Learning to Align Multi-Faceted Evaluation: A Unified and Robust Framework

Anonymous ACL submission

Abstract

Large Language Models (LLMs) are being used more and more extensively for automated evaluation in various scenarios. Previous studies have attempted to fine-tune open-source LLMs to replicate the evaluation explanations and judgments of powerful proprietary models, such as GPT-4. However, these methods are largely limited to text-based analyses under predefined general criteria, resulting in reduced adaptability for unseen instructions and demonstrating instability in evaluating adherence to quantitative and structural constraints. To address these limitations, we propose a novel evaluation framework, ARJudge, that adaptively formulates evaluation criteria and synthesizes both text-based and code-driven analyses to evaluate LLM responses. ARJudge consists of two components: a fine-tuned Analyzer that generates multi-faceted evaluation analyses and a tuning-free Refiner that combines and refines all analyses to make the final judgment. We construct a Composite Analysis Corpus that integrates tasks for evaluation criteria generation alongside text-based and code-driven analysis generation to train the Analyzer. Our results demonstrate that ARJudge outperforms existing fine-tuned evaluators in effectiveness and robustness. Furthermore, it demonstrates the importance of multi-faceted evaluation and code-driven analyses in enhancing evaluation capabilities.

1 Introduction

The rapid advancement of Large Language Models (LLMs) has highlighted the critical need for robust output evaluation methods (Li et al., 2024a). While proprietary models like GPT-4 have emerged as predominant evaluation approaches given their superior capabilities, transparent and controllable considerations have driven research toward fine-tuning open-source LLMs for evaluation tasks (Kim et al., 2024a,b). Recent work has established the viability of open-source alternatives by training LLMs to replicate the evaluation explanations and judgments of proprietary models (Ke et al., 2024; Liu et al., 2024; Hu et al., 2024; Kim et al., 2024b).

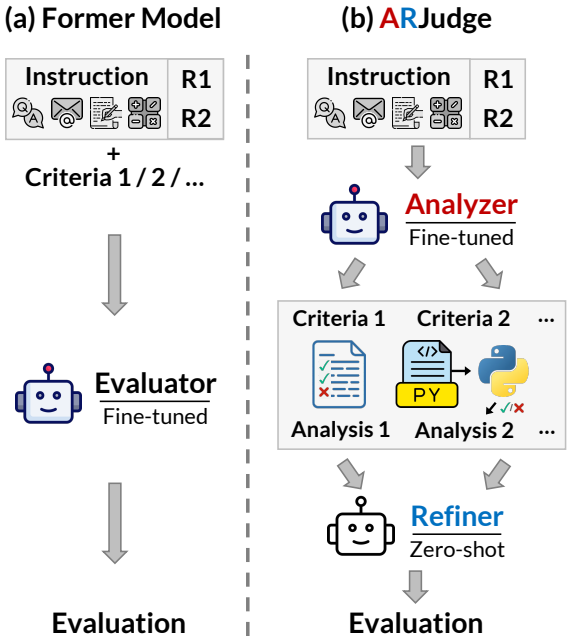

Figure 1: Comparison of previous fine-tuned evaluators and our framework. **Left** is a former model and **Right** is our ARJudge. The Analyzer adaptively defines evaluation criteria and conducts multi-faceted analyses in various forms, e.g., text or code. The Refiner combines all preceding analyses and produces the final evaluation.

However, existing fine-tuned evaluators rely solely on text-based analysis with predefined evaluation criteria, leading to two key limitations (Li et al., 2024b; Hu et al., 2024; Zhu et al., 2023; Kim et al., 2024b). First, evaluation based on predefined criteria can not fully capture the nuanced task requirements. For example, general criteria for writing, such as conciseness or logical structure, may not be sufficient for evaluating creative writing tasks that require an engaging plot. Moreover, it is challenging to effectively adapt predefined criteria to new and diverse instructions (Li et al., 2024b).

Second, LLM-based evaluators demonstrate significant instablity in evaluating adherence to complex instruction requirements, particularly objective criteria such as quantitative or structural constraints (Zhou et al., 2023). For instance, they struggle to reliably assess basic textual attributes such as word counts, a common requirement in writing-related instructions (Zhang and He, 2024). These limitations also extend to the evaluation of formatting constraints.

In this work, we argue that developing robust fine-tuned evaluators requires the ability to adaptively generate evaluation criteria and conduct multi-faceted analyses (Saha et al., 2024). These abilities enhance the evaluators' comprehensive performance in both what to evaluate and how to evaluate. Even for unseen instructions, the evaluators can define tailored criteria and assess instructions with nuanced precision. Furthermore, evaluators should use automated tools to assess objective requirements (Wang et al., 2024a). These tools provide reproducible feedback, offering reliable verification that helps overcome LLMs' inherent limitations in objective evaluation.

