Adaptive Instruction Induction for Enhancing Large Language Model Performance

Yuchen Han[a,1], **Zhaokun Dong**[a,1] **and Yuqing Sun**[a,*]

[a]Shandong University, Jinan 250000, China

Abstract. Instructions, as the primary means of using large language models (LLMs), significantly impact the results. Automatically inducing instructions from few-shot instances is meaningful, yet the induced instructions suffer from two inherent divergences: (1) systematic discrepancies across tasks, and (2) instance-level heterogeneity within a task. To address these challenges, inspired by human analogical reasoning, we propose **AdaIn**, an iteratively adaptive instruction induction framework that leverages instance-level structural similarities. AdaIn groups instances by data features to induce tailored instructions, and adaptively applies them to new samples. Instruction performance is further utilized to guide updates,, enabling iterative identification and refinement of ineffective instructions. We conducted experiments on different tasks to verify our method and the results demonstrate that it outperforms the SOTA results. Ablation experiments indicate that the adaptive strategy in induction and selection instruction contributes much to performance.

1 Introduction

Large language models (LLMs) demonstrate robust capabilities across a diverse range of complex tasks [2, 13, 18]. However, their inference performance is highly sensitive to input instructions, and simple prompts often fail to yield optimal results [8, 23, 10]. To maximize the task-specific potential of LLMs, users must trial various instructions. However, manually crafting such instructions is costly, limiting their ability to fully harness model capabilities.

To efficiently derive suitable natural language instructions, automatic induction methods utilize multiple input–output pairs to generate instructions through the induction model. However, as shown in Fig. 1, designing a universal induction framework remains challenging due to the heterogeneity of task features and the diversity of data features. One challenge is the systematic discrepancies across tasks, which result in different emphases. For example, *transformation* tasks prioritize accurate input–output mapping, whereas *classification* tasks emphasize the distinction between categories. Another challenge stems from instance-level heterogeneity within tasks—such as the number or order of output tokens—which imposes additional constraints on the induced instructions.

Existing methods include prompt design, feedback updating, and model fine-tuning. By designing precise prompts, LLMs can effectively conduct automatic prompt engineering [22]. Feedback methods like APE [24] and APO [12] use natural language gradients to

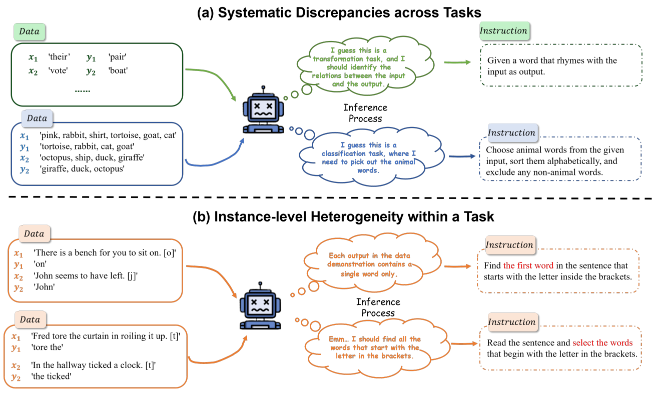

(a) Systematic Discrepancies across Tasks
Data
x_1 'their' y_1 'pair'
x_2 'vote' y_2 'boat'
......
I guess this is a transformation task, and I should identify the relations between the input and the output.
Instruction
Given a word that rhymes with the input as output.
Inference Process
Data
x_1 'pink, rabbit, shirt, tortoise, goat, cat'
y_1 'tortoise, rabbit, cat, goat'
x_2 'octopus, ship, duck, giraffe'
y_2 'giraffe, duck, octopus'
I guess this is a classification task, where I need to pick out the animal words.
Instruction
Choose animal words from the given input, sort them alphabetically, and exclude any non-animal words.
(b) Instance-level Heterogeneity within a Task
Data
x_1 'There is a bench for you to sit on. [o]'
y_1 'on'
x_2 'John seems to have left. [j]'
y_2 'John'
Each output in the data demonstration contains a single word only.
Instruction
Find the first word in the sentence that starts with the letter inside the brackets.
Inference Process
Data
x_1 'Fred tore the curtain in roiling it up. [t]'
y_1 'tore the'
x_2 'In the hallway ticked a clock. [t]'
y_2 'the ticked'
Emm... I should find all the words that start with the letter in the brackets.
Instruction
Read the sentence and select the words that begin with the letter in the brackets.

Figure 1. The challenges of the Instruction Induction

guide prompt updates. However, these methods do not exploit structural patterns in labeled samples, which limits their generalizability. Fine-tuning with synthetic data can improve instruction induction [16], but it is limited to open-source models and entails prohibitive costs for large parameter models.

In this paper, we propose an iteratively adaptive instruction induction method, **AdaIn**. We first analyze the task-specific features implied in the given instances, and **AdaIn** builds a preliminary understanding of the task. Inspired by the human tendency to understand new problems through analogy, we group instances by data features to induce instructions, which are then adaptively applied to new samples. To further refine these instructions, we introduce a self-correction strategy that iteratively identifies key elements through linguistic analysis and leverages inference results on labeled data as performance feedback to guide updates. The contributions are as follows:

- We propose **AdaIn**, an iterative and adaptive instruction induction method. By analyzing both task-level and data-level features, **AdaIn** induces the instruction set that effectively captures and adapts to feature variations.
- For task features, inspired by the chain-of-thought paradigm, we first leverage LLMs to analyze and preliminarily understand task requirements before instruction induction. We further introduce a linguistics-based self-correction strategy to refine induced instructions to enhance their quality.
- For data features, we group similar instances to induce feature-specific instructions. During inference, the sample-adaptive strategy selects the most suitable instruction for each instance.

