TaCIE: Enhancing Instruction Comprehension in Large Language Models through Task-Centred Instruction Evolution

Jiuding Yang [1] **Shengyao Lu** [1] **Weidong Guo**[†] [2] **Xiangyang Li** [2]
Kaitong Yang [2] **Yu Xu** [2] **Di Niu** [1]
[1]University of Alberta
[2]Platform and Content Group, Tencent
[1]{jiuding,shengyao,dniu}@ualberta.ca
[2]{weidongguo,xiangyangli,kaitongyang,henrysxu}@tencent.com

Abstract

Large Language Models (LLMs) require precise alignment with complex instructions to optimize their performance in real-world applications. As the demand for refined instruction tuning data increases, traditional methods that evolve simple seed instructions often struggle to effectively enhance complexity or manage difficulty scaling across various domains. Our innovative approach, Task-Centered Instruction Evolution (TaCIE), addresses these shortcomings by redefining instruction evolution from merely evolving seed instructions to a more dynamic and comprehensive combination of elements. TaCIE starts by deconstructing complex instructions into their fundamental components. It then generates and integrates new elements with the original ones, reassembling them into more sophisticated instructions that progressively increase in difficulty, diversity, and complexity. Applied across multiple domains, LLMs fine-tuned with these evolved instructions have substantially outperformed those tuned with conventional methods, marking a significant advancement in instruction-based model fine-tuning.

1 Introduction

The rapid development of Large Language Models (LLMs) and their expanding real-world applications require closer alignment with complex human instructions to enhance performance across various tasks. This alignment demands high-quality instruction tuning data. However, manually crafting such instructions is impractical due to the time-intensive nature of the process and the tendency for these human-written instructions to remain simplistic (Xu et al., 2024), offering minimal benefits for tuning effectiveness(Kung et al., 2023).

To mitigate the high costs and challenges of manual instruction creation, researchers have developed automated synthesis methods using powerful LLMs to generate more sophisticated instructions from simpler ones. Notable among these are SELF-INSTRUCT by Wang et al. (2023a), which expands the range of instructions from a set of seed inputs, and EVOL-INSTRUCT by Xu et al. (2024), which refines instructions by enhancing either diversity or difficulty. Guo et al. (2024) also introduced *Instruction Fusion*, combining two distinct instructions to increase task complexity.

Despite their success in enhancing instruction quality and LLM performance, existing methods like EVOL-INSTRUCT and *Instruction Fusion* exhibit significant limitations. Firstly, EVOL-INSTRUCT struggles with managing difficulty increments effectively; prompts such as "add one more constraint" often lead to vague enhancements that do not genuinely increase the task's difficulty. For example, a depth evolving experiment with GPT-4o[*] showed that only one out of three attempts successfully intensified the instruction's complexity (Figure 1). Other attempts either replicated existing requirements or merely substituted terms for more complex equivalents, illustrating the challenges of uncontrolled difficulty scaling. Furthermore, Luo et al. (2024)'s application in code generation tasks excessively escalated difficulty, adding seven constraints in just four rounds.

Secondly, these methods fail to adequately address cross-domain tasks. Although EVOL-INSTRUCT was implemented in both math and code generation tasks, it focused primarily on increasing difficulty within the specific domain of the initial instruction, leading to a lack of diversity in task complexity. To mitigate this, (Guo et al., 2024) developed *Instruction Fusion* to combine elements from two seed instructions. However, this approach, limited to a single round of fusion in code generation, falls short of fully exploiting the potential for enhanced complexity.

[†] Corresponding author.

[*]https://platform.openai.com/docs/models

To address these deficiencies, we introduce Task-Centered Instruction Evolution (TaCIE), which employs advanced LLMs such as GPT-4o to dissect and reassemble instructional elements, targeting enhancements in both difficulty and complexity. TaCIE systematically decomposes instructions into background information, objectives, and constraints. This allows for precise modifications and fosters a more significant evolution of the instructions. By shifting the focus from evolving individual instructions to evolving their basic elements, TaCIE not only refines difficulty scaling, but also enables the integration of cross-domain elements. This approach significantly improves the complexity and applicability of the evolved instructions.

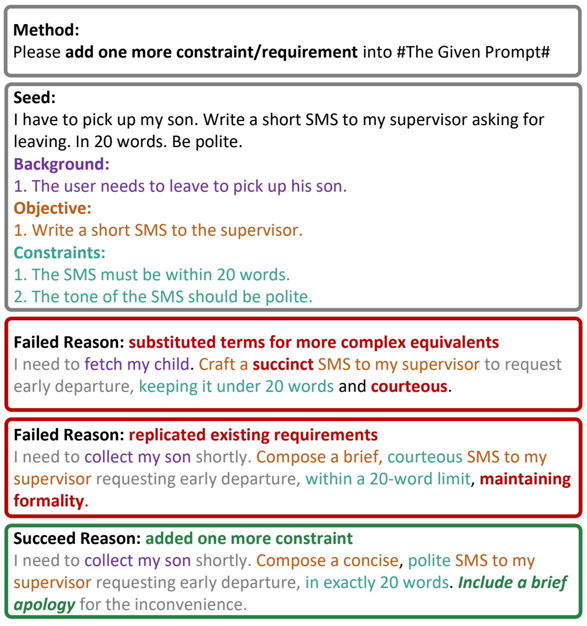
Empirical results show that LLMs fine-tuned with TaCIE-evolved instructions outperform those tuned with existing methods on diverse benchmarks, including MT-Bench (Zheng et al., 2023), AlpacaEval (Li et al., 2023), GSM8K (Cobbe et al., 2021) and HumanEval (Chen et al., 2021). This highlights TaCIE's ability to generate instructions that are more complex, nuanced, and broadly applicable across various domains. The key contributions of this work are:

- We introduce TaCIE, a task-centered instruction evolution method that decomposes seed instructions into three distinct elements and generates evolved instructions by modifying these elements. TaCIE effectively overcomes the challenges associated with difficulty scaling and cross-domain applicability, overcoming the limitation of existing instruction evolution methods such as EVOL-INSTRUCT and *Instruction Fusion*.

- Our extensive fine-tuning experiments across multiple domains demonstrate TaCIE's superior performance in instruction generation. To promote research collaboration, we have open-sourced model weights, training data, and source code, available at https://github.com/XpastaX/TaCIE.

2 Approach

In this section, we propose TaCIE, a novel and efficient task-centred solution for instruction evolution to overcome the limitations of existing methods. Before introducing TaCIE, we first discuss the existing instruction evolution method.



Method:
Please **add one more constraint/requirement** into #The Given Prompt#

Seed:
I have to pick up my son. Write a short SMS to my supervisor asking for leaving. In 20 words. Be polite.
Background:
1. The user needs to leave to pick up his son.
Objective:
1. Write a short SMS to the supervisor.
Constraints:
1. The SMS must be within 20 words.
2. The tone of the SMS should be polite.