To address these challenges, we propose ARJudge, a novel evaluation framework that combines adaptive criteria generation with text-based and code-driven analysis generation to comprehensively assess LLM outputs. ARJudge comprises two core components: (1) an Analyzer that generates multi-faceted evaluation with text-based and code-driven analyses and (2) a Refiner that synthesizes and refines these analyses to produce well-reasoned judgments. We train ARJudge on a curated Composite Analysis Corpus, which contains tasks for generating evaluation criteria and performing multi-faceted analyses in both text and code. This corpus enables the Analyzer to learn context-sensitive evaluation logic, such as deriving criteria from instructions and assessing responses accordingly. Extensive experiments across multiple benchmarks demonstrate ARJudge's superiority and robustness over existing open-source evaluators. Our further analysis validates the necessity and effectiveness of integrating code-driven analyses, which improve accuracy in evaluating instruction following by up to 11.1% compared to text-only methods.

The main contributions of this work include:

- We propose ARJudge, a novel evaluation framework that combines adaptive criteria generation with text-based and code-driven analyses to evaluate LLM outputs. By incorporating code-driven analytical capabilities, ARJudge extends beyond traditional text-based evaluation approaches.

- We develop a training dataset, Composite Analysis Corpus, containing samples for evaluation criteria generation, text-based analyses, and code-driven analyses. It is the first dataset to incorporate multi-faceted analytical samples for evaluator training.

- Extensive experiments across multiple benchmarks demonstrate ARJudge's superior performance over existing fine-tuned evaluators.

2 Composite Analysis Corpus

Collecting comprehensive and detailed evaluation analysis data is essential for fine-tuning an LLM to improve evaluation performance (Li et al., 2024b; Hu et al., 2024). Previous studies (Li et al., 2024b; Hu et al., 2024; Kim et al., 2024a,b) focus exclusively on text-based analysis with predefined general evaluation criteria, showing limited generalization and robustness (Huang et al., 2024a). To address these limitations, we develop a composite analysis corpus to improve LLMs' ability to determine what to evaluate and how to evaluate effectively. The process of constructing the corpus involves three steps: (1) establishing evaluation criteria specifically for each instruction (§2.1), (2) conducting text-based analyses to assess responses using multiple criteria (§2.2), and (3) designing code-driven analyses to assess whether responses satisfy the objective requirements of the instructions (§2.3).

First of all, we collect a large set of instructions from publicly available preference datasets based on Li et al. (2024b). These datasets (Zheng et al., 2023a; Nakano et al., 2021; Havrilla, 2023; Ji et al., 2023) consist of preference pairs of LLM-generated responses to identical instructions. Each pair is annotated with a preference label that identifies the better response. In line with Li et al. (2024b), non-English instructions and multi-turn interactions are removed. Then, we establish multiple evaluation criteria for each instruction.

2.1 Establishing Evaluation Criteria

We define the evaluation criteria in the form of concise questions (Zeng et al., 2024; Kim et al.,

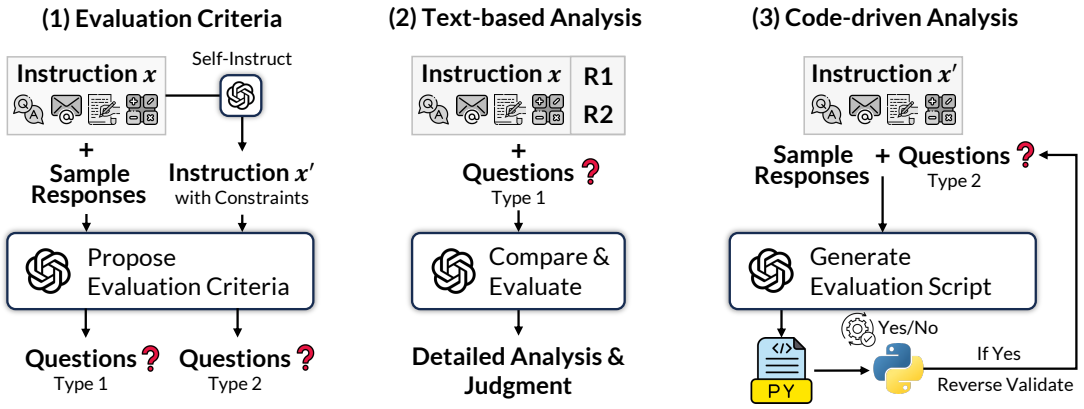

Figure 2: The overview of the corpus construction. "**R1**" and "**R2**" denote two candidate responses with a preference annotation. "**Sample Responses**" are newly sampled responses that we use as references to generate evaluation questions and code scripts. Step (1) produces two types of evaluation questions, respectively. Step (2) and Step (3) develop corresponding text-based and code-driven analyses.

