10
Utilize LLaMA-3.1-70B-Instruct as supervised model											
AutoIF†	10k	47.13	57.55	56.93	67.02	47.63	27.53	20.53	-	60.41	40.50
UltraIF†	10k	53.97	64.15	58.59	68.82	52.55	29.34	22.29	-	59.50	42.20
DecIF	10k	**65.25**	**74.22**	**67.65**	**76.74**	**66.37**	**47.46**	**35.98**	**48.29**	**62.28**	**50.50**
Compare with more challenging dataset											
Tulu-3-IF	30k	68.58	77.22	72.27	80.58	**70.14**	47.34	35.13	49.52	63.03	52.90
UltraIF‡	181k	64.14	72.90	68.39	76.86	65.33	47.71	**39.49**	**53.52**	**67.20**	47.20
DecIF	30k	**71.72**	**79.02**	**74.86**	**81.89**	69.57	**51.27**	38.21	49.25	64.40	**54.30**

Table 1: The SFT results of LLaMA-3.1-8B on four instruction-following benchmarks. Pr and In represent the prompt and instruction levels. S and L stand for strict and loose metrics for IFEval. For LiveBench, we only report the performance of instruction-following subset. Results marked with † mean that we directly report the evaluation results in (An et al., 2025). ‡ represents that we utilize the open-source 181k data from UltraIF.

functions $\mathcal{V}^h_{\text{seed}} = \{v^h_1, \ldots, v^h_M\}$, where each pair encodes a strict and verifiable requirement in a template-based form (e.g., "responses must not exceed [N] words" and a corresponding length-checking script). We then employ an iterative Self-Instruct process (Wang et al., 2023a), where at each iteration, θ is prompted to generate new constraint-function pairs by paraphrasing, modifying, or composing from existing pairs. After L rounds, we collect all unique pairs into the final sets $\mathcal{C}_{\text{hard}} = \{c^h_1, \ldots, c^h_{N_h}\}$ and $\mathcal{V}^h = \{v^h_1, \ldots, v^h_{N_h}\}$, where each v^h_j is a deterministic verification function responsible for validating c^h_j.

Complete Instruction Synthesis. In the final stage, we synthesize complete instructions by first randomly sampling combinations of multiple soft and hard constraints. Instead of directly assuming compatibility, we prompt the LLM θ to perform conflict filtering, but critically, this filtering is not based on the surface forms of the constraints themselves. Instead, we explicitly incorporate their associated validation questions (for soft constraints) and verification functions (for hard constraints) into the filtering prompt. This design allows θ to reason from the perspective of how each constraint will be verified, capturing more subtle and fine-grained interactions that may not be evident from the constraint texts alone. Filtering based on validation logic leads to higher-precision conflict detection.

Once a compatible constraint set is identified, we pair it with a query q and prompt θ to generate a fully specified natural-language instruction. All placeholders in the constraints and validations are instantiated, and the instruction is written to reflect the full set of requirements in a coherent and natural way. This yields the final instruction set:

$$\mathcal{I} = \left\{ \phi(q, \mathcal{C}^s, \mathcal{C}^h) \;\middle|\; q \in \mathcal{Q},\ (\mathcal{C}^s, \mathcal{C}^h) \in \mathcal{F} \right\} \quad (2)$$

where $\mathcal{C}^s \subseteq \mathcal{C}_{\text{soft}}$ and $\mathcal{C}^h \subseteq \mathcal{C}_{\text{hard}}$ denote the subsets of soft and hard constraints, and $\mathcal{F}$ indicates all constraint sets passing the conflict filtering process.

2.2 Response Construction Stage

Given the constructed instruction set $\mathcal{I}$ and the associated validation procedures $\mathcal{V}$, we construct responses in two phases: *response generation* and *reject sampling*.

Response Generation. For each instruction $x = (q, c^s, c^h) \in \mathcal{I}$, we prompt the LLM θ to generate a set of candidate responses:

$$\mathcal{A}(x) = \{\, a_1, \ldots, a_{K(x)} \,\}, \quad (3)$$

where $K(x)$ is the number of responses generated.

Reject Sampling. To ensure all responses adhere to the specified constraints, we perform verification using the paired validation questions and functions described in the previous section. Specifically, for

Method	#Data	IFEval				Multi-IF			FollowBench		LiveBench
		Pr (S)	In (S)	Pr (L)	In (L)	Turn 1	Turn 2	Turn 3	HSR	SSR	Score
Directly utilize the open source dataset											
ShareGPT	10k	49.91	60.43	52.87	63.79	55.36	39.56	31.79	51.02	67.35	38.60
Evol-Instruct	10k	56.01	66.91	60.44	70.74	59.40	41.08	31.23	47.88	66.37	32.40
Conifer	13k	57.49	67.63	64.33	73.50	64.19	22.59	17.25	63.05	73.13	48.60
AIR	10k	62.48	72.42	67.47	76.38	66.84	49.17	37.89	62.50	74.18	43.20
Condor	20k	59.33	69.66	64.33	73.98	70.20	50.35	39.10	63.35	75.38	46.80
Utilize LLaMA-3.1-70B-Instruct as supervised model											
DecIF	10k	**77.45**	**84.05**	**79.30**	**85.61**	**81.18**	**64.57**	**48.87**	**64.84**	**76.95**	**53.10**
Compare with more challenging dataset											
Tulu-3-IF	30k	76.16	82.61	79.30	85.25	81.06	61.85	47.12	**69.00**	**78.51**	41.30
UltraIF‡	181k	68.02	76.98	71.53	79.62	72.26	54.75	45.20	62.12	73.76	49.60
DecIF	30k	**79.48**	**85.49**	**81.33**	**87.29**	**81.65**	**65.92**	**50.95**	67.83	77.57	**57.30**

Table 2: The SFT results of Qwen-3-8B-Base on four instruction-following benchmarks.