Failed Reason: substituted terms for more complex equivalents
I need to fetch my child. Craft a **succinct** SMS to my supervisor to request early departure, keeping it under 20 words and **courteous**.

Failed Reason: replicated existing requirements
I need to collect my son shortly. Compose a brief, courteous SMS to my supervisor requesting early departure, within a 20-word limit, **maintaining formality**.

Succeed Reason: added one more constraint
I need to collect my son shortly. Compose a concise, polite SMS to my supervisor requesting early departure, in exactly 20 words. ***Include a brief apology*** for the inconvenience.

Figure 1: Real examples of applying EVOL-INSTRUCT using GPT-4.

2.1 Background

SELF-INSTRUCT enhanced LLMs by fine-tuning with diverse self-generated instructions, leading to the development of EVOL-INSTRUCT by Xu et al. (2024), which uses ChatGPT[†] to create more challenging and varied instructions from simple seed instructions. These seeds are evolved using five human-designed methods, significantly boosting LLM performance across various tasks. Recognizing limitations in task complexity, Guo et al. (2024) introduced *Instruction Fusion*, merging two seeds to enhance task complexity and performance, effectively complementing EVOL-INSTRUCT. Most recently, Zeng et al. (2024) advanced this by developing an active instruction evolution method that uses GPT to select the optimal evolution strategy for seed instructions, further enhancing the instruction evolution's impact on LLM performance.

However, the methods discussed above share two significant limitations: inadequate management of difficulty increments and insufficient consideration of cross-domain tasks.

Difficulty Increment Management is essential for effective instruction evolution. Current methods struggle with this aspect due to vague prompts provided to LLMs, which lack specific guidance for evolving instructions. This results in uncontrollable and unpredictable outcomes. For instance, Figure 1 demonstrates the evolution of a simple

[†]https://chatgpt.com

instruction using EVOL-INSTRUCT and GPT-4o. Of three attempts, only the last one successfully added a new constraint to the seed instruction. The previous attempts either replaced terms with more complex equivalents or repeated existing requirements. For example, the second attempt evolved an instruction for a formal SMS, which GPT-4o interpreted in a manner too similar to the original, highlighting the inefficiency of these methods and their limited effectiveness in enhancing LLM fine-tuning.

Cross-Domain Task Consideration is equally critical. Despite improvements from methods like *Instruction Fusion*, they fail to accommodate the complexity of cross-domain tasks. For instance, consider a task where an LLM is asked to develop an app function that automatically sends SMS messages in varying tones to selected contacts—this complexity far exceeds that of tasks focusing solely on message sending or SMS creation.

To address these challenges, we propose TaCIE, which is designed to effectively manage difficulty increments and cater to the demands of cross-domain tasks.

2.2 Instruction Decomposition

The most effective way to control incremental difficulty is to ensure that each new prompt introduces additional constraints or logical reasoning steps. To achieve this precise control over the evolution process, we draw inspiration from established decomposition methods in instructional design, as outlined in (Qin et al., 2024; Yang et al., 2024b). We employ GPT-4o to dissect seed instructions into three fundamental components: Background, Objectives, and Constraints. This allows for direct modifications in constraint and logic reasoning applied to these elements.

Figure 1 illustrates an example of this decomposition approach. The Background component encapsulates all relevant information necessary for the instruction, such as facts, motivations, and provided texts for tasks like summarization. The Objectives segment outlines the primary tasks derived from the seed instruction, for instance, composing an SMS as depicted in the example. Lastly, the Constraints section details specific requirements and limitations related to the tasks, including word count and formatting requirements.

Let $\mathbf{C} = \{c_i\}_{1\leq i\leq N}$ be the set of seed instructions, where c_i is the i-th instruction and N is the number of seeds. We define:

$$\mathbf{E} = \{e_i\}_{1\leq i\leq N} = \{\texttt{Decompose}(c_i)\}_{0\leq i\leq N}. \quad (1)$$

Here, e_i represents the decomposed elements of seed c_i, Decompose denotes the decomposition process, and $\mathbf{E}$ is the set of decomposed elements from the seed pool $\mathbf{C}$. A detailed prompt template for decomposition is introduced in the Appendix.

2.3 Task-Centred Instruction Evolution

We break down the original seeds and then iteratively apply two types of evolution: depth evolution and task fusion, as illustrated in Figure 2.

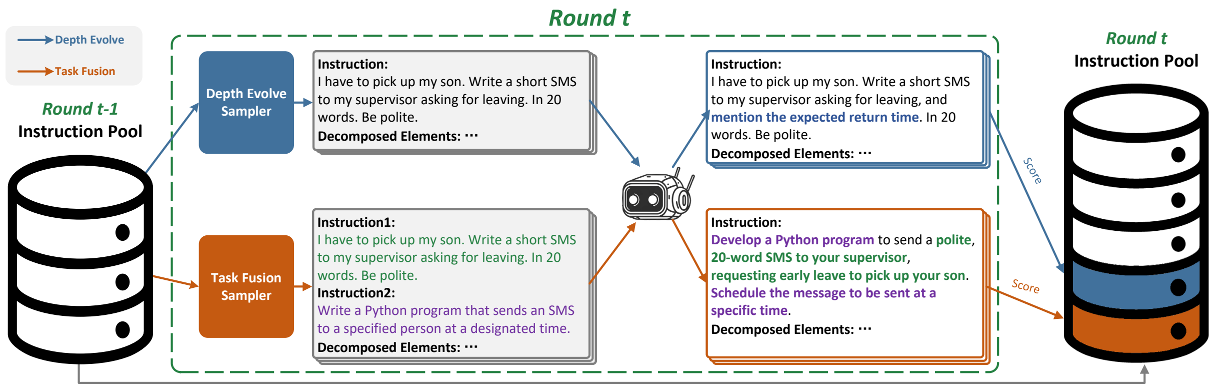
For depth evolution, we aims to increase the difficult of newly generated instructions. To manage the increment in difficulty, we have developed a prompt template. This template guides the evolver to add precisely one additional constraint or one extra background setting to the elements of the seed instruction, thereby enhancing either the difficulty or the logical reasoning required. For example, as shown in Figure 2, the depth evolution successfully added an extra constraint to the original instruction by requiring the mention of the expected return time in an SMS requesting leave.

For task fusion, our objective is to enhance the complexity of tasks in fused instructions, making them more informative. We instruct the evolver to merge all elements from each pair of seed instructions, as illustrated in Figure 2. The different colors in the orange box represent elements from the two seed instructions.