* Corresponding Author. Email: sun_yuqing@sdu.edu.cn
[1] Equal contribution.

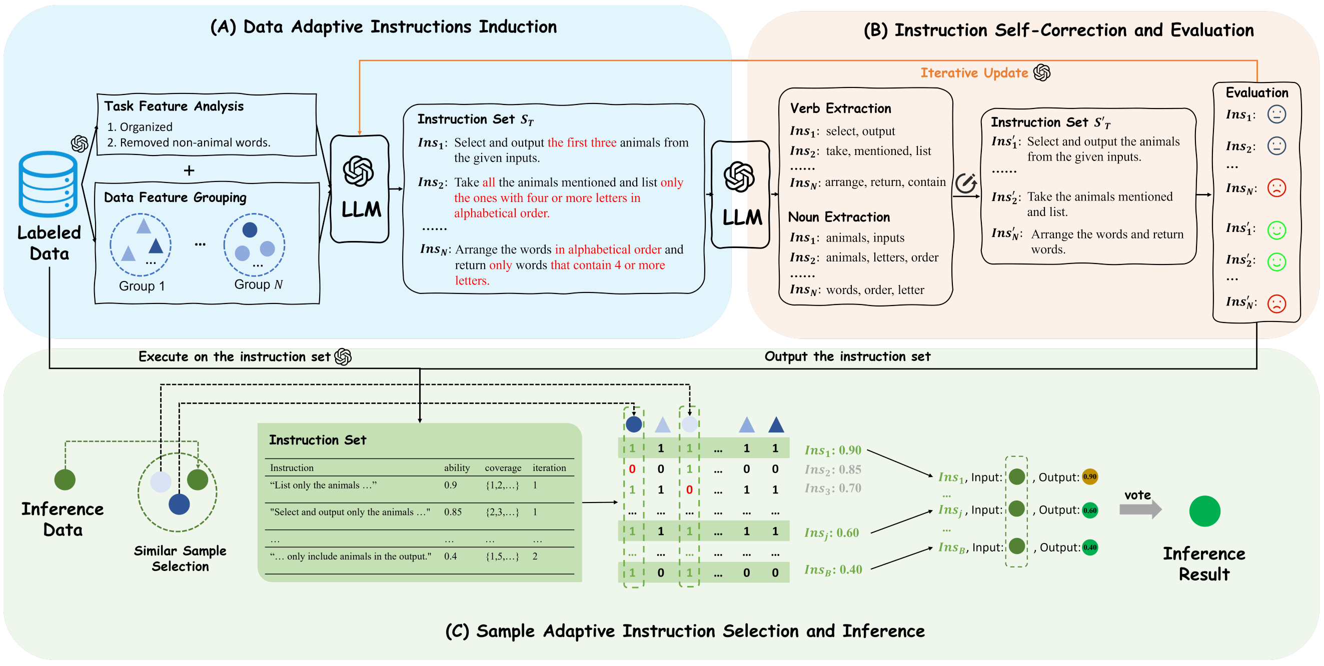

(A) Data Adaptive Instructions Induction
Task Feature Analysis
1. Organized
2. Removed non-animal words.
Data Feature Grouping
Group 1
Group N
Labeled Data
LLM
Instruction Set S_T
Ins_1: Select and output the first three animals from the given inputs.
Ins_2: Take all the animals mentioned and list only the ones with four or more letters in alphabetical order.
Ins_N: Arrange the words in alphabetical order and return only words that contain 4 or more letters.
(B) Instruction Self-Correction and Evaluation
Iterative Update
Verb Extraction
Ins_1: select, output
Ins_2: take, mentioned, list
Ins_N: arrange, return, contain
Noun Extraction
Ins_1: animals, inputs
Ins_2: animals, letters, order
Ins_N: words, order, letter
Instruction Set S'_T
Ins'_1: Select and output the animals from the given inputs.
Ins'_2: Take the animals mentioned and list.
Ins'_N: Arrange the words and return words.
Evaluation
Execute on the instruction set
Output the instruction set
Inference Data
Similar Sample Selection
Instruction Set
Instruction | ability | coverage | iteration
"List only the animals ..." | 0.9 | {1,2,...} | 1
"Select and output only the animals ..." | 0.85 | {2,3,...} | 1
"... only include animals in the output." | 0.4 | {1,5,...} | 2
Ins_1: 0.90
Ins_2: 0.85
Ins_3: 0.70
Ins_j: 0.60
Ins_B: 0.40
Ins_1, Input: , Output:
Ins_j, Input: , Output:
Ins_B, Input: , Output:
vote
Inference Result
(C) Sample Adaptive Instruction Selection and Inference

Figure 2. Overall framework of Our Method

2 Related Work

2.1 *Soft Prompt Update*

Soft prompts are trainable vector representations that serve as implicit prompt embeddings, which can be adjusted during training to optimize model performance [9]. Chen et al. [3] use an open-source LLM with soft prompts to generate natural language instructions, updating the prompts based on instruction evaluation to enhance zero-shot performance. BBT [15] and BBTv2 [14] iteratively query the model's inference API and apply Derivative-Free Optimization (DFO) algorithms to refine continuous prompts within a randomly generated low-dimensional subspace. Deng et al. [5] propose RLPROMPT, a reinforcement learning–based method that trains a parameter-efficient policy network to generate optimized discrete prompts, thereby enhancing model performance. PE2 [22] redesigns the prompt for guiding LLMs in instruction induction by incorporating additional components, aiming to more effectively perform both the induction and automatic updating processes.

2.2 *Natural Language Instruction Induction*

Automatic induction methods for generating natural language instructions derive instructions from multiple input-output pairs. Or Honovich et al. [7] design specific prompt templates to guide models in producing instructions that describe the relationships between example input–output pairs. The Self-Consistency method [19] samples multiple candidate instructions from LLMs and applies hypothesis search to select those with better execution performance. ItD [16] enhances the model's induction capability through fine-tuning, thereby improving the quality of induced instructions.