Method	#Data	IFEval				Multi-IF			FollowBench		LiveBench
		Pr (S)	In (S)	Pr (L)	In (L)	Turn 1	Turn 2	Turn 3	HSR	SSR	Score
The results of reinforcement learning											
R1-Distill-Qwen-7B	-	58.41	68.35	61.37	71.22	60.17	47.64	35.58	49.05	64.41	61.70
+VerIF‡	22k	67.84	77.22	71.16	79.74	73.14	59.84	47.28	**57.77**	70.79	**71.30**
+DecIF	22k	**69.87**	**78.54**	**73.20**	**81.06**	**73.27**	**60.53**	**49.26**	57.69	**72.04**	67.20
Tulu-3-SFT	-	63.22	73.14	67.10	76.02	67.48	52.06	40.67	48.93	64.31	44.00
+VerIF‡	22k	79.48	85.25	81.70	87.41	80.71	66.97	55.75	64.15	73.67	**60.00**
+DecIF	22k	**85.95**	**90.89**	**88.17**	**92.33**	**83.86**	**71.36**	**59.64**	**65.00**	**74.39**	59.50

Table 3: The RL results of R1-Distill-Qwen-7B and Tulu-3-SFT on four instruction-following benchmarks. ‡ means we directly download the source model published by authors.

each candidate response $a \in \mathcal{A}(x)$, we define the set of valid responses as:

$$\mathrm{A}^*(x) = \left\{ a \in \mathcal{A}(x) \;\middle|\; v^s(a) = 1 \;\wedge\; v^h(a) = 1 \right\}, \quad (4)$$

where v^s and v^h are the validation functions corresponding to the soft and hard constraints.

We use the resulting set $\mathcal{A}^*(x)$ and its filtered variants to construct training data for different learning paradigms:

(1) **Supervised Fine-Tuning (SFT):** The entire set $\mathcal{A}^*(x)$—that is, all responses passing both soft and hard constraint validations—is used as the SFT training data.

(2) **Reinforcement Learning (RL):** For RL, we retain only those instructions whose associated validation logic is complete and unambiguous. Specifically, we exclude any instructions that contain incomplete validation questions, or whose verification functions result in runtime errors or timeouts.

2.3 Reward Strategy in RL

For reinforcement learning, we employ a soft reward (Ping et al., 2026) scheme that reflects partial compliance with constraints, rather than using only binary rewards (i.e., assigning 1 for fully correct and 0 otherwise). Formally, for an instruction $x = (q, c^s, c^h)$ with m total constraints and a response a, let $S(a)$ denote the set of constraints satisfied by a according to the constraint validation functions. The reward $r(a)$ is defined as

$$r(a) = \frac{|S(a)|}{m}, \quad (5)$$

where $|S(a)|$ is the number of constraints satisfied by a. For example, if $m = 5$ and $|S(a)| = 3$, then $r(a) = 0.6$.

3 Experiments

3.1 Experimental Setup

Detailed information about baselines, evaluation benchmarks, hyperparameters, and prompt templates can be found in Appendix A and D.

Baselines. Following (An et al., 2025), we use LLaMA-3.1-70B-Instruct for supervised data generation, and adopt a wide range of SFT baselines,

including ShareGPT (Chiang et al., 2023), Evol-Instruct (Xu et al., 2023), Conifer (Sun et al., 2024), Condor (Cao et al., 2025), AIR (Liu et al., 2025), AutoIF (Dong et al., 2024a), and UltraIF (An et al., 2025). We further include the instruction-following subset of Tulu-3 (Lambert et al., 2025) and UltraIF's open-source 181k set as challenging baselines. For RL settings, we directly compare with VerIF (Peng et al., 2025).

Evaluation Benchmarks. We evaluate on four instruction-following benchmarks: IFEval (Zhou et al., 2023), Multi-IF (He et al., 2024), FollowBench (Jiang et al., 2024), and LiveBench (White et al., 2025). Additional benchmarks include GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021b) (math), HumanEval (Chen et al., 2021) (code), BBH (Suzgun et al., 2022) and HellaSwag (Zellers et al., 2019) (reasoning), GPQA (Rein et al., 2023) and MMLU (Hendrycks et al., 2021a) (knowledge), Arena Hard (Li et al., 2024a) (conversation), and LiveBench (all) (White et al., 2025) (general). This comprehensive evaluation ensures fair comparison across reasoning, knowledge, coding, and interactive abilities.

3.2 Implementation Details

We use vLLM (Kwon et al., 2023) for efficient inference throughout the data generation pipeline. The LLM is prompted iteratively to generate domains, requests, scenarios, and personas, resulting in over 1M unique queries. For constraints, we construct 32 soft constraint categories and 87 templates, and expand a seed set of 5 hard constraints (with corresponding verification functions) to obtain 41 final hard constraint templates. Data synthesis uses temperature 0.6, top_p 0.95, and max_tokens 8192. Model evaluation is performed with greedy decoding for stability. For training, we use LLaMA-Factory (Zheng et al., 2024) for SFT and VeRL (Sheng et al., 2024) for RL, setting a learning rate of 1×10^{-5} for SFT (3 epochs) and 1×10^{-6} for RL (16 rollouts, 1 epoch, and GRPO (Shao et al., 2024) optimized algorithm). To avoid data leakage, we remove any training instruction with Jaccard similarity above 0.7 to evaluation data, following (Zhou et al., 2025).

3.3 Supervised Fine-Tuning Results

Table 1 and 2 present the performance of DecIF on four instruction-following benchmarks compared to other baselines. We apply SFT to two base models (LLaMA-3.1-8B and Qwen-3-8B-Base) and the results demonstrate that DecIF significantly outperforms all baselines. Specifically, compared to UltraIF, our method achieves nearly 8-18% improvements on IFEval, MultiIF, and LiveBench with the same amount of training data, along with nearly a 3% enhancement on FollowBench. When scaling up to 30k training data, DecIF continues to achieve the best performance on most benchmarks, even when compared to the instruction-following subset of Tulu-3 and UltraIF-181k. Overall, DecIF constructs accurate and diverse instruction-following data using only LLMs and minimal external resources, and achieves substantial performance improvements over prior approaches and datasets.

Method	Code	Reasoning	Math	Conversation	General
	HumanEval	BBH	GSM8K	Arena Hard	LiveBench [All]
The common abilities of SFT models					
AutoIF†	46.34	67.18	51.50	9.20	17.50
UltraIF†	43.90	67.33	48.60	12.20	21.30
DecIF	**46.95**	**67.45**	**60.42**	**12.20**	**21.90**
The common abilities of RL models					
Tulu-3-SFT	61.59	67.64	78.32	16.80	24.80
VerIF‡	58.54	67.93	77.86	**17.20**	25.80
DecIF	**65.85**	**68.80**	**79.23**	16.60	**28.40**

Table 4: The common capability results of LLaMA-3.1-8B (for SFT) and Tulu-3-SFT (for RL) on code, reasoning, math, conversation and general domains. Results marked with † mean that we directly report the evaluation results in (An et al., 2025).