Despite the benefits these evolutionary methods offer to LLMs, selecting the right candidate instruction is crucial. A well-chosen seed can lead to evolved instructions of higher quality. To maximize the effectiveness of the proposed evolution methods, we have also developed a new candidate sampling technique. The entire process is segmented into the following stages:

Seed Collection. We commence by collating seed instructions from a diverse assortment of open-source datasets tailored to various specializations: Alpaca (Taori et al., 2023) and ShareGPT (Chiang et al., 2023) for general instruction comprehension and execution; GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) for mathematical problem-solving; and CodeAlpaca (Luo et al., 2024) for coding tasks. Initially, we employ Sentence Transformers[‡] to compute embeddings for all

[‡]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2



Instruction:
I have to pick up my son. Write a short SMS to my supervisor asking for leaving. In 20 words. Be polite.
Decomposed Elements: ···

Instruction1:
I have to pick up my son. Write a short SMS to my supervisor asking for leaving. In 20 words. Be polite.
Instruction2:
Write a Python program that sends an SMS to a specified person at a designated time.
Decomposed Elements: ···

Instruction:
I have to pick up my son. Write a short SMS to my supervisor asking for leaving, and mention the expected return time. In 20 words. Be polite.
Decomposed Elements: ···

Instruction:
Develop a Python program to send a polite, 20-word SMS to your supervisor, requesting early leave to pick up your son. Schedule the message to be sent at a specific time.
Decomposed Elements: ···

Figure 2: An illustration of TaCIE during the t-th round of evolution.

instructions within these datasets. Using the Elbow method, we identify the optimal clustering configuration for each source. After clustering, we randomly select one seed instruction from each cluster. To broaden the diversity of our initial seed pool, we engage GPT-4o to generate three additional instructions that each have a different objective from the original seed. This approach enriches our collection of instructions, creating a robust foundational seed pool for the TaCIE framework, detailed further in the Appendix.

Sample Scoring. The effectiveness of instruction tuning for LLMs relies on the quality of instruction-response pairs, rather than their quantity. As Zhou et al. (2024) have noted, instruction sets of higher quality confer more significant benefits to LLMs than those merely larger in volume. To curate such beneficial instructions, following Kung et al. (2023), we employ uncertainty to filter out less beneficial instructions.

According to Kung et al. (2023), it has been established that LLMs benefit more from fine-tuning with informative and novel instructions. Such instructions exhibit a significant alteration in the probability of their original responses when subjected to minor perturbations, like the random omission of a certain percentage of words. This alteration is considerably less pronounced in instructions that are less informative (low uncertainty), which are either "easy" (high response probability) or "difficult" (low response probability) to the LLMs.

In the TaCIE framework, candidate sampling for the evolution process is guided by calculating uncertainty scores, which help identify instructions with potential for greater informativeness. The depth evolution strategy primarily targets seeds with high uncertainty, aiming to evolve these into more difficult instructions by incorporating additional logical reasoning steps or constraints. In contrast, task fusion merges information from two less informative seeds (low uncertainty) to create a single, more comprehensive instruction. This approach ensures that newly generated instructions are enriched with useful content without becoming overly complex, thereby enhancing their practical applicability in training LLMs. Further details on this process are elaborated in the Appendix.

Specifically, we define uncertainty u_i as the score of the i-th instruction c_i and use the following `Score` function to calculate it:

$$u_i = \texttt{Score}(c_i, r_i) = \frac{1}{N_U} \sum_{j=1}^{N_U} (|q_i - \bar{q}_i^j|), \quad (2)$$

where N_U is the number of perturbation for each instruction, and r_i represents the response to the instruction c_i. q_i is the response probability of c_i, and $\bar{q}_i^j$ is the response probability of the j-th perturbation of c_i.

The response probabilities are defined as follows:

$$q_i = \mathcal{P}(r_i | c_i, \mathbf{W}), \quad (3)$$

and

$$\bar{q}_i^j = \mathcal{P}(r_i | \bar{c}_i^j, \mathbf{W}). \quad (4)$$

Here, q_i and $\bar{q}_i^j$ represent the probabilities of a response r_i, conditioned on the original instruction c_i and the perturbed instruction $\bar{c}_i^j$, respectively. These probabilities are calculated using the model weights $\mathbf{W}$, reflecting the model's evaluation of the responses based on the provided instructions.

Candidate Sampling. As mentioned by Guo et al. (2024), difficulty gradient is also an important key for better fine-tuning performance. To balance informative, "easy", and "difficult" instructions, we designed a sampler which samples the candidates for the depth evolution and task fusion according to different weighting approach. Denote the target evolution amount to be M_e and M_t, for depth evolution, we directly use the uncertainty to weight each seed, and defined the sample probability of each instruction as:

$$p_i^{\mathsf{depth}} = \frac{u_i}{\sum_{k=1}^{N} u_k}, \tag{5}$$

and we sample M_e instructions:

$$\mathbf{C}^{\mathsf{depth}} \sim \texttt{Multinomial}(M_e, \{p_i^{\mathsf{depth}}\}_{i=1}^{N}), \tag{6}$$

For task fusion, we use the following weighting function:

$$p_i^{\mathsf{fuse}} = \frac{s_i}{\sum_{k=1}^{N} s_k} \tag{7}$$

where

$$s_i = \frac{1}{(n_{c_i}+1) \times n_{\mathtt{obj}_i} \times n_{\mathtt{root}_i} \times u_i}. \tag{8}$$

Here s_i is the punished uncertainty of the seed instruction c_i. For each instruction, we punish the uncertainty with three factors. n_{c_i} represent the frequency of instruction c_i being used as the seed for task fusion; $n_{\mathtt{obj}_i}$ is the number of objectives of the instruction, which will increase if it is fused instruction last round; $n_{\mathtt{root}_i}$ is the frequency of the root domain that instruction lies in, such as coding, math, etc. The sampling process is shown in the Algorithm 1.

We categorize task fusion into two types: in-domain fusion and cross-domain fusion. In-domain fusion integrates tasks within the same domain, whereas cross-domain fusion combines tasks from different domains to generate more complex outcomes. The process begins by sampling M_f initial candidates, denoted as C_a, which form the first set of seeds. Subsequently, we sample another set of M_f candidates, C_b, and pair each candidate with those in C_a based on the differences in their domains. The sampling of C_b and its pairing process are repeated until we achieve the desired number of pairs for both in-domain and cross-domain fusions.

For simplification, in a cross-domain fused instruction involving a pair (c_a, c_b), we designate the root domain of the instruction as the root domain of c_a.