2024b). Each question describes one aspect that a high-quality response should fulfill. For example, responses to the instruction "Draft an email to my deputy chairperson requesting their signature on the attached approval letter in a professional and polite manner" can be evaluated using the following three questions: "*1. Does the response include a polite and professional request for the deputy chairperson to sign the attached approval letter? 2. Does the response mention the attached approval letter and provide the necessary details about it? 3. Does the response offer assistance with any questions or clarifications the deputy chairperson might have about the approval letter?*" We establish two types of questions by prompting an LLM in a zero-shot manner. **Type 1** focuses on generating text-based analysis, while **Type 2** involves generating Python functions and using execution feedback as code-driven analysis.

To generate the first type of question, we prompt an LLM using three sample responses produced by advanced LLMs as well as the instruction x. Such sample responses offer a reference understanding of the instruction. The specific prompt is shown in Figure 6. We collect three questions q_{text} for each instruction x following Zeng et al. (2024) and construct training samples in the format (x, q_{text}).

For the second type, we must generate new instructions x' with objective constraints in advance, since their proportion in the datasets is relatively low. We use the self-instruct (Wang et al., 2023b) method to add objective constraints to the instructions and then produce evaluation questions for verifying these constraints. Following the verifiable instructions[1] summarized by Zhou et al. (2023), we first generate several objective constraints for each instruction, such as "word count" and "end with". The specific prompt is shown in Figure 7. Then, we randomly select one to three constraints to add to each instruction and collect the corresponding evaluation questions q_{code}. The training samples are constructed in the format (x', q_{code}).

2.2 Collecting Text-based Analysis

We perform pairwise text-based analyses by providing an LLM with the instruction x, two responses r_1 and r_2, and their corresponding evaluation questions $\{q_{text}\}$. The output necessitates a comparative analysis for each question, followed by a final determination of the better response. The specific prompt is shown in Figure 8. We exclude analyses where the final decision contradicts existing human annotations in the datasets. The training samples are constructed in the format $(x \oplus r_1 \oplus r_2 \oplus q_{text}, y_{text})$. Here, y_{text} denotes the associated analysis result for the evaluation question q_{text}, which begins with the hint: "*Let's evaluate whether responses meet the criteria*".

2.3 Developing Code-driven Analysis

To enhance evaluation robustness, we develop code-driven analyses to assess evaluation questions designed to verify objective requirements. The process is completed in two steps: **Collecting Python Scripts** and **Reverse Validation**. The first step involves generating Python functions to analyze

[1]Verifiable instructions are instructions that can be objectively verified for compliance using tools.

whether a response satisfies the objective requirement included in an evaluation question. The second step reversely checks whether the generated function's code is designed to analyze the evaluation question.

Collecting Python Scripts. Given three sample responses and one evaluation question q_{code}, we prompt an LLM to generate a Python function for verifying the compliance of sample responses. The input of the function is one response, and the output is a comprehensive intermediate of the results related to the evaluation questions. To ensure good generalization, the sample responses are a mix of outputs from advanced and weak LLMs. The specific prompt is shown in Figure 9. After prompting, we extract the generated Python function using Markdown parsing. We preliminarily filter out invalid code using two checks: 1. The written Python function fails to execute with the provided sample responses as input; 2. The function fails when tested with an additional set of three sample responses. By filtering out invalid code, we ensure that the generated Python functions are executable and generalizable.

Reverse Validation. To further validate whether the generated Python functions fulfill their intended purpose, we design a reverse validation process. Specifically, we first prompt an LLM with the plain text of the evaluation function, requesting an explanation of the expected outputs. Second, we prompt the LLM again to check for consistency between the explanation and its associated question:

$$\begin{aligned} e &\sim LLM(f, \text{prompt}_{explain}) \\ r &= LLM(e, q_{code}, \text{prompt}_{check}) \end{aligned} \tag{1}$$

where f is the evaluation function, e denotes the generated explanation, r indicates whether the explanation is consistent with the question q_{code}. The specific prompts are included in Figure 10. If the function is found to be inconsistent with the aim of the evaluation question, it is discarded. Finally, we collect the effective Python functions and construct training samples in the format $(x' \oplus r_1 \oplus r_2 \oplus q_{code}, y_{code})$. Here, y_{code} represents the Python function f concatenated with the code output hint "*Let's write a Python function*".