Given the varying quality of instructions generated in a single induction process, many approaches evaluate, filter, and update them to improve execution accuracy in downstream tasks. The APE method [24] generates candidate instructions through direct inference and semantic similarity-based sampling, then evaluates and selects them based on execution accuracy. The ProTiGi algorithm [12] forms natural language "gradients" from small batches of data to represent prompt updates, and propagates these gradients into the prompt to refine it. However, these methods do not fully leverage data features and lack linguistic knowledge guidance.

3 Method

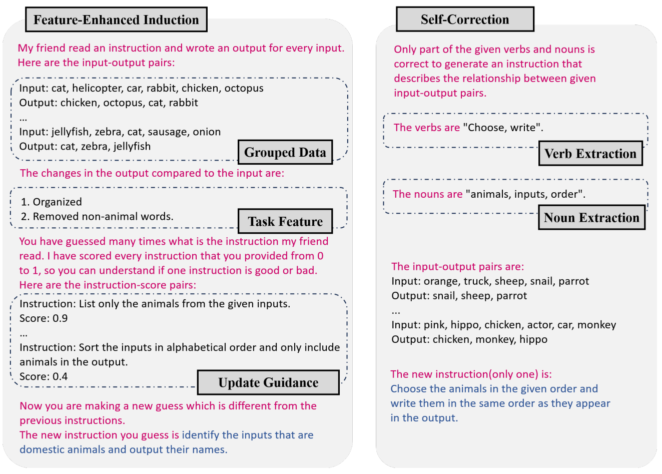
3.1 *Task Description and Framework*

Let M denote the induction model. For a given set of labeled instances $\mathbf{D}(x, y)$, the goal of instruction induction task is to generate a set of natural language instructions $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_B\}$ under the budget $B \in N^+$, such that Γ achieves the best expected performance over the true data distribution $\mathcal{P}$. Formally,

$$\Gamma^\star = \arg \max_{\Gamma \subset \mathcal{I}, |\Gamma| \leq B} \mathbb{E}_{(x,y)\sim\mathcal{P}}\big[s(\Gamma, (x, y))\big] \tag{1}$$

where $\mathcal{I}$ denotes the instruction space and $s(\Gamma, (x, y))$ is the evaluation score of Γ on instance (x, y). Our objective is to induce instructions that generalize well to all data drawn from the distribution $\mathcal{P}$.

Inspired by the way of analogical reasoning (i.e., interpreting novel tasks through structural similarities), we propose **AdaIn**, an iteratively adaptive instruction induction framework that leverages instance-level similarities. As illustrated in Fig. 2, **AdaIn** consists of three phases: (1) Data-Adaptive Instruction Induction, which groups instances by data features to induce feature-specific instructions; (2) Instruction Self-Correction and Evaluation, which refines instructions and provides iterative update guidance; and (3) Sample-Adaptive Inference, which enables dynamic instruction selection for new inputs via similar pattern matching. The following sections present each phase in detail.



My friend read an instruction and wrote an output for every input. Here are the input-output pairs:

Input: cat, helicopter, car, rabbit, chicken, octopus
Output: chicken, octopus, cat, rabbit
...
Input: jellyfish, zebra, cat, sausage, onion
Output: cat, zebra, jellyfish

The changes in the output compared to the input are:

1. Organized
2. Removed non-animal words.

You have guessed many times what is the instruction my friend read. I have scored every instruction that you provided from 0 to 1, so you can understand if one instruction is good or bad. Here are the instruction-score pairs:

Instruction: List only the animals from the given inputs.
Score: 0.9
...
Instruction: Sort the inputs in alphabetical order and only include animals in the output.
Score: 0.4

Now you are making a new guess which is different from the previous instructions.
The new instruction you guess is identify the inputs that are domestic animals and output their names.

Only part of the given verbs and nouns is correct to generate an instruction that describes the relationship between given input-output pairs.

The verbs are "Choose, write".

The nouns are "animals, inputs, order".

The input-output pairs are:
Input: orange, truck, sheep, snail, parrot
Output: snail, sheep, parrot
...
Input: pink, hippo, chicken, actor, car, monkey
Output: chicken, monkey, hippo

The new instruction(only one) is:
Choose the animals in the given order and write them in the same order as they appear in the output.

Figure 3. The prompts used in our method.

3.2 *Data Adaptive Instruction Induction*

According to the Theory of Multiple Representations in cognitive psychology [1], even for the same task, different data features require distinct instructions to address the variations effectively. Therefore, we analyze the data features and task features to enhance instruction induction.

First, we use LLMs to analyze the given instances to extract their common characteristics and potential knowledge, helping the induction model grasp the task's core rules and requirements. As shown in Fig. 1, task features vary considerably with task type. For example, *classification* tasks focus on input attributes to determine categories, while *transformation* tasks emphasize input-output mappings. This process provides an initial understanding of the task and its relevant knowledge, thereby guiding subsequent instruction induction.

The diversity of data features is critical for instruction generalization, and a single instruction often cannot fully adapt to complex data. In instruction induction, grouping samples with similar features helps improve induction accuracy. We design a data adaptive instruction induction method to tailor instructions. For each sample, we randomly select N anchor samples and, for each anchor (x_i, y_i), choose its n nearest neighbors based on the feature similarity, forming $Group_i$. Each group is used as examples in the induction prompt, from which the model M derives instructions adapted to various features:

$$\gamma_i^{ind} = M(Group_i; P_{ind}) \tag{2}$$

Where γ_i^{ind} denotes the instruction induced from the $Group_i$, and P_{ind} is the instruction induction prompt (see Fig. 3).