Evolution with Evolver. In the round t of evolution, based on our designed weighting functions above, we first sample M_d candidates for the depth evolution, and M_f candidate pairs from the seed

Algorithm 1: Task Fusion Sampling

```
Input: Seed pool $\mathbf{C}$ which weights; Target number of
  task fusion $M_f$.
Output: Candidate pairs.
Initialize $\mathtt{pairs_{in}}, \mathtt{pairs_{cross}} \leftarrow \emptyset$, $\mathtt{idx} \leftarrow 0$
Let $\mathbf{C}_a \sim \mathtt{Multinomial}(M_f, \{p_i^{\mathsf{fuse}}\}_{i=1}^{N})$
while $\mathtt{pairs_{in}} < \frac{M_f}{2}$ or $\mathtt{pairs_{cross}} < \frac{M_f}{2}$ do
    $\mathtt{remain} \leftarrow M_f - (|\mathtt{pairs_{in}}| + |\mathtt{pairs_{cross}}|)$
    $\mathbf{C}_b \sim \mathtt{Multinomial}(\mathtt{remain}, \{p_i^{\mathsf{fuse}}\}_{i=1}^{N})$
    for each $c_b \in \mathbf{C}_b$ do
        $c_a \leftarrow \mathbf{F}_a[\mathtt{idx}]$
        if $\mathtt{domain}(c_a) == \mathtt{domain}(c_b)$ then
            if $|\mathtt{pairs_{in}}| < \frac{M_f}{2}$ then
                $\mathtt{pairs_{in}} \cup \{(c_a, c_b)\}$
                $\mathtt{idx} \leftarrow \mathtt{idx} + 1$
            end
        end
        else
            if $|\mathtt{pairs_{cross}}| < \frac{M_f}{2}$ then
                $\mathtt{pairs_{cross}} \cup \{(c_a, c_b)\}$
                $\mathtt{idx} \leftarrow \mathtt{idx} + 1$
            end
        end
    end
end
return $\mathtt{pairs_{in}} \cup \mathtt{pairs_{cross}}$
```

pool for the i-th round $\mathbf{C}^t$ for the task fusion, which are:

$$\mathbf{C}^{t,\mathsf{depth}} = \{c_i^{t,\mathsf{depth}}\}_{1 \le i \le M_d},$$
$$\mathbf{C}^{t,\mathsf{fuse}} = \{(c_{i,a}^{t,\mathsf{fuse}}, c_{i,b}^{t,\mathsf{fuse}})\}_{1 \le i \le M_f}.$$

Next, we prompt evolver LLM to perform the two kinds of evolution and obtained the new instruction sets $\mathbf{D}^t = \{d_i^t\}_{1 \le i \le M_d}$ and $\mathbf{F}^t = \{f_i^t\}_{1 \le i \le M_f}$ correspondingly, where

$$d_i^t = \mathtt{Evol}(c_i^{t,\mathsf{depth}}, e_i^{t,\mathsf{depth}}, \mathsf{P}^{\mathsf{depth}}),$$
$$f_i^t = \mathtt{Evol}((c_{i,a}^{t,\mathsf{fuse}}, e_{i,a}^{t,\mathsf{fuse}}, c_{i,b}^{t,\mathsf{fuse}}, e_{i,b}^{t,\mathsf{fuse}}), \mathsf{P}^{\mathsf{fuse}}).$$

Here $\mathsf{P}^{\mathsf{depth}}$ and $\mathsf{P}^{\mathsf{fuse}}$ are the prompt template designed for the evolution, and it detailed in Appendix.

After that, we merge them in to the previous seed pool to form the new set of candidates $\mathbf{C}^{t+1} = \mathbf{C}^t + \mathbf{D}^t + \mathbf{F}^t$, and update the weight of all samples according to their scores and evolution history for the next round.

2.4 Data Statistic

Figure 3 presents detailed statistics from our evolved instruction pool. We sampled a total of 12,000 seeds from multiple sources: 3,000 each from ShareGPT, Alpaca, and Code Alpaca, along with 1,500 from both the MATH and GSM8K training sets. Using these seeds, we prompted GPT-4o

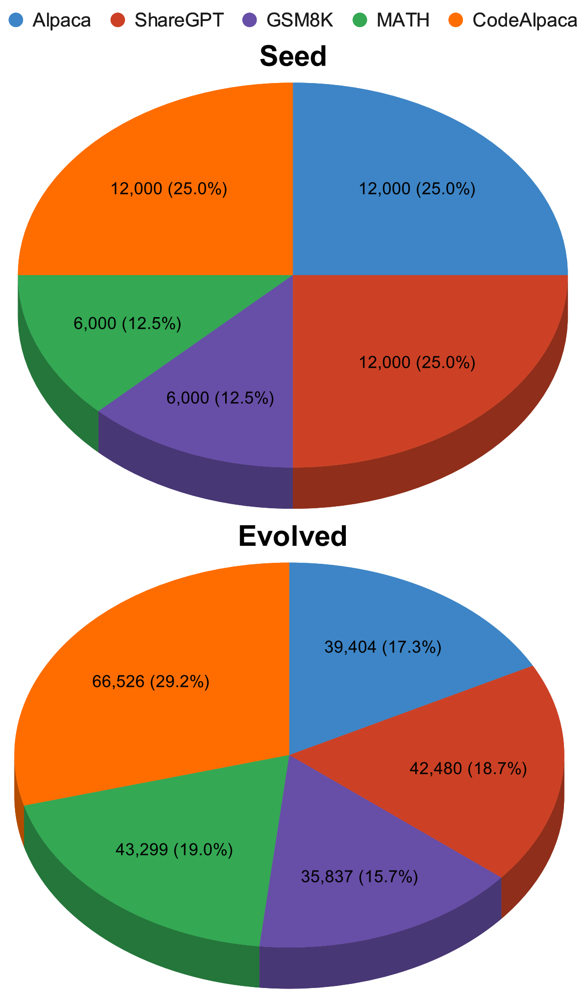


Figure 3: The domain distribution. Note each fused instruction contributes to multiple domains due to objectives from two seeds.

to generate an additional 36,000 variants, aiming to diversify the seed pool with varied objectives. Over six evolutionary rounds—excluding samples that failed in evolution or were unrecognized by our scripts—we applied two distinct evolutionary methods, ultimately producing 143,917 viable samples out of 144,000 attempts. This yields a success rate of over 99.94%, significantly surpassing the performance of EVOL-INSTRUCT. During the process, we utilized Llama-3-8B-Instruct[§] to evaluate the uncertainty of each instruction.

Additionally, the figure illustrates frequency statistics across different domains. The scoring language model identified mathematics and coding problems as particularly "informative"—a finding consistent with expectations, given that these categories often require robust logical reasoning and exhibit considerable variability with minor instructional modifications. This shift in domain distribution has effectively improved the balance of our data mix, enhancing instruction tuning. Subsequent experiments confirm that this rebalancing has notably improved performance across all evaluated metrics.

Figure 4 depicts the distribution of rounds for each evolutionary method. From this visualization, it is clear that TaCIE, in contrast to full-size evolution—which processes all candidates in each round and reaches 144,000 instructions in just three rounds—selects more informative seeds for evolution. This approach enables the generation of informative instructions over a higher number of evolutionary rounds within the same total number of generated instructions.

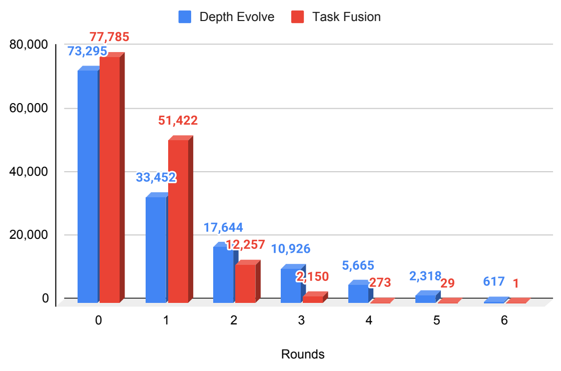


Figure 4: The distribution of evolution rounds.