3.4 Reinforcement Learning Results

Table 3 shows the RL results on R1-Distill-Qwen-7B and Tulu-3-SFT. The results demonstrate that our RL data exhibits significant performance improvements compared to VerIF (Peng et al., 2025), particularly on the more instruction-following-capable Tulu-3-SFT model. This indicates that the data constructed by DecIF combined with the soft reward strategy enables more powerful foundation models to achieve enhanced capabilities, which also indirectly highlights DecIF's strong scalability potential. Notably, during our training process, we employed CompassJudger-1-7B (Cao et al., 2024) as the judge model for hard constraints rather than using larger 32B or even 70B-scale models, which further enhanced training efficiency.

3.5 Common Capabilities Evaluation

To ensure that the constructed instruction-following SFT and RL data does not negatively impact or conflict with common capabilities, we follow (An et al., 2025) and conduct a comprehensive evalua-

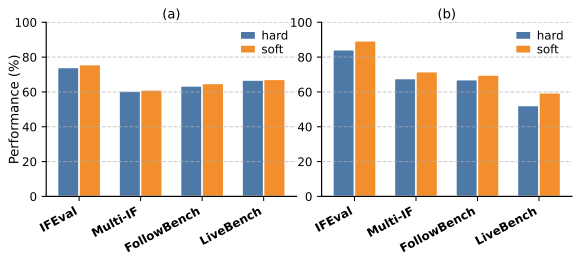

Figure 2: (a) The results of different reward strategies on R1-Distill-Qwen-7B. (b) The results of different reward strategies on Tulu-3-SFT.

tion across coding, math, reasoning, conversation, and general capabilities. Based on the data construction process of DecIF outlined earlier, the resulting instruction data is more accurate and diverse than previous methods, encompassing a broader range of instructions across various domains. As shown in Table 4, DecIF outperforms AutoIF, UltraIF and VerIF in common capabilities. This further indicates that the data synthesized by DecIF are more diverse and comprehensive in domains.

3.6 Ablation Study

Table 5 summarizes the results of our ablation studies on DecIF. During the SFT data synthesis stage, we ablate several fine-grained generation steps as well as the reject sampling process. We observe that as the synthesis becomes more direct, iteratively removing our fine-grained procedure, the model performance consistently degrades, primarily due to reduced query diversity. Removing reject sampling also harms performance by introducing a large amount of inaccurate training data. In the RL training stage, eliminating the verification function significantly degrades performance on rule-based benchmarks such as IFEval. Similarly, removing the model validation step also leads to notable performance drops.

4 Further Analysis

More analysis on various base models, long-CoT SFT, and efficiency analysis can be found in Appendix C.

4.1 Soft Rewards v.s Hard Rewards

As shown in Figure 2, we compare the performance of soft reward and hard reward strategies on R1-Distill-Qwen-7B and Tulu-3-SFT. We find that the soft reward strategy demonstrates superior performance. *This also indicates that fine-grained rewards offer greater advantages compared to simple binary rewards, inspiring future research directions to explore more fine-grained reward strategies.*

Method	IFEval	Multi-IF	FollowBench	LiveBench
The ablation study of data synthesis				
DecIF	**70.97**	**49.94**	**55.29**	**50.50**
w/o `d`	68.89	48.12	53.07	47.90
w/o `d+r`	66.17	46.78	50.24	47.20
w/o `d+r+s`	63.52	44.89	47.38	40.40
w/o `reject`	65.35	44.70	51.13	45.40
The ablation study of RL training				
DecIF	**89.34**	**71.62**	**69.70**	**59.50**
w/o `code`	77.63	60.32	66.75	53.60
w/o `llm`	87.65	69.84	63.52	56.70

Table 5: The comprehensive ablation study of data synthesis and RL training stages. `d`, `r`, `s`, and `reject` represent domains, requests, scenarios, and reject sampling. `code` and `llm` represent the rewards from Python code and LLMs.

4.2 Scaling Capability of DecIF

As shown in Figure 3 (a) and (b), we investigate the effects of data scaling during both the SFT and RL phases, respectively. Specifically, we observe that model performance first improves and then declines as the amount of data increases. More data does not always lead to better results. The experiments demonstrate that the model achieves optimal instruction-following performance with around 30K samples for SFT and 20K samples for RL. *These findings suggest that future work should prioritize data accuracy and diversity within a limited sample size, rather than indiscriminately increasing data quantity.*

4.3 Mixture with Other SFT Data

Given that the SFT stage of recent advanced models (Meta, 2024) typically relies on a substantial volume of high-quality, cross-domain data, we argue that top-tier instruction-following datasets should not only excel when trained in isolation, but also demonstrate strong compatibility with training data from diverse domains. Crucially, such datasets should avoid introducing conflicts that could undermine overall model performance. To evaluate this, we replace the instruction-following component of Tulu-3 with data synthesized by DecIF. As shown in Figure 3 (c), substituting the original data with our synthesized dataset results in improved performance on IFEval, while maintaining stable performance across other domains without any signs of degradation. *This outcome clearly underscores the*

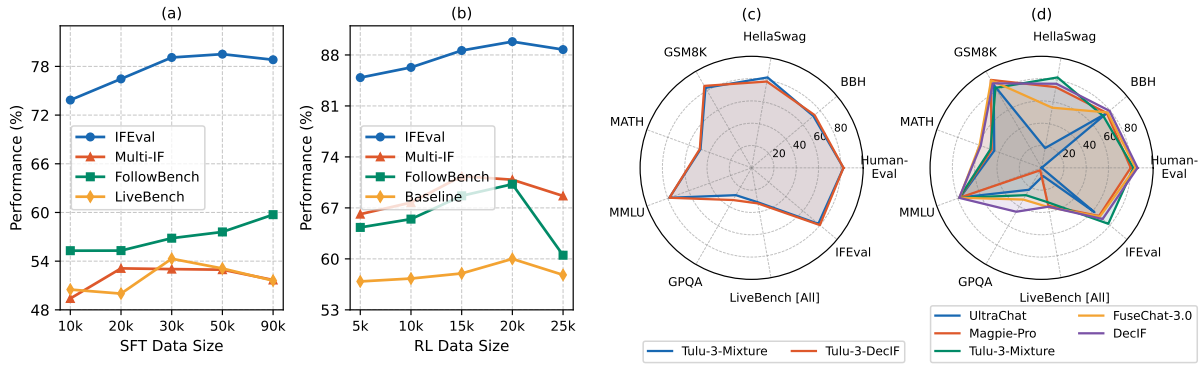

Figure 3: (a, b) The results of data scaling. (c) The results on Tulu-3-Mixture and Tulu-3-DecIF. (d) The results of different large-scale general-purpose instruction data.