3 ARJudge

After constructing the corpus, we collect around 25K composite training samples. We fine-tune an LLM based on them and develop **ARJudge**, a novel evaluation framework that adaptively evaluates LLM-generated responses and integrates both text-based and code-driven analyses. ARJudge consists of two components: a fine-tuned **Analyzer** and a tuning-free **Refiner**. Figure 1 presents the overall framework. The Analyzer is trained on the Composite Analysis Corpus to adaptively generate evaluation criteria for any instruction and produce multi-faceted evaluation, including both text-based and code-driven analyses. The Refiner leverages the general LLM's generalist evaluation capabilities to refine the analysis results produced by the Analyzer and make the final judgment. This framework partially preserves the generalist evaluation pattern of the general model while enhancing the evaluation pattern in the fine-tuning dataset.

3.1 Training

We train the Analyzer with diverse training samples and tasks, including question generation samples (x, q_{text}) and (x', q_{code}), text-based analysis samples $(x \oplus r_1 \oplus r_2 \oplus q_{text}, y_{text})$, and code-driven analysis samples $(x' \oplus r_1 \oplus r_2 \oplus q_{code}, y_{code})$. By training on these combined samples, we aim to enhance the LLM's comprehensive analytical capabilities, enabling it to adaptively propose evaluation criteria and conduct multi-faceted analyses. We employ distinct prompt templates for question generation and response analyses, while maintaining a consistent prompt template for both text-based and code-driven analyses. Different forms of analyses are triggered by their respective starting hints.

3.2 Evaluation

Given an instruction x and two responses r_1 and r_2, the Analyzer first generates several evaluation questions. Then, it performs a comparative analysis of the two responses based on each evaluation question. Notably, the Analyzer autonomously determines whether to generate Python functions according to question characteristics. If the analysis text includes Python functions, the Analyzer will call a Python interpreter to execute them and return the execution feedback as the code-driven analysis results. Finally, the above multi-faceted analysis results are aggregated and sent to the Refiner for further evaluation. We instruct the Refiner to evaluate the above analysis and refine it with a renewed focus on the instruction's requirements. The Refiner will determine which response is better in a zero-shot manner.

4 Experiments

4.1 Implementation Details

To construct the Composite Analysis Corpus, we prompt GPT-4o to generate evaluation questions for each instruction and collect text-based analysis. Besides, we prompt Claude-3.5-Sonnet to generate Python functions for code-driven objective analysis. We selected Claude-3.5-Sonnet due to its superior performance in code generation. We fine-tune Qwen2.5-7B-Instruct (Qwen, 2025) on the corpus, creating a model we refer to as the Analyzer for performing multi-faceted evaluations. We use the same model in a zero-shot setting as the Refiner, with carefully crafted prompt templates. All generation in the main experiments is performed using greedy decoding by setting the temperature to 0. Details are described in Appendix A.

4.2 Benchmarks

We assess our framework on various evaluation datasets. Four human-annotated pairwise evaluation test sets are included: PandaLM Eval (Wang et al., 2024b), Auto-J Eval (Li et al., 2024b), MT-Bench (Zheng et al., 2023a), and the LLMBar series (Zeng et al., 2024). These sets were chosen for their broad coverage of evaluation tasks and their diverse set of evaluation criteria. For the LLMBar series, we use four adversarial sets, Neighbor, GPTInst, GPTOut, and Manual, as unseen sets. Unlike the other three sets and our training datasets, where candidate responses are directly sampled based on instructions, the responses in LLMBar are artificially designed to challenge evaluators by incorporating potentially misleading qualities, such as a more engaging tone. One GPT-4-annotated pairwise evaluation set, JudgeLM Eval (Zhu et al., 2023), is adopted. For all pairwise sets, samples with two equally preferred responses were omitted. Additionally, an instruction-following benchmark, IFEval (Zhou et al., 2023), is incorporated. We use this benchmark to assess the effectiveness of code-driven analysis.

4.3 Baselines

Tuning-free General LLMs We compare our framework with several general LLMs that can evaluate response quality. Three powerful LLMs, GPT-4o (OpenAI, 2024), Deepseek-v3 (DeepSeek-AI, 2024), and Claude-3.5-Sonnet (Anthropic, 2024), are used due to their balanced and comprehensive performance across most evaluation tasks (Huang et al., 2024a). Additionally, the backbone model used for fine-tuning the Analyzer, Qwen2.5-7B-Instruct (Qwen, 2025), is employed to demonstrate improvements.

Fine-tuned Evaluators We employ five fine-tuned evaluation models that can conduct pairwise evaluation. PandaLM (Wang et al., 2024b) compares two responses and identifies the better one. Auto-J (Li et al., 2024b) and Prometheus (Kim et al., 2024b) support both single-response scoring and pairwise response comparison. Themis (Hu et al., 2024) rates each response based on various criteria and determines the better one by comparing their scores. JudgeLM (Zhu et al., 2023) provides a comparison of two responses along with their corresponding scores. We use official models with 7B parameters for PandaLM, Prometheus, and JudgeLM, and models with 13B and 8B parameters for Auto-J and Themis, respectively.