3 Experiments

3.1 Experimental Settings

To effectively showcase the capabilities of TaCIE, we selected a diverse range of baselines, encompassing both general-purpose chatting LLMs and domain-specific LLMs. Our primary comparisons are with EVOL-INSTRUCT, *Auto Evol-Instruct*, and *Instruction Fusion*. These three methods extensively utilize instruction evolution techniques, making them directly comparable to our approach.

We assess performance across four dimensions. To evaluate general instruction comprehension, we utilize MT-Bench (Zheng et al., 2023). IFEval (Zhou et al., 2023) is employed to test instruction-following capabilities. The GSM8K test set (Cobbe et al., 2021) measures mathematical proficiency, and HumanEval (Chen et al., 2021) gauges coding skills.

We utilized GPT-4o both as the evolver and the

[§]https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct response generator for all instructions. Our foundational model for these experiments was LLaMA-3-8B (Dubey et al., 2024), augmented with LLaMA-3-8B-Instruct for scoring purposes. Additionally, we fine-tuned Mistral-7B-v0.1 (Jiang et al., 2023) and Qwen2-7B (Yang et al., 2024a) to evaluate the applicability of data generated with a dedicated scorer across different models. We also included performance metrics from their advanced official chat LLMs. However, it's important to note that their fine-tuning protocols often incorporate techniques beyond instruction tuning, such as Direct Preference Optimization (Rafailov et al., 2024). Therefore, their results are provided for reference only.

Consistent with existing literature, all our experiments were conducted using a batch size of 128 and a learning rate of 5×10^{-6}. Each LLM was trained over four epochs using bfloat16 precision on four Nvidia A100 80G GPUs. For these processes, we utilized resources from LLaMA Factory (Zheng et al., 2024) and integrated DeepSpeed[¶] with zero-stage 2 optimization.

3.2 Result Analysis

Table 1 shows TaCIE's impact on LLaMA-3 across four benchmarks. We compared the performance of LLMs fine-tuned on 48,000 instructions from both evolved and seed datasets. The results demonstrate substantial improvements in instruction following, math, and coding tasks with the evolved instructions, while maintaining competitive performance on MT-bench. Using LLaMA-3-Instruct as the scorer, TaCIE effectively identifies and samples informative candidates, significantly enhancing task performance in these domains. Despite only a minor performance increment on MT-bench due to its multi-round chatting requirements—a feature not covered in our datasets—the overall gains from the evolved instructions outweigh this limitation. The domain shift focuses on complex tasks over multi-round chatting (Section 2.4), leading to less performance boost but considerable benefits across other domains, validating the evolution approach.

To justify the transferability of the evolved instructions (candidates sampled with the LLaMA-3 based scorer can also benefits other base LLMs), we use them to further fine-tune Mistral-7B-v0.1 and Qwen2-7B, which also show significant improvements all most domains. Here the beneftis

Model	Para.	Data	MT-Bench	IFEval	GSM8K	HumanEval
			Close-Source Model			
GPT-4	-	-	8.99	85.37	92	84.1
GPT-3.5	-	-	7.9	-	80.8	73.2
			Open-Source General Model			
WizardLM	13B	70k	6.35	18.5	-	24
Vicuna-v1.3	13B	-	6.57	33.44	10.77	-
LLaMA-2-Chat	13B	-	6.65	39.85	15.24	32.3
Mistral-instruct-v0.1	7B	-	6.84	42.51	14.25	31.1
LLaMA-3-Instruct	8B	-	8.05	73.01	79.6	62.2
Qwen2-Instruct	7B	-	8.41	55.08	82.3	79.9
			TaCIE General Model			
LLaMA-3-seed	8B	48k	7.14	38.45	64.82	43.9
TaCIE-LLaMA-3	8B	48k	**7.18**	39.74	68	51.2
TaCIE-LLaMA-3	8B	144k	6.63	**42.7**	**72.25**	**54.9**
Mistral-seed	7B	48k	6.75	27.73	53.3	21.3
TaCIE-Mistral	7B	48k	**6.99**	18.11	58.38	36.6
TaCIE-Mistral	7B	144k	6.73	**30.13**	**66.64**	**43.3**
Qwen2-seed	7B	48k	7.62	39	82.56	72.6
TaCIE-Qwen2	7B	48k	7.85	**43.99**	**83.32**	69.5
TaCIE-Qwen2	7B	144k	**7.87**	41.4	81.65	**76.8**
TaCIE-LLaMA-3-id	8B	10k	6.9	31.05	74	**51.8**
TaCIE-LLaMA-3-cd	8B	10k	**6.96**	**34.18**	**74.45**	**51.8**
			Open-Source Task-Specific			
AIE-ShareGPT	7B	10k	7.51	-	-	-
WizardMath	7B	96k	-	-	54.9	-
MetaMath	7B	395k	-	-	66.51	-
AIE-GSM8K	7B	7k	-	-	70.74	-
WizardCoder	15B		-	-	-	
CodeLlama-Instruct	13B	-	-	-	-	42.7
AIE-CodeAlpaca	13B	20k	-	-	-	65.85
IF-20k	13B	20k	-	-	-	**67.7**
			TaCIE Task-Specific			
TaCIE-ShareGPT	7B	10k	**7.53**	-	-	-
TaCIE-GSM8K	7B	7k	-	-	**73.46**	-
TaCIE-CodeAlpaca	13B	20k	-	-	-	67.1

Table 1: Experimental results of TaCIE.

for Qwen2 model is less than that on the other two base LLMs, because Qwen2-7B have better performance than LLaMA-3-8B[‖], so what LLaMA-3-8B find informative may not be true for Qwen2-7B. However, it still outperforms the baseline fine-tuned with seeds.

Besides the general purpose LLMs, we also conduct experiment fine-tuning base models using domain-specific instructions. We mainly compare our performance with AIE proposed by Zeng et al. (2024). For fair comparison, we use the same base LLMs and sample the same amount of evolved instructions that only contains single-domain information among all of its evolution history. For ShareGPT, we mix 7,000 evolved instructions with 3,000 original samples to cover multi-round requirements of MT-Bench.

As shown in the domain-specific part in Table 1, we outperform AIE on all three domains, especially on math and coding, which requires better logic for answering. For *Instruction Fusion*, we sampled 20,000 samples from their 110,000 evolved coding instructions for baseline fine-tuning (IF-20k). Although they has evolved more complex instructions with higher diversity due to the large amount of in-

[¶]https://github.com/microsoft/DeepSpeed

[‖]https://qwenlm.github.io/blog/qwen2/

structions, we still achieved 67.1% on HumanEval, which is comparable to their 67.7%.