3.3 *Instruction Self-correction and Evaluation*

During our exploration of the instruction induction process, we observed that LLMs often add unnecessary restrictive modifiers, consistent with the findings of Yang et al. [21]. For example, the intended instruction '*Select and output only the animals from the given inputs*' was induced as '*Choose two animals from the inputs and write them in reverse alphabetical order*'. While some modifiers, such as output order, do not affect correctness, others—like limiting the number of animals—can cause execution errors.

Therefore, we propose an instruction self-correction mechanism that removes potentially error-inducing restrictive conditions, thereby increasing the likelihood of achieving higher execution accuracy. According to linguistic knowledge, the core structure of a sentence typically consists of verbs and nouns [4, 6]. We extract the core structure from each induced instruction, removing restrictive conditions that may cause execution errors. To remove potentially error-inducing restrictions, we extract the verbs and nouns from each induced instruction γ^{ind} (Fig. 3) and generate new instructions γ^{cor}, forming the self-corrected set Γ^{cor}.

Our goal is to select the optimal instruction set. To transform our problem into an optimization problem, we design a scoring function s to evaluate the quality of the generated instructions. For each instruction γ, we evaluate it on the labeled instances $\mathbf{D}$. The scoring function $s(\gamma, \mathbf{D})$ represents the average execution accuracy of instruction γ on the labeled instances $\mathbf{D}$.

$$s(\gamma, \mathbf{D}) = \frac{1}{n} \sum_{i=1}^{n} eval\left(\hat{y}_i, y_i\right) \tag{3}$$

Here, $\hat{y}_i$ represents the output given the instruction γ and input x. The function $eval\left(\hat{y}_i, y_i\right)$ is used to evaluate the difference between the output $\hat{y}_i$ guided by the instruction and the golden output y_i.

In iteration t, we evaluate the induced set Γ_t, its self-corrected version Γ_t^{cor}, and the previous best set $\Gamma_{t-1}^{\star}$, selecting the top B_t instructions as the new best set $\Gamma_t^{\star}$. These instructions and their scores are appended to the induction prompt as feedback for the next round.

3.4 *Adaptive Instruction Selection and Inference*

After T iterations, we obtain the final optimal instruction set $\Gamma_T^{\star}$. Since no single instruction can address all samples, instruction selection is performed dynamically based on each sample's features. Following the Analogical Reasoning Theory, for each new instance, we retrieve similar labeled ones and extract their effective instructions to form a tailored subset Γ^{sub}.

We design an instruction-sample matrix M_T to record the execution results of each instruction in the final instruction set on the labeled dataset $\mathbf{D}$. In this matrix, rows represent the instructions in the instruction set, while the columns represent the labeled instances. Let m_{ij} denote the execution status of the i-th instruction on the j-th labeled instance.

For a new sample x_{new}, we retrieve its top-k similar labeled instances and using M_T identify the instructions that are effective for them. These instructions are selected as the instruction subset Γ_{sub}. Inference is then performed via weighted voting over the predictions from Γ_{sub}, where each instruction's weight reflects its estimated ability:

$$\widehat{y}^{*} = \arg\max_{\hat{y}} \sum_{\gamma_i \in \Gamma_{\text{sub}}} s(\gamma_i, \mathbf{D}) \cdot \log P(\hat{y} | x_{new}, \gamma_i) \tag{4}$$

Here, $\log P(\hat{y}|x_{new}, \gamma_i)$ denotes the log-likelihood of the candidate output $\hat{y}$ given the input sample x_{new} and instruction γ_i, as computed by the inference model.

4 Experiments

4.1 *Dataset and Setups*

We evaluate our method on the Instruction Induction [7] and SNI [20] datasets. The Instruction Induction dataset consists of 24 sub-tasks, covering diverse natural language understanding tasks. For the SNI dataset, we select 10 sub-tasks for experimentation.

To comprehensively assess our approach, we conduct separate experiments on GPT-3.5-Turbo-Instruct [11] and LLaMA2-7B [17]. We select similar samples based on their performance on initial inductive instructions. For each task in datasets, we randomly sample 100 examples as labeled instances, following the same setting as the baselines. During induction, we set $k = 5$ similar instances for each round and set the instruction pool size at $B = 7$.

4.2 Baselines

We compare **AdaIn** with the following baseline methods:

APE [24] generates candidate instructions from labeled instances and selects the best-performing one.

APO [12] constructs a natural language "gradient" from small data batches to iteratively critique and update the prompt.

PE2 [22] redesigns the prompt by adding targeted components to better guide the instruction induction and automatic prompt updating.

ItD [16] boosts LLMs' inductive ability using a Deductive Data Generation module and a Naive Bayesian Induction module.

5 Results and Analysis

5.1 Main Results

Table 1. The results of the experiments conducted on the 14 subtasks of the Instruction Induction dataset and the 10 subtasks of the SNI dataset.

DataSet	**Instruction Induction**		**SNI**
Model	GPT-3.5	LLaMA2-7B	LLaMA2-7B
APE	62.60	<u>32.10</u>	<u>23.02</u>
APO	68.85	23.00	20.28
PE2	<u>69.95</u>	-	-
ItD	-	31.14	-
Ours (k=1)	**74.52**	**42.61**	**27.78**

We evaluate the execution performance of instructions generated by our method and the baselines' on the test set. For our method, we utilize the final instruction set after iteration, and the accuracy is measured using the sample-adaptive instruction selection strategy described in Section 3.4. We report mean and standard deviation over 5 runs. '–' indicates experiments that were not conducted due to unavailable code or resource limitations.