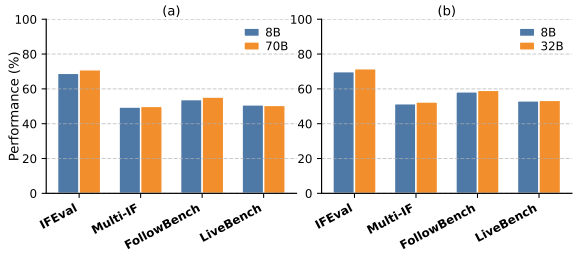

Figure 4: The results of LLaMA-3.1-8B training on different SFT data which is generated by different supervised models. (a) Utilize LLaMA-3.1-8B-Instruct and LLaMA-3.1-70B-Instruct as supervised models. (b) Utilize Qwen-3-32B and Qwen-3-8B as supervised models.

compatibility of our synthesized data with multi-domain training data, thereby further highlighting the broad applicability and potential of DecIF in diverse training contexts.

4.4 Large-Scale SFT Data Synthesis of DecIF

We observe DecIF's strong potential to synthesize large-scale, general-purpose data. To explore this capability, we remove constraints generation stage from the original pipeline and directly generate queries. Using Qwen-3-32B as supervised model, we obtain approximately 150k diverse instruction-response pairs. As shown in Figure 3 (d), the large-scale general-purpose data synthesized by DecIF achieves competitive performance when compared to recent advanced datasets such as UltraChat (Ding et al., 2023), Magpie-pro (Xu et al., 2024), Tulu-3-Mixture, and FuseChat-3.0 (Yang et al., 2025b). Compared to UltraChat, Tulu-3 and FuseChat-3.0, DecIF does not rely heavily on pre-existing documents or external resources to synthesize data. In contrast to Magpie, which requires complex filtering mechanisms to eliminate low-quality or unparseable outputs, DecIF employs a more controllable approach based on decomposition. *We believe that DecIF also holds great potential in generating instruction data. By incorporating additional control signals, the synthesis process can be further refined.*

4.5 Exploration of Different Supervised Models

As shown in Figure 4, we investigate the performance of LLaMA-3.1 series (70B and 8B) and Qwen-3 series (32B and 8B) as supervised models for synthesizing instruction-following data through DecIF. To our surprise, we find that smaller models (8B) generated synthetic data performing comparably to larger models (32B and 70B) across four IF benchmarks. This achievement stems from DecIF's decomposition-based synthesis principle and our code-LLM hybrid verification mechanism. Through these innovations, we break down complete instruction synthesis into fine-grained steps, enabling smaller models to perform high-quality generation. Moreover, our precise reject sampling scheme effectively filters out inaccurate responses from smaller models, ultimately deriving data quality on par with larger models. This establishes DecIF as *a more flexible and reliable framework for enhancing instruction-following capabilities.*

5 Related Work

Instruction-Following. Instruction-following is fundamental for aligning LLMs with complex human intent (Li et al., 2024b; Dong et al., 2024a,b). Early work such as ShareGPT (Chiang et al., 2023) leveraged user-shared dialogues for tuning, while later approaches reduce reliance on human annotation via synthetic generation—using

back-translation (Li et al., 2024b), iterative refinement (Sun et al., 2024; Liu et al., 2025), or code execution feedback (Dong et al., 2024a). Recent methods further improve alignment by explicitly incorporating constraint validation into data construction and model training, leveraging rule-based or code-based verification for hard constraints and LLM-based heuristics for soft constraints (Peng et al., 2025). Decomposition-based frameworks such as UltraIF (An et al., 2025) generate more diverse and structured data by splitting prompts into sub-queries and constraints.

Data Synthesis. Automated data synthesis is critical for scaling LLMs instruction tuning beyond expensive manual annotation. Approaches such as Self-Instruct (Wang et al., 2023b), WizardLM (Xu et al., 2023), Condor (Cao et al., 2025), PersonaHub (Ge et al., 2024), and Magpie (Xu et al., 2024) iteratively generate large-scale, diverse instruction data using minimal or no human supervision, leveraging bootstrapping, knowledge-driven generation, or persona-based diversification. Building on these advances, DecIF introduces a fully automated process that jointly synthesizes instructions, constraints, and validation logic with only LLMs and minimal external resources, enabling accurate and diverse instruction-following data construction.

6 Conclusion

We introduce DecIF, a fully automated framework for generating accurate, diverse, and verifiable instruction-following data for SFT and RL. DecIF's modular decomposition and validation-aware design ensure both high diversity and reliability without heavy dependence on external resources or existing datasets. Experiments on instruction-following and comprehensive general benchmarks confirm significant improvements over existing methods and demonstrate strong generalization. DecIF lays a solid foundation for future advances in high-quality instruction-following data synthesis.

Limitations

In this paper, we propose DecIF, a method that enables the synthesis of high-quality instruction-following data from scratch, without relying heavily on existing documents or datasets. Despite DecIF's superior performance, it still has several limitations.

(1) Although we have explored the potential of DecIF in synthesizing large-scale, general-purpose instruction data, due to space and scope limitations, we do not further investigate its full capacity. In future work, we plan to conduct a more in-depth study of DecIF's ability to generate large-scale instruction data, with finer-grained control, to further advance the field.

(2) In this paper, we primarily focus on single-turn English instruction-following data. We plan to explore DecIF's potential in generating multi-turn dialogue data and instruction data in other languages in future work.

(3) In the reinforcement learning phase, this paper does not further explore deeper training dynamics. We also plan to make further advancements in RL for instruction-following in the future.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (62072052) and the National Key Research and Development Program of China (2024YFF0907401). We sincerely appreciate the academic support, which has been crucial to the successful completion of this study.

References

Kaikai An, Li Sheng, Ganqu Cui, Shuzheng Si, Ning Ding, Yu Cheng, and Baobao Chang. 2025. Ultraif: Advancing instruction following from the wild. *Preprint*, arXiv:2502.04153.

Maosong Cao, Alexander Lam, Haodong Duan, Hongwei Liu, Songyang Zhang, and Kai Chen. 2024. Compassjudger-1: All-in-one judge model helps model evaluation and evolution. *Preprint*, arXiv:2410.16256.

Maosong Cao, Taolin Zhang, Mo Li, Chuyu Zhang, Yunxin Liu, Haodong Duan, Songyang Zhang, and Kai Chen. 2025. Condor: Enhance llm alignment with knowledge-driven data synthesis and refinement. *Preprint*, arXiv:2501.12273.

Jiawei Chen, Xinyan Guan, Qianhao Yuan, Guozhao Mo, Weixiang Zhou, Yaojie Lu, Hongyu Lin, Ben He, Le Sun, and Xianpei Han. 2025. Consistentchat: Building skeleton-guided consistent multi-turn dialogues for large language models from scratch. In *The 2025 Conference on Empirical Methods in Natural Language Processing*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde 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, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv: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. *Preprint*, arXiv:2110.14168.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. *Preprint*, arXiv:2305.14233.