4.4 Main Results

The main comparative results against baseline methods are shown in Table 1. Following Zeng et al. (2024) and Li et al. (2024b), we calculate the accuracy of the pairwise preference evaluation with and without swapping the two candidate responses, respectively. The average accuracy and the positional agreement rate are displayed as **Acc** and **Agr**. The performance in LLMBar is the average of its four subsets. We observe that ARJudge surpasses all fine-tuned evaluators of similar model sizes. Notably, on the challenging LLMBar set, ARJudge outperforms the best fine-tuned baseline, Prometheus2-7B, by 26.7%. Even without more exposure to challenging samples like LLMBar, ARJudge achieves an average 15.6% improvement over its backbone model, Qwen2.5-7B-Instruct. Additionally, ARJudge's performance is comparable to that of powerful tuning-free LLMs on some test sets. For example, ARJudge performs on par with GPT-4o and Claude-3.5-Sonnet on Auto-J Eval and with DeepSeek-V3 on LLMBar. Besides, compared to other fine-tuned methods, ARJudge can generalize to more test sets.

Table 2 further presents detailed evaluation results in different subsets of LLMBar. Our ARJudge performs the best on most subsets and has made significant improvements compared to the backbone model, Qwen2.5-7B-Instruct. On LLMBar-Neighbor, it achieves higher evaluation accuracy than the advanced DeepSeek-V3.

Models	JudgeLM Eval		PandaLM Eval		Auto-J Eval		MTBench		LLMBar		Ave
	Acc	Agr	Acc	Agr	Acc	Agr	Acc	Agr	Acc	Agr	
Tuning-free											
GPT-4o	81.8	88.1	83.1	87.5	78.6	82.5	78.8	85.4	79.8	83.4	80.4
Claude-3.5-Sonnet	82.9	86.4	86.4	91.4	78.2	85.5	**80.8**	89.1	**83.4**	90.3	**82.3**
Deepseek-v3	**83.2**	85.9	**87.4**	87.8	**82.9**	84.2	79.7	87.0	68.6	81.6	80.4
Qwen2.5-7B	80.0	78.0	80.7	79.2	73.8	65.1	75.2	72.1	52.6	65.7	72.5
Fine-tuned											
PandaLM-7B	69.9	74.7	73.1	77.8	65.2	71.0	74.0	78.4	25.9	82.5	61.6
Auto-J-13B	77.9	86.6	77.2	87.2	**79.7**	87.5	75.0	84.2	27.8	83.6	67.5
Prometheus2-7B	76.5	80.3	76.3	70.9	75.1	77.2	74.3	79.5	41.5	77.6	68.7
JudgeLM-7B	**81.8**	86.0	70.3	81.4	66.1	80.2	64.6	77.1	28.1	82.0	62.2
Themis-8B	66.4	-	61.3	-	39.2	-	34.9	-	26.6	-	45.7
ARJudge	81.0	83.3	**82.4**	83.5	78.5	80.3	**78.3**	81.3	**68.2**	72.9	**77.7**

Table 1: Results of different evaluators on the pairwise comparison. "**Acc**" and "**Agr**" denote average accuracy and positional agreement rate. "**Ave**" is the average "Acc" across all test sets. The highest average accuracy is marked by **bold** for two series models, respectively.

Models	LLMBar			
	Neighbor	GPTInst	GPTOut	Manual
Tuning-free				
GPT-4o	81.0	86.4	75.5	76.1
Claude-3.5-Sonnet	**83.2**	**87.0**	**76.6**	**87.0**
Deepseek-v3	61.6	76.6	69.2	67.4
Qwen2.5-7B	47.0	56.0	61.7	45.6
Fine-tuned				
PandaLM-7B	14.9	21.2	48.9	18.5
Auto-J-13B	20.5	21.2	47.9	21.7
Prometheus2-7B	25.4	31.0	**63.8**	45.6
JudgeLM-7B	21.3	25.5	41.5	23.9
Themis-8B	20.2	32.6	31.9	21.7
ARJudge	**72.4**	**73.4**	60.7	**67.4**

Table 2: Evaluation accuracy on test subsets of LLMBar series. The highest average accuracy is marked by **bold**.