AdaIn outperforms all baselines, achieving a higher accuracy with a lower variance. The performance of the four methods on Instruction Induction is shown in Table 2. Our method achieves the highest execution accuracy across different tasks, indicating superior instruction effectiveness and greater robustness. As shown in Table 1, under the same LLaMA-7B setting, **AdaIn** significantly outperforms the fine-tuning-based ItD, achieving an improvement of 11.47%. On the SNI dataset, **AdaIn** also surpasses APE and APO, achieving the highest execution accuracy and further demonstrating its effectiveness.

To further investigate the superiority of our approach on the Instruction Induction dataset, we manually examine the final instruction set and intermediate results for the *translation_en-fr* task presented in Table 2. We found that leveraging data features to guide the selection of inductive examples helps the model generate higher-quality instructions. For instance, in the *translation_en-fr* task, our method produced the instruction: "*Translate the given word into French using traditional translation techniques, such as grammar and vocabulary rules.*" This instruction not only specifies the output goal but also provides detailed guidance, a specificity enabled by using examples with similar characteristics, which encourages more informative and well-structured instructions.

5.2 Ablation Study

We conduct ablation studies to evaluate the contribution of each component of our method. The experiments are divided into two parts: the first focuses on the induction process, examining the impact of task feature extraction, data feature grouping, update guidance, and self-correction; the second part targets the inference process, specifically evaluating the contribution of the sample-adaptive instruction selection mechanism.

5.2.1 Ablation on Induction Components

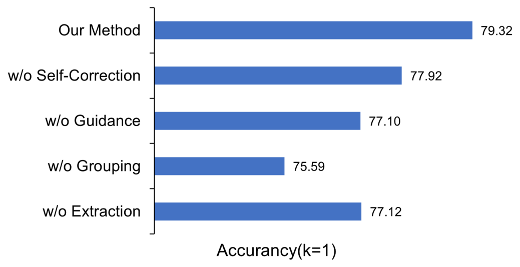


Figure 4. The ablation experiment results conducted on the complete set of 24 tasks in Instruction Induction.

We present the results of the ablation experiment for each component in the induction stage in Fig. 4.

Task and data features enhance instruction quality. We evaluate task feature extraction ("Extraction") and data feature grouping ("Grouping"). Among them, data features contribute the most to performance. The grouping step clusters instances with similar characteristics, enabling the induced instruction set to better capture the diversity across samples. Meanwhile, task feature extraction provides the model with a preliminary understanding of the task by analyzing the given instances and identifying the underlying knowledge required to solve it.

High-quality induced instructions guide iterative updates. The second is the addition of update guidance in the induction template, denoted as 'Guidance'. The update guidance module leverages previously induced instructions and their evaluations to iteratively refine instructions more effectively.

The self-correction process helps eliminate redundant constraints in induced instructions. The third component, 'Self-Correction', removes overly restrictive or potentially error-inducing conditions from instructions, thereby increasing the likelihood of achieving higher execution accuracy. Further analysis of this mechanism is provided in Section 5.4.

5.2.2 Ablation on Inference Component

Adaptive instruction selection strategy has better performance than using the single best-performing instruction. Comparison

Table 2. Results on 14 tasks in Instruction Induction. The results of the comparative methods are derived from PE2 [22].

Task	APE		APO		PE2		Ours (k=1)	
	Avg↑	Std↓	Avg↑	Std↓	Avg↑	Std↓	Avg↑	Std↓
antonyms	77.60	3.01	77.00	2.67	78.80	3.97	**82.80**	1.79
informal_to_formal	59.53	3.37	54.10	10.61	**61.26**	4.73	60.93	3.61
negation	77.80	2.48	75.40	7.17	76.00	7.24	**82.40**	1.34
orthography_starts_with	63.80	2.14	68.60	2.50	67.60	1.74	**69.40**	3.21
rhymes	25.60	12.52	56.75	22.72	65.00	19.88	**77.20**	27.87
second_word_letter	76.20	15.12	94.20	2.32	94.20	1.17	**96.80**	6.61
sentence_similarity	18.40	4.13	22.20	15.14	20.00	9.84	**40.00**	4.90
sentiment	88.20	2.79	88.80	2.79	88.80	2.79	**91.00**	1.41
synonyms	10.40	5.20	27.60	11.71	27.80	8.84	**29.20**	3.90
taxonomy_animal	80.80	9.06	88.80	7.86	89.00	8.76	**93.20**	4.15
translation_en-de	**85.00**	0.89	84.60	0.80	84.40	0.80	84.00	1.58
translation_en-es	84.80	0.98	85.40	0.80	85.40	0.49	**88.80**	1.30
translation_en-fr	71.80	9.99	81.80	3.66	80.00	3.22	**88.60**	1.52
word_in_context	56.40	6.92	58.60	7.66	**61.00**	1.67	59.00	1.58
Average	62.60	5.61	68.85	7.03	69.95	5.37	**74.52**	4.63

Table 3. Experimental results on the 10 subtasks from the SNI dataset using llama2:7b.

tasks	APO	APE	Ours(k=1)
identifying_essential_words	19.61	15.24	**31.97**
conala_max_absolute_value	**35.70**	23.77	21.61
logic2text_sentence_generation	**21.91**	19.00	21.52
dart_text_generation	29.67	44.51	**47.38**
winobias_text_generation	33.85	**49.10**	36.20
outcome_extraction	12.53	**12.72**	11.58
tweetqa_question_generation	13.89	12.00	**20.21**
synthetic_remove_vowels	24.62	24.12	**33.75**
find_numbers_or_alphabets_in_list	14.67	45.50	**51.92**
nummersense	**3.49**	1.31	1.68
avg	20.99	24.73	**27.78**