In conclusion, TaCIE's evolved instructions significantly boost base LLM performance across tasks, particularly in complex areas like instruction comprehension, math, and coding. These enhancements persist even when applied to LLMs with varying architectures, such as Mistral-7B-v0.1 and Qwen2-7B, demonstrating their broad applicability and transferability. These findings confirm the effectiveness of the TaCIE evolution approach in enhancing instruction sets for diverse LLM applications.

3.3 Ablation Study

To further demonstrate task fusion's effectiveness, we fine-tuned LLaMA-3-8B with 10,000 single-domain and 10,000 cross-domain-only evolved instructions (excluding seeds). The results, shown as TaCIE-LLaMA-3-id and TaCIE-LLaMA-3-cd in Table 1, reveal that models fine-tuned with cross-domain instructions outperform those fine-tuned with in-domain instructions. This supports the notion that fusing objectives from different domains introduces greater complexity and information, enhancing LLM learning.

Additionally, to evaluate the scalability of TaCIE, we fine-tuned LLaMA-3-8B with various subsets of instructions randomly sampled from our instruction pool. The results, depicted in Figure 5, indicate that performance peaks when utilizing the entire instruction pool. This observation suggests that further enhancements in performance could be achieved through additional rounds of instruction pool evolution. We also extended this fine-tuning process to two other base LLMs using the complete set of 143,917 instructions (Table 1). This approach demonstrated significant improvements across multiple metrics when compared to using only 48,000 instructions.

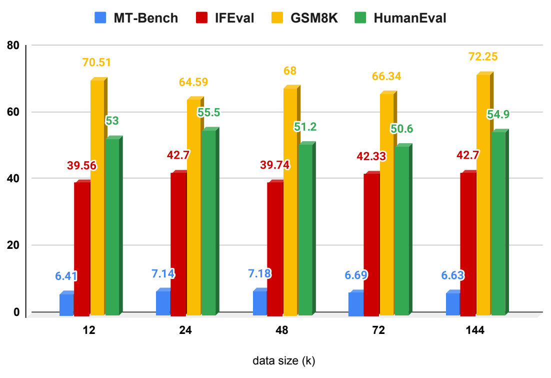


Figure 5: The performance scaling of TaCIE.

4 Related Work

4.1 Instruction Tuning

Instruction Tuning (Wei et al., 2022) is a pivotal method for aligning Large Language Models (LLMs) with human instructions, enhancing their applicability to real-world scenarios (Zhang et al., 2023). This approach aims to improve the zero-shot capabilities of well-trained LLMs, enabling them to perform tasks guided solely by natural human instructions. It allows LLMs to cater to a diverse array of general requests (Wang et al., 2023b). However, the effectiveness of Instruction Tuning heavily depends on the availability of high-quality data to optimize both performance and interaction quality (Zhou et al., 2024; Zhao et al., 2024).

For enhancing and proofing high-quality instructions, Zhou et al. (2024) introduced LIMA, an LLM trained on just 1,000 high-quality instances. This model demonstrated that smaller datasets of superior quality can be more beneficial than larger, less curated ones. Additionally, Zhao et al. (2024) found that detailed, lengthy instructional responses also enhance LLM performance, underscoring the significance of data quality.

Conversely, Kung et al. (2023) explored the concept of task uncertainty by examining how minor perturbations in instructions affect response probabilities. They proposed that instructions whose perturbations can cause significant shifts in response probabilities are particularly informative and novel for LLMs, thus aiding in better alignment. This approach has inspired our methodology for the design of seed sampling.

These studies collectively highlight the critical role of high-quality instructions and responses in the efficacy of LLMs. Nonetheless, the reliance on manually crafted instructions or templates constrains the diversity, quantity, and creativity of the data available.

4.2 Instruction Evolution

In response to the growing demand for high-quality data, Wang et al. (2023a) developed SELF-INSTRUCT, a strategy that utilizes LLMs for both generating data and tuning instructions. This method produces enhanced synthetic instructions and responses. Building on this concept, Xu et al. (2024) introduced EVOL-INSTRUCT, which evolves initial simple instructions into more challenging or varied forms by leveraging sophisticated LLMs (e.g., ChatGPT) and a designed evolution methodology. Further expanding on this, Zeng et al. (2024) proposed an advanced framework that allows LLMs to autonomously determine the most effective evolution strategy for a given set of seed instructions.

Guo et al. (2024) focused on task complexity by developing *Instruction Fusion*, which integrates two simple tasks into a single, more complex challenge, thereby enhancing performance. This method has inspired further exploration into cross-domain task fusion within our proposed approach.

5 Conclusion

In this paper, we introduce TaCIE, a novel method for evolving instruction that enhances difficulty management and promotes cross-domain complexity. TaCIE employs a decomposition approach to break down seed instructions into three fundamental elements. This transformation shifts the evolution process to the elemental level, allowing for targeted modifications that culminate in the regeneration of advanced instructions. Experimental results underscore TaCIE's efficacy, demonstrating significant performance improvements across a variety of base LLMs compared to previous methods.

Limitations

Our work focus on task-centered evolution, making them more difficult and complex in controllable manner, thus the multi-turn chatting is not considered in our evolving procedure. Future work could easily built evolution method for multi-turn conversation based on our proposed method.

The total cost of our experimental process was approximately 2,000 USD, largely driven by our reliance on GPT-4o for augmentation and evaluation—a common challenge in LLM-related research. However, the expenses associated with using such APIs are decreasing due to rapid advancements in LLM technologies, making these tools more affordable and accessible.

Ethics Statement

Our data collection relies on publicly released datasets. The augmented data were generated using GPT-4o, whose outputs are already carefully monitored, ensuring that no privacy-sensitive or confidential information was included.

References

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, 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, Josh 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.

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

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. *arXiv preprint arXiv:2110.14168*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Weidong Guo, Jiuding Yang, Kaitong Yang, Xiangyang Li, Zhuwei Rao, Yu Xu, and Di Niu. 2024. Instruction fusion: Advancing prompt evolution through hybridization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3883–3893, Bangkok, Thailand. Association for Computational Linguistics.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Po-Nien Kung, Fan Yin, Di Wu, Kai-Wei Chang, and Nanyun Peng. 2023. Active instruction tuning:

Improving cross-task generalization by training on prompt sensitive tasks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1813–1829.

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

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*.

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

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

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.

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. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508.

Yufei Wang, Wanjun Zhong, Liangyou Li, Fei Mi, Xingshan Zeng, Wenyong Huang, Lifeng Shang, Xin Jiang, and Qun Liu. 2023b. Aligning large language models with human: A survey. *arXiv preprint arXiv:2307.12966*.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. Finetuned language models are zero-shot learners. *Preprint*, arXiv:2109.01652.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. 2024. WizardLM: Empowering large pre-trained language models to follow complex instructions. In *The Twelfth International Conference on Learning Representations*.