Models	JudgeLM	PandaLM	Auto-J	MTBench	LLMBar
Qwen2.5-7B	80.0	80.7	73.8	75.2	52.6
ARJudge	81.0	82.4	78.5	78.3	68.2
-w/o FT	73.1	75.6	68.7	70.0	62.5
-w/o FT&MF	74.7	72.2	65.6	67.8	63.7
-w/o Refine	81.7	82.8	79.6	79.1	63.7

Table 3: Comparison results under ablation settings. "JudgeLM", "PandaLM", and "Auto-J" are abbreviation of the associated testsets. "**FT**" and "**MF**" represent fine-tuning and multi-faceted.

4.5 Ablation Study

To further investigate the effectiveness of our framework, we analyze several variations of ARJudge, as detailed below. (1) **w/o FT**: we replace the fine-tuned Analyzer with the same tuning-free model as the Refiner and prompt the model to generate evaluation questions and conduct the multi-faceted evaluation. (2) **w/o FT&MF**: we apply the model as in the w/o FT setting, generating Chain-of-Thought (CoT) evaluations directly. (3) **w/o Refine**: we retain the fine-tuned Analyzer and make slight modifications to the prompt for the Refiner to directly output the label of the better response.

The ablation results are shown in Table 3. We observe accuracy drops across all test sets with the ablation variants, indicating the effectiveness of each component in ARJudge. Specifically, fine-tuning significantly enhances a general LLM's evaluation capability, enabling it to propose reasonable evaluation questions and analyze responses accordingly. Evaluation questions help the LLM focus on relevant aspects and enhance its evaluation performance. Interestingly, we find that the effects of refinement differ between the fine-tuned and tuning-free Analyzer. In JudgeLM Eval, PandaLM Eval, Auto-J, and MTBench, the refinement keeps evaluation accuracy under the fine-tuned Analyzer's analysis (w/o Refine vs. ARJudge) but significantly decreases it under the tuning-free Analyzer's analysis (Qwen2.5-7B vs. w/o FT&MF). It may be related to the controversial phenomenon that LLMs cannot truly self-correct (Huang et al., 2024b). Additionally, for challenging samples in LLMBar, refinement significantly strengthens the performance of the fine-tuned and tuning-free ones.

4.6 Capability to Evaluate Using Code

Code-driven analysis plays a crucial role in robustly verifying the objective requirements of instructions. To assess the effectiveness of code-driven analy-

sis, we use the execution results of the IFEval official code as a benchmark and compute the **Consistency** between its judgment (Loose or Strict) and that of other models. We compare ARJudge with GPT-4o, Claude-3.5-Sonnet, and Qwen2.5-7B-Instruct. These three models are prompted to make judgments in a zero-shot manner. As shown in Figure 3, our framework achieves a significant improvement over the backbone model, Qwen2.5-7B-Instruct, with the help of code-driven analysis. Moreover, ARJudge performs comparably to GPT-4o and Claude-3.5-Sonnet, demonstrating its potential as a viable alternative. Notably, the execution success rate of the generated code is 100%.

4.7 Effect of Increasing Analysis Quantity

We extend our analysis by scaling up the number of question sampling attempts, exploring the effect of increasing analysis quantity. We set the temperature to 0.2 to sample evaluation questions multiple times, ensuring diversity in the generated questions. As shown in Figure 4, evaluation accuracy improves with more analyses for most datasets, including JudgeLM Eval, Auto-J Eval, PandaLM Eval, and MTBench. The highest accuracy is achieved with four or five rounds of question sampling and their combined analysis. However, in the LLMBar series, additional analysis had little or even a negative impact on accuracy. This may be because the Analyzer has greater uncertainty about the evaluation samples in these sets, and additional analysis further amplifies this uncertainty.

4.8 Generalization of Evaluation Capability

To further demonstrate the generalization of evaluation capability, we compute the ratio of judgment change after refining as shown in Table 4. Combining Table 3 and 4, we observe that the Refiner maintains evaluation performance in JudgeLM Eval, PandaLM Eval, Auto-J Eval, and MTBench, while significantly increasing it in the LLMBar series. This indicates that re-analysis improves the generalization of evaluation capability, especially in handling unseen challenging samples.

5 Case Studies

We show an example of a multi-faceted evaluation generated by ARJudge in Figure 5. Given an instruction and two responses, the Analyzer first generates three evaluation questions and the corresponding multi-faceted analyses. The last question is analyzed by constructing a Python function and assessing execution feedback to determine requirement completeness. Then, the Refiner reviews the preliminary analysis and refines it by reconsidering the instruction's requirements.