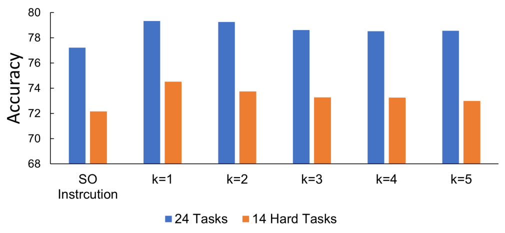


Figure 5. Comparison of SO instruction vs. instruction set.

methods often use instructions with the highest execution accuracy on labeled instances. We refer to this instruction as the SO (Seemingly Optimal) instruction. To validate the advantage of using the instruction set, we report the execution performance of both the SO instruction and our instruction set on the test sets of the full 24 tasks and a subset of 14 challenging tasks. The results are shown in Figure 5. As can be seen, using only the SO instruction for inference on the test samples results in lower performance compared to using the instruction set, regardless of the value of k. This proves that using the instruction set for inference is effective and that, during the induction phase, our induction method outperforms other methods. Secondly, we investigate the impact of the hyperparameter k in the adaptive instruction selection process. We observe that setting $k = 1$, i.e., selecting the effective instruction corresponding to the most similar instance, yields the best performance. This can be attributed to the few-shot nature of the experimental setting, where even the second most similar instance may exhibit a gap from the test instance. As a result, incorporating instructions from less similar instances may introduce noise and negatively affect the final prediction.

Table 4. The execution performance of induced instruction sets across different LLMs.

	SO	k=1	k=2	k=3	k=4	k=5
gpt-3.5-turbo-instruct	77.22	**79.32**	79.26	78.68	78.52	78.55
4o-mini	60.13	61.75	61.75	**61.93**	61.72	61.16
llama-3.1-8b	56.19	**64.68**	63.46	63.23	62.55	62.63
llama-3.1-70b	56.20	**64.64**	63.64	63.55	63.58	63.02
llama-3.1-405b	76.54	**79.86**	79.08	79.02	78.82	78.56

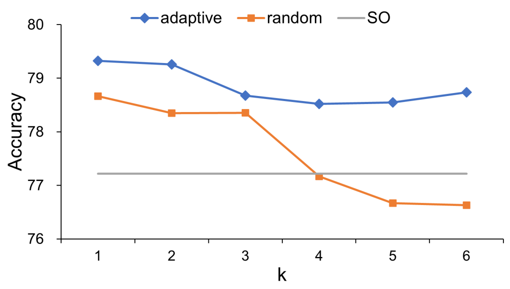


Figure 6. The impact of instructions selection strategy.

Instruction set improves execution performance across different LLMs. We also utilize the final generated instruction set to perform downstream task reasoning on different models, with the results shown in Table 4. Regardless of the model used, the overall execution accuracy of the instruction set is always higher than that of the SO instruction, no matter the value of k. Additionally, the results in Table 4 indicate that the choice of reasoning model also affects the performance of the final instruction subset. As the model parameters increase, the execution accuracy shows an upward trend.

The effective instructions derived from similar samples can significantly enhance execution accuracy. We have designed additional experiments to further investigate the efficacy of our adaptive instruction selection based on sample similarity. Under the condition that all other aspects remain unchanged, for each test sample, we replace the instruction subset used for inference on this sample with a randomly selected instruction subset, ensuring that both subsets are of the same size. Fig. 6 illustrates the impact of these two different methods of selecting instruction subsets on the final voting results. It can be observed that selecting instruction subsets based on sample similarity enhances the overall execution accuracy of the instruction set on the test data compared to the random selection method. We found that as the parameter k decreases, the model's execution accuracy shows a significant improvement. This phenomenon suggests that the effective instructions corresponding to the labeled samples most similar to the current sample's features are also applicable to the current sample, thereby effectively validating the hypothesis we proposed in Section 3.4.

5.3 *Analysis of Data Feature*

Table 5. Comparison of the instruction's execution on its corresponding inductive data and other data.

	Inductive data	Other data
antonyms	75.43	70.68
informal_to_formal	57.51	53.28
orthography_starts_with	89.71	76.03
translation_en-es	91.43	89.59
word_in_context	62.86	58.38

To verify the effectiveness of data features, we investigate the performance of instructions when applied within inductive data versus across other data. As shown in Table 5, the inductive data of an instruction refers to the labeled examples used to induce it, while the other data comprises all remaining labeled examples. For each sub-task, we compute and report the average execution performance of all instructions in the final instruction set on both their inductive data and other data. We find that the execution accuracy of instructions on their original induction groups is significantly higher than on other groups. This indicates that data features are beneficial for instruction induction, and highlights the necessity of introducing an instruction set to handle the diversity of data features.

5.4 *Analysis of Self-correction*

For each sub-task, we set the instruction set capacity to 7, resulting in 168 instructions for the 24 sub-tasks. We also provide statistics on their distribution, showing the round of generation and whether they were produced through the self-correction process. The statistical results are shown in Fig. 7. For the complete set of 24 tasks, 60.12% of the instructions were generated in the third round, and 31.55% of the instructions were produced by the self-correction process. This further demonstrates the effectiveness of both self-correction and iteration.