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

Guanting Dong, Xiaoshuai Song, Yutao Zhu, Runqi Qiao, Zhicheng Dou, and Ji-Rong Wen. 2024b. Toward general instruction-following alignment for retrieval-augmented generation. *Preprint*, arXiv:2410.09584.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. 2024. Scaling synthetic data creation with 1,000,000,000 personas. *Preprint*, arXiv:2406.20094.

Yun He, Di Jin, Chaoqi Wang, Chloe Bi, Karishma Mandyam, Hejia Zhang, Chen Zhu, Ning Li, Tengyu Xu, Hongjiang Lv, Shruti Bhosale, Chenguang Zhu, Karthik Abinav Sankararaman, Eryk Helenowski, Melanie Kambadur, Aditya Tayade, Hao Ma, Han Fang, and Sinong Wang. 2024. Multi-if: Benchmarking llms on multi-turn and multilingual instructions following. *Preprint*, arXiv:2410.15553.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021a. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

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. *Preprint*, arXiv:2103.03874.

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. *Preprint*, arXiv:2310.20410.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. 2024a. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *Preprint*, arXiv:2406.11939.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason Weston, and Mike Lewis. 2024b. Self-alignment with instruction backtranslation. *Preprint*, arXiv:2308.06259.

Wei Liu, Yancheng He, Hui Huang, Chengwei Hu, Jiaheng Liu, Shilong Li, Wenbo Su, and Bo Zheng. 2025. Air: Complex instruction generation via automatic iterative refinement. *Preprint*, arXiv:2502.17787.

Meta. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

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

Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2025. Verif: Verification engineering for reinforcement learning in instruction following. *Preprint*, arXiv:2506.09942.

Bowen Ping, Zijun Chen, Tingfeng Hui, Qize Yu, Chenxuan Li, Junchi Yan, and Baobao Chang. 2026. Longact: Harnessing intrinsic activation patterns for long-context reinforcement learning. *Preprint*, arXiv:2604.14922.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2025. Constraint back-translation improves complex instruction following of large language models. *Preprint*, arXiv:2410.24175.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *Preprint*, arXiv:2311.12022.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024.

Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2024. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*.

Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. 2024. Conifer: Improving complex constrained instruction-following ability of large language models. *Preprint*, arXiv:2404.02823.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *Preprint*, arXiv:2210.09261.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023a. Self-instruct: Aligning language models with self-generated instructions. *Preprint*, arXiv:2212.10560.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023b. 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)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Colin White, Samuel Dooley, Manley Roberts, Arka Pal, Ben Feuer, Siddhartha Jain, Ravid Shwartz-Ziv, Neel Jain, Khalid Saifullah, Sreemanti Dey, Shubh-Agrawal, Sandeep Singh Sandha, Siddartha Naidu, Chinmay Hegde, Yann LeCun, Tom Goldstein, Willie Neiswanger, and Micah Goldblum. 2025. Livebench: A challenging, contamination-limited llm benchmark. *Preprint*, arXiv:2406.19314.

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. *Preprint*, arXiv:2304.12244.

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

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Ziyi Yang, Fanqi Wan, Longguang Zhong, Canbin Huang, Guosheng Liang, and Xiaojun Quan. 2025b. Fusechat-3.0: Preference optimization meets heterogeneous model fusion. *Preprint*, arXiv:2503.04222.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *Preprint*, arXiv:1905.07830.

Jiajun Zhang, Zeyu Cui, Jiaxi Yang, Lei Zhang, Yuheng Jing, Zeyao Ma, Tianyi Bai, Zilei Wang, Qiang Liu, Liang Wang, Binyuan Hui, and Junyang Lin. 2026a. From completion to editing: Unlocking context-aware code infilling via search-and-replace instruction tuning. *Preprint*, arXiv:2601.13384.

Jiajun Zhang, Yuying Li, Zhixun Li, Xingyu Guo, Jingzhuo Wu, Leqi Zheng, Yiran Yang, Jianke Zhang, Qingbin Li, Shannan Yan, Zhetong Li, Changguo Jia, Junfei Wu, Zilei Wang, Qiang Liu, and Liang Wang. 2026b. Realchart2code: Advancing chart-to-code generation with real data and multi-task evaluation. *Preprint*, arXiv:2603.25804.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

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

Xin Zhou, Martin Weyssow, Ratnadira Widyasari, Ting Zhang, Junda He, Yunbo Lyu, Jianming Chang, Beiqi Zhang, Dan Huang, and David Lo. 2025. Lessleak-bench: A first investigation of data leakage in llms across 83 software engineering benchmarks. *Preprint*, arXiv:2502.06215.

Appendix

A Detailed Experimental Setup

In this section, we provide detailed explanations of the components that are omitted or only briefly mentioned in the main text.

A.1 Detailed Information about Baselines

In this paper, we compare DecIF with a comprehensive set of baselines to thoroughly validate its effectiveness. Specifically,

ShareGPT(Chiang et al., 2023) is an open-source and multi-turn conversation dataset that contains 52k user-shared chatting histories with GPT-4. We randomly select 10k subset to serve as the baseline.

Evol-Instruct (Xu et al., 2023) is a large-scale complex instruction dataset that evolves existing instructions across both depth and breadth. We also randomly select a 10k subset to serve as the baseline.

Conifer (Sun et al., 2024) is a 13k instruction-following dataset synthesized from seed instructions derived from ShareGPT, using a three-stage process involving query reframing, constraint generation, and recombination. We directly utilize the full set as the baseline.

AIR (Liu et al., 2025) collects initial instructions from existing documents and performs iterative instruction refinement to construct a 10k instruction-following dataset. We use the full set of the data as the baseline.

Condor (Cao et al., 2025) incorporate world knowledge tree to synthesize a large-scale instruction data. We directly use the open-source 20k data to serve as the baseline.

AutoIF (Dong et al., 2024a) manually design a variety of constraints and innovatively employ Python functions to evaluate the responses. In this paper, we directly report the performance in (An et al., 2025).

UltraIF (An et al., 2025) incorporate existing documents and datasets to train the UltraComposer and synthesize a large-scale instruction-following data. We also directly report the performance in its original paper.

For more challenging comparison and validate the generalizability of DecIF, we also utilize **Tulu-3** (Lambert et al., 2025) and **Magpie** (Xu et al., 2024) as our baselines.

For RL baselines, we compare DecIF with VerIF (Peng et al., 2025) witch utilize Constraint Back-translation (Qi et al., 2025) to synthesize hard and soft constraints.