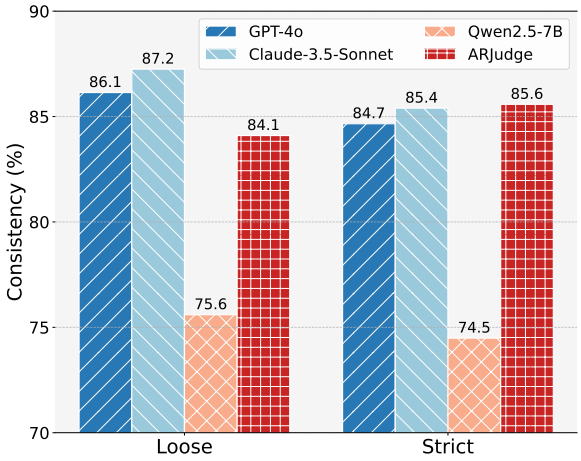

Figure 3: Results on the consistency between code-driven evaluation and IFEval evaluation. "Loose" and "Strict" are two judgment criteria in IFEval.

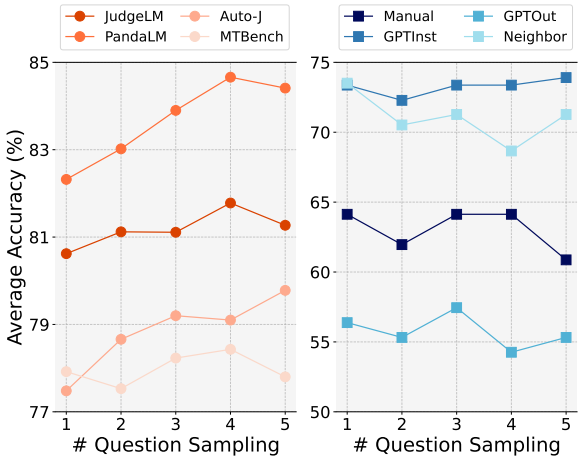

Figure 4: Evaluation results with increasing analyses. The right displays the results of four subsets in LLMBar.

Models	JudgeLM	PandaLM	Auto-J	MTBench	LLMBar
W→C	3.9	4.4	2.3	2.1	7.8
C→W	4.6	4.8	3.4	2.9	3.6

Table 4: Ratio of change after refining. "W→C" denotes a judgment changing from wrong to correct after refinement, while "C→W" denotes the opposite.

6 Related Work

6.1 Tuning-Free Generalist Evaluators

Tuning-free generalist evaluators leverage the inherent capabilities of large language models (LLMs) to assess responses through the use of carefully

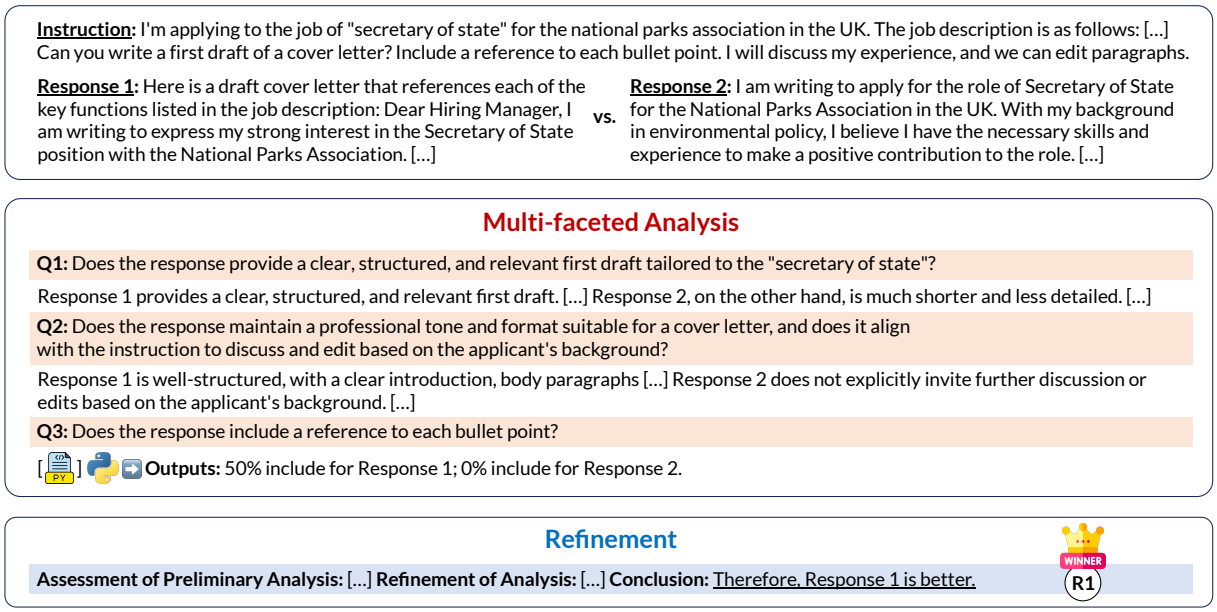

Figure 5: An example of evaluation generated by ARJudge.

designed prompts, offering exceptional flexibility and scalability. Various techniques have been employed to enhance the accuracy of these evaluations, such as in-context learning (Fu et al., 2023; Lin and Chen, 2023)), adding task-specific criteria (Kotonya et al., 2023; Zhuo, 2024), and Chain-of-Thought analysis (Liu et al., 2023; Zhuo, 2024)).