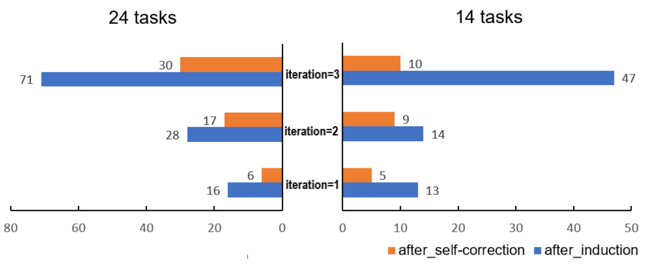
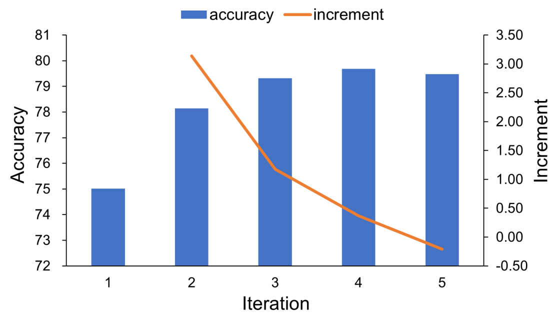
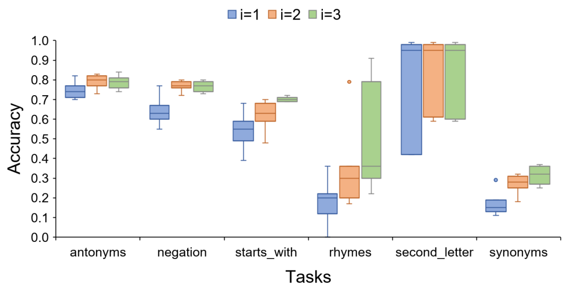


Figure 7. The distribution of instructions in the instruction set.



(a) Overall Execution Accuracy for Each Iteration



(b) Execution Accuracy of Each Instruction

Figure 8. Exploration of Iteration Rounds

5.5 *Analysis of iteration*

To validate the effectiveness of iteration, we demonstrate the capabilities of the instructions in the instruction set after each iteration. Fig. 8(a) shows the overall execution accuracy of the instruction set across 24 tasks as iteration rounds increase. Fig. 8(b) presents the execution accuracy of each instruction in the set after each iteration for six representative tasks. As can be seen, both the overall execution accuracy of the instruction set and the execution accuracy of each instruction improve with each iteration, demonstrating the effectiveness of iterative updates. Additionally, we further increase the number of iterations, and as shown in Figure 8(a), after $i = 3$, the instruction set's overall performance plateaus and even begins to decline, proving that 3 iterations is optimal.

5.6 *Analysis of Budget B*

In practice, B is chosen to balance performance and efficiency. We set $B = 7$ in our main experiments, and further evaluate its impact by reporting average instruction accuracy under different budget settings.

B	3	4	5	6	7
Acc.	78.95	78.90	78.98	79.03	79.32

Table 6. Average instruction accuracy under different budget B settings.

5.7 *Empirical Insights*

To provide some empirical insights about the linguistics or structural properties of the self-correction mechanism, we compare the instructions before and after using self-correction on different tasks. We observe that the high frequently removed words are typically restrictive modifiers—Quantifiers (e.g., each(103), any(52), all(47), two(56), more(25), some(20), Conjunctions(e.g., and(98), with(36), at(36), but(14)), Ordinal markers(e.g., alphabetical(20), first(38), second(13), last(17)), Comparative modifiers(e.g., similar(70), same(41), opposite(15)), and Negative modifiers(e.g., not(56), no(32), negative(41)), where the number in parentheses indicates the frequency. After a single round of self-correction, the accuracy is improved from 66.82 to 68.14, which demonstrate the effectiveness in refining the restrictive or misleading expressions.

5.8 *Case Study*

Table 7. Instruction Set Details per Iteration in the Synonyms Task

iteration 1	
Find words that have a similar meaning or concept, but may be expressed differently.	0.29
Read each word and write a synonym or a closely related word as the output.	0.18
"Provide each synonym of the input input as output"	0.19
Think of a synonym for each word.	0.11
Define each word by providing a related word or phrase with a similar meaning.	0.15
For each input word, provide a synonym or another related word as the output.	0.15
For each input word, write a word or phrase with a similar meaning or connotation.	0.13
iteration 2	
Find alternative words to convey the idea that the input conveys the meaning of.	0.25
Identify words that are synonymous or have a closely related meaning and write them as the output for each input word.	0.32
Find alternative words or phrases that convey a similar idea or meaning as the input.	0.25
Find words that have a similar meaning or connection and use them as output, while maintaining the original meaning of the input word.	0.31
Use context clues to determine words with similar meanings and provide them as outputs.	0.28
Think of related concepts when writing about input and use those related words to generate the output.	0.31
Find and replace synonyms of input with related words.	0.18
iteration 3	
Use a thesaurus to find words with similar meanings as the input and use these words as outputs.	0.27
Use a thesaurus to find synonyms of the input words and use them as the output.	0.32
Use a thesaurus to find synonyms or related words for the input and use them as the output.	0.37
Use a thesaurus to find words with similar meanings and use them as outputs, while staying true to the original meaning of the input word.	0.36
Use a thesaurus to find alternative words for the input and write them as output.	0.31
Find words that are related to the input in a broader sense and use them as outputs.	0.25
Use a thesaurus to find synonyms and related words to create the output for each input.	0.33

We present a detailed iterative process for the synonyms task. Table 7 illustrates the changes in the set of instructions after each round of iteration, with three rounds completed in total.

In this task, most labeled examples contained only a single word, $e.g., Input : decide; Output : determine$, leading the initially induced instructions to emphasize that a single synonymous word is sufficient. However, instructions that encourage the generation of multiple near-synonyms tend to achieve higher evaluation scores. Therefore, the blue instruction achieved the highest score in iteration 1. Iterative induction guides the model to generate instructions similar to high-scoring ones, favoring multi-word outputs that align better with evaluation criteria.