A.2 Detailed Description of Evaluation Benchmarks

For the evaluation of instruction-following capability, we utilize the following benchmarks:

IFEval (Zhou et al., 2023) is an easily producible benchmark specifically designed to assess the instruction-following capabilities of LLMs. It consists of approximately 500 prompts, each containing 25 types of verifiable instructions. In our evaluation, we employ both loose and strict accuracy metrics at both the prompt and instruction levels.

Multi-IF (He et al., 2024) is a benchmark designed to evaluate LLMs' proficiency in following multi-turn and multilingual instructions. It comprises 4,501 multilingual conversations, each consisting of three turns. We report the average accuracy across all languages for each of the three rounds in the experiment.

FollowBench (Jiang et al., 2024) is a multi-level, fine-grained constraint-following benchmark designed for LLMs. It integrates five distinct types of fine-grained constraints, Content, Situation, Style, Format, and Example, and emphasizes a multi-level mechanism in the construction of instruction prompts. In our experiments, we utilize GPT-4o-2025-03-26 to evaluate whether the outputs of LLMs satisfy each individual constraint.

LiveBench (White et al., 2025) is an LLM benchmark that encompasses a wide array of challenging tasks, including math, coding, reasoning, language, instruction-following, and data analysis, with answers automatically scored based on objective ground-truth values. We use the instruction-following subset to asses the instruction-following capability and utilize all data to evaluate the general capability.

For other common abilities, we incorporate the following benchmarks to broadly assess the models:

GSM8K (Cobbe et al., 2021) consists of 8.5K high-quality, multilingual grade school math word problems, meticulously crafted to evaluate the multi-step mathematical reasoning proficiency of language models. In the experiment, we report the overall accuracy achieved by the models.

MATH (Hendrycks et al., 2021b) is a challenging benchmark containing 12,500 high school-level mathematics problems spanning algebra, geometry, and more. Each problem includes a textual description and a step-by-step solution. The benchmark is designed to assess the ability of language models to perform formal mathematical problem-solving. In the experiment, we report the accuracy of final answers predicted by the models.

HumanEval (Chen et al., 2021) comprises 164 programming problems, each including function signatures, docstrings, bodies, and unit tests, with an average of 7.7 tests per problem. It is designed to evaluate the coding capabilities of LLMs. HumanEval assesses the models' ability to synthesize programs from docstrings, testing their language comprehension, reasoning, algorithmic thinking, and elementary mathematics skills. In the experi-

Method	#Data	IFEval				Multi-IF			FollowBench		LiveBench
		Pr (S)	In (S)	Pr (L)	In (L)	Turn 1	Turn 2	Turn 3	HSR	SSR	Score
Results of Tulu-3 instruction-following subset											
Qwen-3-4B	30k	75.42	82.37	**78.37**	**85.01**	**78.80**	60.65	44.54	61.96	72.41	**53.10**
Qwen-3-14B	30k	77.26	83.93	81.52	**87.92**	67.71	61.47	49.86	**70.72**	**79.62**	44.90
Gemma-3-4B	30k	60.26	69.90	63.96	73.74	67.12	41.92	30.25	50.85	60.86	**48.20**
Gemma-3-12B	30k	68.95	77.58	72.27	80.34	75.85	51.54	39.89	64.43	**75.82**	59.60
Results of DecIF instruction-following data											
Qwen-3-4B	30k	**76.16**	**82.85**	77.82	84.41	78.21	**64.73**	**51.74**	**62.50**	**72.85**	50.70
Qwen-3-14B	30k	**80.04**	**85.49**	**82.62**	87.77	**83.00**	**71.09**	**60.27**	69.43	78.74	**61.10**
Gemma-3-4B	30k	**65.62**	**74.10**	**68.76**	**76.98**	**70.58**	**48.03**	**35.63**	**52.27**	**61.05**	47.30
Gemma-3-12B	30k	**72.27**	**79.74**	**74.86**	**82.13**	**76.32**	**55.17**	**41.63**	**65.01**	75.37	**61.20**

Table 6: The results on four instruction-following benchmarks under Qwen-3 and Gemma-3 series models. Note that we utilize LLaMA-3.1-70B-Instruct as supervised model.

Method	#Data	IFEval				Multi-IF			FollowBench		LiveBench
		Pr (S)	In (S)	Pr (L)	In (L)	Turn 1	Turn 2	Turn 3	HSR	SSR	Score
Results of Qwen-3-32B											
Qwen-3-32B (w/o think)	30k	**84.84**	**89.57**	87.62	91.61	86.95	78.79	71.64	**76.01**	**82.37**	83.40
Qwen-3-32B (w/ think)	30k	84.29	88.85	**88.91**	**92.33**	**87.63**	**79.62**	**71.98**	75.52	80.58	**83.90**
Results of non-thinking instruction-following data											
Qwen-3-4B	30k	74.31	**82.13**	78.19	**85.01**	74.16	61.43	**50.71**	65.84	75.27	51.90
Qwen-3-8B	30k	**78.56**	84.65	81.89	**87.53**	80.86	**66.39**	**55.32**	**72.95**	**80.24**	61.20
Qwen-3-14B	30k	**80.41**	85.61	**84.66**	**88.97**	82.02	71.71	**62.21**	**73.58**	**81.14**	67.40
Results of long-CoT instruction-following data											
Qwen-3-4B	30k	**74.86**	82.01	78.19	84.53	**76.32**	**62.00**	46.71	**66.56**	74.46	**61.70**
Qwen-3-8B	30k	78.19	**84.77**	81.89	87.29	**81.91**	63.50	49.66	70.84	77.51	**68.20**
Qwen-3-14B	30k	79.48	85.61	82.62	87.89	**84.51**	**73.04**	54.41	71.96	78.64	**71.90**

Table 7: The results on four instruction-following benchmarks under various base models with long-CoT data. Note that we utilize Qwen-3-32B as supervised model.

ment, we report the Pass@1 score on HumanEval.

BBH (Suzgun et al., 2022) is a clean, challenging, and tractable subset benchmark filtered from Big Bench, comprising 23 types of difficult tasks and a total of 6,511 evaluation examples. BBH primarily focuses on comprehensively assessing the reasoning capabilities and problem-solving skills of models. In the experiment, we report accuracy metrics on BBH.

HellaSwag (Zellers et al., 2019) is a challenging benchmark designed to evaluate the commonsense reasoning capabilities of language models through sentence completion tasks. It comprises approximately 70,000 multiple-choice questions, each presenting a context followed by four possible endings. Only one of these endings is correct. In the experiments, we report the model's accuracy in selecting the correct ending.