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

Jiuding Yang, Weidong Guo, Kaitong Yang, Xiangyang Li, Zhuwei Rao, Yu Xu, and Di Niu. 2024b. Optimizing and testing instruction-following: Analyzing the impact of fine-grained instruction variants on instruction-tuned llms. *arXiv preprint arXiv:2406.11301*.

Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. 2024. Automatic instruction evolving for large language models. *arXiv preprint arXiv:2406.00770*.

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

Hao Zhao, Maksym Andriushchenko, Francesco Croce, and Nicolas Flammarion. 2024. Long is more for alignment: A simple but tough-to-beat baseline for instruction fine-tuning. *arXiv preprint arXiv:2402.04833*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

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.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36.

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

A Data Analysis

A.1 Diversification

As detailed in Section 2, we utilize the Elbow method to optimally cluster each data source and randomly select one instruction from these clusters as the initial seeds. These seeds are then used to prompt GPT-4o to generate a diverse set of 48,000 instructions, forming our initial instruction pool (Round 0). Figure 6 shows a 2D projection of both

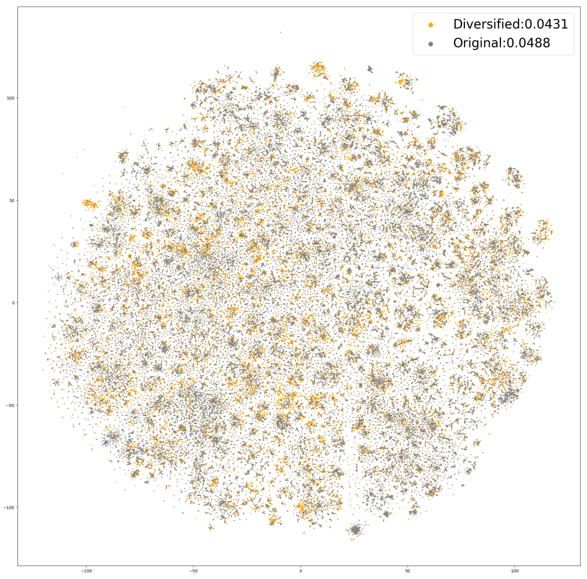


Figure 6: The 2D projection of original data source and diversified seed pool.

the randomly sampled instructions from the original data sources and our diversified seed pool. It is evident that the diversified seeds cover more of the previously blank regions compared to the original data.

To quantify this increase in diversity, we employ the variance calculation method used in *Instruction Fusion* by Guo et al. (2024):

$$U = \frac{1}{N}\Sigma_{i=1}^{N}(d_i - \mu)^2, \tag{9}$$

where

$$d_i = ||(e_i, e_i^{\text{NN}})|| \tag{10}$$

and

$$\mu = \frac{1}{N}\Sigma_{i=1}^{N} d_i. \tag{11}$$

In this formula, U represents the uniformity of the distribution, d_i denotes the Euclidean distance between the semantic embedding e_i of a seed and the embedding e_i^{NN} of its nearest neighbor, and μ is the average Euclidean distance across all seeds. This calculation of variance in nearest neighbor distances provides a measure of the instruction pool's diversity. A lower variance signifies a more uniform distribution of data points, indicative of greater diversity. As depicted in the accompanying figure, the variance for the diversified seed pool stands at 0.0431, which is approximately 12% lower than that of the original data source, confirming an enhancement in diversity.

A.2 Uncertainty Shift

To examine the impact of depth evolution and task fusion on instruction uncertainty, we generated 24,000 depth-evolved instructions and 12,000 task-fused instructions, both in-domain and cross-domain. Figure 7-9 show 2D projections of the original and evolved instructions. From these figures, it is apparent that depth evolution does not significantly alter the uncertainty of instructions. In contrast, both in-domain and cross-domain task fusion lead to an increase in uncertainty, from 0.07 to 0.09. This indicates that task fusion, through the merging of two seed instructions, produces more informative content, thereby enhancing the overall uncertainty within the dataset. However, merging highly informative instructions could potentially overload the learning process of LLMs due to increased complexity. To mitigate this, during the sampling process, we prioritize less informative seeds, balancing the generation of enriched instructions while ensuring they remain tractable for learning enhancements.

B Prompt Templates

In this section, we introduce all prompt templates we designed and use in TaCIE.

B.1 Instruction Decomposition

To better decompose each instruction, we provided GPT-4o with two examples during the decomposition process.

Given a prompt, your task is to:
*1.**Extract Backgrounds Settings:** Identify and list the backgrounds of the prompt, such as facts, and motivations. If the prompt provides extra information such as code to debug, passage to polish or summarize, directly include them in this section, do not summarize them. Do not include Objectives or Requirements. If no Backgrounds Settings is given in the prompt, output 'N/A' in the **Extract Background Settings:** section.*
*2.**Extract Objective:** List the core task of the prompt.*
*3.**Extract Constraints:** List all specific requirements or constraints for the objectives. If no Constraints is given in the prompt, output 'N/A' in the **Extract Constraints:** section.*

—

Given Prompt:

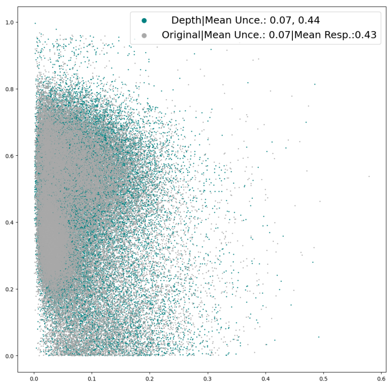


Figure 7: Depth Evolution

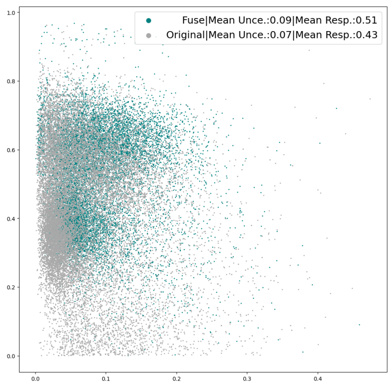


Figure 8: In-domain Task Fusion

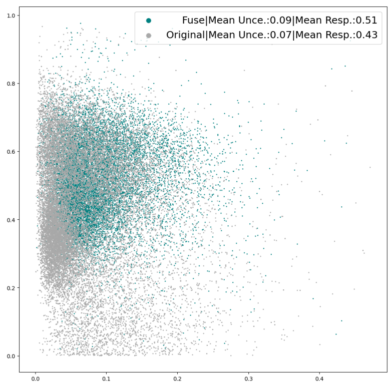


Figure 9: Cross-domain Task Fusion

I have to pick up my son. Write a short SMS to my supervisor asking for leaving. In 20 words. Be polite.
Extract Background Settings:
1.The user needs to pick up his son.

Extract Objectives:
1.Write an SMS.

Extract Constraints:
1.The SMS should be short.
2.The SMS should ask the supervisor's permission to leave.
3.The SMS should be polite.
4.The SMS should not exceed 20 words.