Table 8. The induction results performed on the modified labeled instances.

iteration 1	
Provide alternative words or phrases that could replace the given input, which have similar meanings.	0.31
Think of words that have similar meanings to the given input.	0.31
List the synonyms of a word to reflect a potential change between the input and output.	0.30
Provide alternative words or phrases that have similar meanings for better understanding.	0.32
Provide alternative words or phrases that have similar meanings for the input word. This will help improve understanding of the input-output relationship.	0.27
Please provide five synonyms or related words for the given input.	0.28
Use synonyms or variations of the given word to expand the output and add more specific or nuanced terms to describe the input.	0.33
iteration 2	
Provide alternative words or phrases with similar meanings for the given input to improve understanding of the relationship between the input and output.	0.30
Provide different words or phrases that could be substituted for the input to convey a similar meaning.	0.35
Use a thesaurus to find synonyms of the given input and write them in the output to add more depth and clarity.	0.36
Provide synonyms or related words for the given input to create a more diverse and comprehensive output.	0.38
Think of alternative words or phrases that convey a similar meaning to the given input and use them to create a more nuanced and varied output.	0.36
Provide alternative words or phrases that could replace the given input, which have similar meanings.	0.31
Encourage the use of vocabulary expansion by providing alternative words or phrases with similar meanings for a given input.	0.34
iteration 3	
Provide alternative words or phrases with similar meanings for the given input to improve understanding of the relationship between the input and output.	0.30
Provide synonyms or related words for the given input to create a more diverse and comprehensive output that conveys the same meaning.	0.36
Use a thesaurus or dictionary to find synonyms or related words for the given input and incorporate them into the output to enhance understanding and diversity.	0.36
Instruction: Enhance understanding of relationships between words by offering synonyms or related words for the given input to create a more diverse and nuanced output.	0.35
Provide different words or phrases that could be substituted for the input to convey a similar meaning.	0.35
Use a thesaurus to find synonyms of the given input and write them in the output to add more depth and clarity.	0.36
Provide synonyms or related words for the given input to create a more diverse and comprehensive output.	0.38

When data features were adjusted by expanding the number of synonyms in the output, as shown in Table 8, instructions induced from the modified annotated data consistently achieved higher scores under the same number of iterations. This further demonstrates the significant impact of data features on the instruction induction task.

6 Conclusion

In this paper, we propose the iteratively adaptive instruction induction method (**AdaIn**). Through data adaptive instruction induction and iterative update, the quality of the induced instructions is improved. Additionally, by sample adaptive instruction selection, the most suitable instructions are automatically chosen based on the features of the current sample. We conducted experiments on 24 tasks, and the results validate our method effectiveness.

Acknowledgements

This work was supported by the Innovative Development Joint Fund Key Projects of Shandong NSF (ZR2022LZH007), the National Natural Science Foundation of China (62376138).

References

[1] S. Ainsworth. Deft: A conceptual framework for considering learning with multiple representations. *Learning and instruction*, 16(3):183–198, 2006.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[3] L. Chen, J. Chen, T. Goldstein, H. Huang, and T. Zhou. Instructzero: Efficient instruction optimization for black-box large language models. *arXiv preprint arXiv:2306.03082*, 2023.

[4] N. Chomsky. *Aspects of the Theory of Syntax*. Number 11. MIT press, 2014.

[5] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. Xing, and Z. Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391, 2022.

[6] Y. Han, T. Liu, Y. Sun, T. Huang, H. Wu, and S. Wu. Exploring word composition knowledge in language usages. In *International Conference on Knowledge Science, Engineering and Management*, pages 61–72. Springer, 2024.

[7] O. Honovich, U. Shaham, S. Bowman, and O. Levy. Instruction induction: From few examples to natural language task descriptions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1935–1952, 2023.

[8] Z. Jiang, F. F. Xu, J. Araki, and G. Neubig. How can we know what language models know? *Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

[9] X. L. Li and P. Liang. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, 2021.

[10] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098, 2022.

[11] OpenAI. Introducing chatgpt. https://openai.com/blog/chatgpt, 2023b.

[12] R. Pryzant, D. Iter, J. Li, Y. Lee, C. Zhu, and M. Zeng. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, 2023.

[13] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

[14] T. Sun, Z. He, H. Qian, X. Huang, and X. Qiu. Bbtv2: Pure black-box optimization can be comparable to gradient descent for few-shot learning. *arXiv preprint arXiv:2205.11200*, 2022.

[15] T. Sun, Y. Shao, H. Qian, X. Huang, and X. Qiu. Black-box tuning for language-model-as-a-service. In *International Conference on Machine Learning*, pages 20841–20855. PMLR, 2022.

[16] W. Sun, H. Xu, X. Yu, P. Chen, S. He, J. Zhao, and K. Liu. ItD: Large language models can teach themselves induction through deduction. In L.-W. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2719–2731, Bangkok, Thailand, Aug. 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.150. URL https://aclanthology.org/2024.acl-long.150/.

[17] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[18] J. Wang, Y. Sun, Y. Liang, X. Li, and B. Gong. Iteratively calibrating prompts for unsupervised diverse opinion summarization. In *ECAI 2024*, pages 3939–3946. IOS Press, 2024.

[19] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

[20] Y. Wang, S. Mishra, P. Alipoormolabashi, Y. Kordi, A. Mirzaei, A. Arunkumar, A. Ashok, A. S. Dhanasekaran, A. Naik, D. Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*, 2022.

[21] Z. Yang, P. Li, and Y. Liu. Failures pave the way: Enhancing large language models through tuning-free rule accumulation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1751–1777, 2023.

[22] Q. Ye, M. Axmed, R. Pryzant, and F. Khani. Prompt engineering a prompt engineer, 2024. URL https://arxiv.org/abs/2311.05661.

[23] Z. Zhao, E. Wallace, S. Feng, D. Klein, and S. Singh. Calibrate before use: Improving few-shot performance of language models. In *International conference on machine learning*, pages 12697–12706. PMLR, 2021.

[24] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large language models are human-level prompt engineers, 2023. URL https://arxiv.org/abs/2211.01910.