GPQA (Rein et al., 2023) is a highly challenging subset of the GPQA benchmark, consisting of 198 multiple-choice questions across biology, physics, and chemistry. The Diamond subset is distinguished by its stringent validation criteria. In the experiments, we report the model's accuracy in selecting the correct answer.

MMLU (Hendrycks et al., 2021a) is a comprehensive benchmark designed to evaluate the multitask accuracy of language models across a diverse set of academic and professional subjects. It encompasses 57 tasks, including areas such as elementary mathematics, U.S. history, computer science, law, and medicine, totaling approximately 15,908 multiple-choice questions. In the experiments, we report the average accuracy across all tasks.

Arena Hard (Li et al., 2024a) is an automated LLM benchmark comprising 500 challenging user queries, carefully curated to evaluate the comprehensive performance of LLMs in user dialogue scenarios. In the experiment, we use GPT-4o-2025-

03-26 as the judge model and report the win rate of our models compared to the baseline model (GPT-4-0314).

B Step-by-Step Examples of DecIF

In this section, we present concrete example cases from each stage of the DecIF synthetic instruction-following data generation process to help readers better visualize and understand the overall workflow.

For the query generation stage, we sequentially synthesize four pieces of meta information. A specific example is as follows:

Sports (domain) → *Write a news* (request) → *Someone wants to write a sports news article reporting on a recent football match between two major teams* (scenario) → *A high school journalism student wants to write a sports news article about a recent football match between two major teams for the school newspaper* (persona).

For the constraint formulation stage, we decompose the constraints into soft and hard constraints. For soft constraints, we first prompt LLMs to design a set of high-level constraint categories. Then, we utilize LLMs to generate template-style constraints and validation questions based on the categories. A specific example of the category and its constraint template is:

Style (category) → *The response should be in [placeholder] style* (template-style soft constraint) & *Is the response satisfy the [placeholder] style* (validation question).

For hard constraints, we start with a seed set of rule-based constraints and their corresponding verification functions, while we expand the set using LLMs. A specific example of the final hard constraint set can be:

The response should be under [placeholder] characters (template-style hard constraint) & *def evaluate(response, max_word=[placeholder]: return response.split() <= max_word* (verification function).

C Further Experiments

C.1 More SFT Results on Various Base Models

To further validate the effectiveness of DecIF, we conduct experiments across a variety of base models. Specifically, we compare DecIF with the instruction-following subset of Tulu-3, using several state-of-the-art models such as Qwen-3-4B, Qwen-3-14B, Gemma-3-4B, and Gemma-3-12B. Detailed results are presented in Table 6. DecIF consistently achieves strong performance across most benchmarks, demonstrating its broad applicability and effectiveness.

C.2 Long-CoT SFT

Recently, the long-chain-of-thought (long-CoT) training approach has demonstrated strong performance across various domains, particularly in mathematical reasoning and code generation. In this section, we investigate its potential within instruction-following scenarios. Specifically, we employ Qwen-3-32B to generate long-CoT responses for our synthesized instructions and use these responses to train a range of base models. As presented in Table 7, models trained with long-CoT data exhibit a performance drop on IFEval compared to non-thinking baselines, yet achieve substantial improvements on LiveBench. Furthermore, in the Multi-IF benchmark, thinking models show superior performance in the first turn, but their effectiveness diminishes as the number of interaction turns increases. In contrast, non-thinking models perform relatively better in later turns. We hypothesize that this is due to the single-turn nature of our training data, thinking models are not exposed to multi-turn long-CoT dialogues during training, which limits their ability to maintain coherent reasoning over extended interactions. Overall, incorporating long-CoT data during the SFT stage yields notable gains for certain input types, with minimal adverse effects on others. We believe that long-CoT data holds significant promise for advancing instruction-following capabilities and warrants further exploration as a direction for future research.

	Conflict Filtering	Reject Sampling	RL Filtering
Retention Rate	61.72%	37.69%	83.65%

Table 8: The efficiency analysis of our data synthesizing pipeline.

C.3 Efficiency Analysis

We analyze the efficiency of DecIF's data synthesis process, evaluating three key retention metrics as shown in Table 8. First, during constraint conflict filtering, we randomly combine constraints and prompt the model to filter them, achieving around 60% retention rate. Notably, this filtering step

Prompt Template for Domain Generation

Generate exactly {number of domains} real-world domains that these tasks might address.
Criteria:
- Cover everyday life comprehensively.
- Each domain is broad, distinct, and has clear practical value.
- All tasks within the domain must be solvable by a language model.
Examples:
- Education
- Healthcare
- Finance
- Technology
- Travel
- Computer Science
- Artificial Intelligence
- Data Science
- Mathematics
Strict Output Format Requirements
- Each domain must start with a hyphen followed by a space ("- ")
- Do not number the items
- Do not include any additional text, explanations, or formatting
- Do not repeat any examples from the input
- Maintain exactly one domain per line
Output exactly {number of domains} domains in this format:
- Domain A
- Domain B
- Domain C
...

Prompt Template for Request Generation

Generate nearly {number of requests} diverse short task instructions (meta requests) specifically for the {domain} domain.
Requirements
1. Each instruction must be **less than 4 words**, specific, unique, realistic, common, and *model-solvable*.
2. All instructions must be relevant to the {domain} domain.
3. No duplicate instructions within the output.
4. Instructions should be clear and actionable (avoid vague commands like "Do task").
Example output for "Education" domain (Strictly follow this format and use lowercase letters)
- explain the math concept
- grade student essays
- create lesson plan
- suggest teaching methods
- recommend educational apps
Now generate nearly {number of requests} diverse meta requests specifically for the {domain} domain:

Prompt Template for Scenario Generation

For the meta request {meta_request}, generate {count} diverse and realistic scenarios
that would naturally require this action in everyday life. Each scenario should:
1) Include clear contextual elements (time, location, event trigger)
2) Cover different domains (work, study, family, social, etc.)
3) Be no more than 2 sentences and avoid mentioning specific personas
4) Use hyphen formatting (- ...) for each scenario
Example:
Meta request: Book a restaurant
- To celebrate their anniversary, a couple wants to book a seaside restaurant with ocean views for the weekend
- A company needs to book a private room with projector equipment for a team dinner
- While traveling for business, one needs to book a hotel restaurant near the conference center that offers free breakfast
...
Now generate scenarios for:
Meta request: {meta_request}

Prompt Template for Persona Generation

For the following scenario: {scenario}
Generate {count} variations of this scenario, each featuring a distinct persona with an implicit skill level.
Ensure personas come from different domains (e.g., education, healthcare, business, etc.) and subtly convey their skill level through job title or context (e.g., primary school teacher vs. university professor)
Each variation should:
1) Use a dash ("-") at the beginning
2) Be no more than 2 sentences long
3) Clearly embed the persona and context while preserving the original scenario theme.
Example:
Scenario: Someone wants to schedule a video call with a colleague.
- A university professor schedules a late-night video call with a co-author in another time zone to finalize a journal submission
- A customer service representative arranges a video call with a client to resolve a technical issue
- A primary school teacher sets up a video call with a parent to discuss the student's recent behavioral changes
- A freelance designer books a video call with a startup founder to pitch branding ideas
- A hospital administrator schedules a video call with regional clinics to coordinate resource distribution
Start generating now.