—

Given Prompt:
Debug my python code. a=100
b=1
c=0
*print(d=a*b/c)*

Extract Background Settings:
1.Code to debug:
*```python*
a=100
b=1
c=0
*print(d=a*b/c)*
*```*

Extract Objectives:
1.Debug the given Python code.

Extract Constraints:
N/A

—

***Given Prompt**:*
At 30, Anika is 4/3 the age of Maddie. What would be their average age in 15 years?

Extract Background Settings:
1.age of Anika = 30
2.age of Anika = 4/3 x age of Maddie

Extract Objectives:
1.Calculate the average age of Anika and Maddie in 15 years.

Extract Constraints:
N/A

—

Given Prompt:
{{ prompt }}

B.2 Diversification

During the diversification of the original seed, we use the following prompt to let GPT-4o generate another three different instructions for a given seed.
For the provided prompt, we detail the following elements:

*1.**Background Settings:** This section presents the background information relevant to the prompt, including pertinent facts and motivations. If the prompt lacks background settings, this section will be labeled as 'N/A'.*
*2.**Objectives:** This section outlines the main tasks associated with the prompt.*

*3.**Constraints:** This lists any specific requirements or limitations tied to the objectives. If there are no constraints, this section will be labeled as 'N/A'.*

Given Prompt:
{{ prompt }}

{{ extracted }}

Based on the information provided, your task is to:

*1.Develop ten new objectives to replace the original **Objectives** of the prompt. Each new objective should be diverse and maintain the same level of difficulty as the original.*
*2.For each new objective, craft a corresponding prompt that mimics the tone and style of the original prompt. Ensure to vary the **Background Settings** and **Constraints** while making sure each prompt is reasonable and answerable.*

Format each new objective and prompt as follows,do no provide corresponding background, objectives, and constraints:

New Objective 1:
[Describe the new diverse objective.]

New Prompt 1:
[Present the new prompt based on the objective.]
...

B.3 Depth Evolution

Based on a prompt's existing background, objectives, and constraints, increase its difficulty using ONLY one of the following methods: 1.If the prompt primarily involves reasoning, such as solving a mathematical problem, enhance its complexity by introducing an additional background element. Also, modify the existing background elements to ensure the task remains logical and solvable.
2.Otherwise, introduce one additional reasonable constraint to ONLY one of the objectives of the given prompt to increase its difficulty.

*Please respond using the format provided in the examples below. You can only change either the **Background Settings:** or the **Constraints:**. Do not change both.*

*Ensure your response contains the following four sections even if they are empty: **Prompt:**, **Background Settings:**, **Objective:** and **Constraints:***
—

### Original

Prompt:
At 30, Anika is 4/3 the age of Maddie. What would be their average age in 15 years?

Background Settings:
1.age of Anika = 30
2.age of Anika = 4/3 x age of Maddie

Objectives:
1.Calculate the average age of Anika and Maddie after 15 years

Constraints:
N/A

### Rewritten

Prompt:
At 30, Anika's age is twice the age of Adam, and Maddie's age is the average of Anika's and Adam's ages. What would be the average age of Anika and Maddie in 15 years?

Background Settings:
1.age of Anika = 30
2.age of Anika = 2 x age of Adam
3.age of Maddie = (age of Anika + age of Adam) / 2

Objectives:
1.Calculate the average age of Anika and Maddie after 15 years

Constraints:
N/A

—

### Original

Prompt:
Could you write me an android application that has a login page and can connect to a server? Please also list all prior knowledge I need to know

to understand your code.

Background Settings:
N/A

Objectives:
1.Write an Android application.
2.List all prior knowledge that is required to understand the code.

Constraints:
1.Include a login page in the application.
2.Enable the application to connect to a server.

### Rewritten

Prompt:
Could you write me an android application that has a login page, can connect to a server, and encrypts all communications with the server. Please also list all prior knowledge I need to know to understand your code.

Background Settings:
N/A

Constraints:
1.Include a login page in the application.
2.Enable the application to connect to a server.
3.Encrypt all communications with the server.

—

### Original

Prompt:
{{ prompt }}

{{ extracted }}

B.4 Task Fusion

*Based on the prompt's existing background, objectives, and constraints, your task is to act as a Prompt Fusion Specialist. Your target is to fuse **Given Prompt A** and **Given Prompt B** into a single, cohesive **Fused Prompt**, following the two steps below:*
*1.Merge the elements in the background, objectives, and constraints of both **Given Prompt A** and **Given Prompt B** respectively, make sure the objectives are dependent to each other and solvable. Do not compress multiple elements into a single one.*
*2.Based on the integrated background, objectives, and constraints, fuse the **Given Prompt A** and **Given Prompt B** into a new prompt. Mimic the tone and style of the original prompts. Make sure the new prompt is coherent and solvable*

—

Example Given Prompt A:
I have to pick up my son. Write a short SMS to my supervisor asking for leaving. In 20 words. Be polite.

Background Settings:
1.The user needs to pick up his son.

Objectives:
1.Write an SMS.

Constrains:
1.The SMS should be short.
2.The SMS should ask the supervisor's permission to leave.
3.The SMS should be polite.
4.The SMS should not exceed 20 words.

Example Given Prompt B:
I am planning to give you a voice, and communicate through the speech medium. I need a speech recognizer, a wake call detector, and a speech synthesizer for your voice. Suggest a python script utilizing existing libraries to achieves the goal.

Background Settings:
1.The user is planning to give a voice to a system and communicate through speech.
2.The user needs a speech recognizer, a wake call detector, and a speech synthesizer.

Objectives:
1.Suggest a Python script using existing libraries to achieve the goal.

Constraints:
N/A

Fused Background Settings:
1.The user needs to pick up his son.
2.The user is planning to give a voice to a system and communicate through speech.

3.The system requires a speech recognizer, a wake call detector, and a speech synthesizer.

Fused Objectives:
1.Suggest a Python script using existing libraries that enables a system to recognize speech, detect wake calls, and synthesize speech.
2.Use this system to compose and send an SMS.

Fused Constraints:
1.The SMS should be short.
2.The SMS should ask the supervisor's permission to leave.
3.The SMS should be polite.
4.The SMS should not exceed 20 words.
5. The SMS should be composed by the system and sent to the user's supervisor.

Fused Prompt:
Suggest a Python script utilizing existing libraries that includes a speech recognizer, a wake call detector, and a speech synthesizer to enable communication through speech for a system I am planning to implement. Use this system to send a short, polite SMS to my supervisor asking for permission to leave early because I have to pick up my son. The message should not exceed 20 words. Be Polite.

—

Please respond using the format provided in the example above. Give the merged background, objectives, and constraints respectively, then fuse the two given prompts into a new one incorporating your new background, objectives, and constraints.

Given Prompt A:
{{ prompt1 }}

{{ extracted1 }}

Given Prompt B:
{{ prompt2 }}

{{ extracted2 }}