Prompt Template for High-Level Constraint Catrgories Generation

Generate {num} high-level primary constraint types for text generation responses.
Requirements: 1. Focus on COMMON user needs (exclude rare/niche requirements)
2. Categories should be MUTUALLY EXCLUSIVE
3. Each type should represent a DISTINCT dimension of constraints
4. Use broad categories like 'Content', 'Format' etc.
Output format:
- [Type Name]: [Description (1 sentence)]
Examples:
- Content Constraints: Requirements about the substance and information included
- Format Constraints: Requirements about the structure and presentation
- Style Constraints: Requirements about linguistic style and tone
- Example Constraints: Requirements about following a limited set of samples

Prompt Template for Template-Style Hard Constraints Generation

You are an expert in evaluating language model outputs through LLM-based semantic constraints.
=== TASK ===
Generate MULTIPLE constraints for: {primary_type}
=== REQUIREMENTS ===
1. Each constraint should:
- Focus on meaning, appropriateness, coherence, style, or nuance
- Be impossible or impractical to verify using strict programmatic code
- Require semantic or subjective judgment
2. For EACH constraint, provide:
- Natural language description
- A precise yes/no verification question based on the description
3. DO NOT include constraints related to:
- JSON format
- Character or word count
- Keyword presence
- Capitalization, punctuation, or grammar
- Regex or other rule-based methods
4. If {primary_type} is mostly rule-verifiable or unsuitable for semantic evaluation, output:
No constraints
=== EXAMPLE (Valid Type: Style Constraints) ===
Response must use [placeholder] tone
Question: Is the tone consistently use [placeholder] tone?
=== EXAMPLE (Invalid Type: Length Constraints) ===
Output:
No constraints
=== YOUR TASK ===
Generate LLM-based constraints for: {primary_type}

Prompt Template for Template-Style Soft Constraints Generation

You are an expert in designing code-verifiable constraints for evaluating language model outputs.
=== TASK ===
Given:
1. An original constraint (natural language)
2. Its associated Python evaluation function
Your job is to generate multiple NEW constraints that:
- Are verifiable using Python code (reusing or modifying the original function)
- Follow the same underlying principle or logic as the original
- Focus on structural, behavioral, or surface properties that can be programmatically checked
=== REQUIREMENTS ===
For EACH new constraint, output:
- A natural language description of the new constraint
- A corresponding Python evaluation function that returns a boolean
Do NOT generate constraints that depend on:
- Semantic nuance
- Subjective interpretation
- Human judgment of style, tone, or meaning
Here is the given constraint and its corresponding verification functions:
Constraint: {constraint}
Function: {function}
Now generate several new constraints and their corresponding verification functions follow the above output format.

Prompt Template for Conflict Filtering

You are an expert in constraint validation and logical consistency checking.
Your task is to analyze the given set of constraints (evaluation functions and verification questions) and determine whether:
- They contain **conflicting requirements**
- They impose **impossible or mutually exclusive conditions**
- They have **hidden risks** that would make it extremely difficult or impossible for a language model to satisfy all constraints simultaneously
Functions:
{functions}
Questions:
{questions}
Output:
"yes" if there are conflicts or risks
"no" if the constraints are consistent and feasible
Do not output any explanation, only "yes" or "no"

Prompt Template for Query Generation

You are a helpful assistant that generates instructions and evaluation functions for response validation.
Given:
1. A persona description
2. Multiple Python evaluation functions (with placeholders for parameters)
3. A list of natural language verification questions
Your task is to:
- Generate an **instruction** that reflects the **persona** and includes two types of constraints:
- **Objective constraints**, which are clearly verifiable via the provided evaluation functions
- **Subjective qualities**, which are evaluable through the provided natural language verification questions
- Output only the **instruction**, followed by the **evaluation functions with concrete parameters**, and the **verification questions**, each wrapped in “‘python“‘ code blocks
- Only specify the parameters of the evaluate function; do not modify the function name and function body (e.g., avoid adding extra code like nltk.download)
- **Must contain** the "import" information such as "import re" or "import nltk" in the function body
Example Input:
Persona: You are a middle school science teacher helping students write clear experiment reports.
Functions:
“‘python
def evaluate_max_words(response, max_words):
"""Validate total word count <= max_words."""
import nltk
from nltktokenize import word_tokenize
words = word_tokenize(response)
return len(words) <= max_words
“‘
Questions:
“‘python
Is the originality in the response meaningful and contextually appropriate, rather than artificially imposed or distracting?
“‘
...
Example Output:
Instruction: Write a science experiment report suitable for middle school, with no more than 300 words total. Ensure the originality of the response is meaningful and fits the context naturally. The conclusion should clearly reflect the observations made in the experiment.
“‘python
def evaluate_max_words(response, max_words=300):
"""Validate total word count <= max_words."""
import nltk
from nltktokenize import word_tokenize
words = word_tokenize(response)
return len(words) <= max_words
“‘
...
Now generate the output based on the following persona, evaluation functions, and verification questions:
Persona: {persona}
Functions:
{functions}
Questions:
{questions}

Prompt Template for Response Generation

You are an expert tasked with answering the given query.
Please provide a clear and concise response directly, without introductory phrases such as 'What a great question', 'Here is the answer' or similar expressions.
Focus solely on addressing the query.
Now please answer the given query while strictly following its inside constraints.
[Query] {query}

remains computationally efficient without significantly impacting overall framework performance. Second, for SFT data, after reject sampling, the retention reaches approximately 40%. For RL data, where we only need to filter erroneous verification code and incomplete validation questions, retention exceeds 80%. These results demonstrate that DecIF maintains high retention rates while synthesizing accurate and diverse instruction-following data nearly from scratch. *Crucially, this efficient synthesis can be accomplished using smaller models, confirming DecIF's superior efficiency*.

D Prompt Templates

In this section, we list all the prompt templates used in DecIF in the above boxes.