Despite their versatility, tuning-free evaluators often suffer from biases such as position bias (Raina et al., 2024; Wang et al., 2023a; Zheng et al., 2023b) and verbosity bias (Khan et al., 2024; Ye et al., 2024), which can skew evaluation outcomes. Methods like response-adapted references (Zhang et al., 2024), multi-agent collaboration (Xu et al., 2023), and divide and conquer (Saha et al., 2024; Li et al., 2023) have been proposed to mitigate these issues, improving the fairness and reliability of LLM-based evaluations.

6.2 Specialized Fine-Tuned Evaluators

While tuning-free approaches provide flexibility, specialized fine-tuned evaluators are explicitly trained on human-labeled preference data to achieve higher accuracy and domain-specific reliability. These models undergo supervised fine-tuning or reinforcement learning-based optimization to align their evaluations more closely with expert judgments (Li et al., 2024b; Wang et al., 2024b; Kim et al., 2024a,b; Xie et al., 2024).

While fine-tuned evaluators offer improved accuracy, they face notable challenges in scalability and generalization (Huang et al., 2024a). Unlike tuning-free approaches, which can adapt to new tasks with minimal configuration, fine-tuned models require ongoing updates through methods such as supervised fine-tuning or direct preference optimization (Rafailov et al., 2024). To remain effective amidst evolving benchmarks (Zheng et al., 2023a; Zeng et al., 2024), Auto-J (Li et al., 2024b) leverages a large dataset of scoring and preference annotations while incorporating dynamic in-context learning techniques, such as few-shot prompting, to enhance adaptability. Similarly, FLAMe (Vu et al., 2024) combines fine-tuning on labeled preference data with large-scale multitask instruction tuning, enabling it to dynamically adapt to new evaluation criteria while maintaining flexibility.

7 Conclusion

This work proposes a novel evaluation framework, ARJudge, which adaptively designs evaluation criteria and performs multi-faceted evaluation in both text and code. A new Composite Analysis Corpus, designed for both criteria generation and multi-faceted analysis, is developed to train ARJudge. Extensive experiments demonstrate the superiority and robustness of our framework across diverse evaluation benchmarks. Notably, with code-driven analyses, ARJudge gains strong evaluation capabilities for assessing instruction following. Future studies can explore the effective use of more tools, such as a search engine, to improve evaluation honesty and mitigate hallucination.

Limitations

While our framework outperforms various baseline approaches in LLM evaluation, there is still room for improvement. Our method is limited to using code to enhance evaluation robustness and does not consider additional tools such as search engines or specialized agents. Furthermore, our approach partially relies on the LLM's own reasoning ability for evaluation. If the LLM itself lacks strong reasoning capabilities, the effectiveness of refinement may be limited. Additionally, our evaluation is restricted to pairwise comparisons and does not enhance the model's ability to score single responses. Although single-response scoring can be achieved by modifying the Refiner's prompt, its accuracy has not been properly aligned.

References

Anthropic. 2024. Claude 3.5 Sonnet. https://anthropic.com/news/claude-3-5-sonnet.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

Jinlan Fu, See-Kiong Ng, Zhengbao Jiang, and Pengfei Liu. 2023. Gptscore: Evaluate as you desire. In *North American Chapter of the Association for Computational Linguistics*.

Alex Havrilla. 2023. synthetic-instruct-gptj-pairwise.

Xinyu Hu, Li Lin, Mingqi Gao, Xunjian Yin, and Xiaojun Wan. 2024. Themis: A reference-free NLG evaluation language model with flexibility and interpretability. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 15924–15951. Association for Computational Linguistics.

Hui Huang, Yingqi Qu, Xingyuan Bu, Hongli Zhou, Jing Liu, Muyun Yang, Bing Xu, and Tiejun Zhao. 2024a. An empirical study of llm-as-a-judge for llm evaluation: Fine-tuned judge model is not a general substitute for gpt-4. *Preprint*, arXiv:2403.02839.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024b. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Jiaming Ji, Mickel Liu, Josef Dai, Xuehai Pan, Chi Zhang, Ce Bian, Boyuan Chen, Ruiyang Sun, Yizhou Wang, and Yaodong Yang. 2023. Beavertails: Towards improved safety alignment of LLM via a human-preference dataset. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Pei Ke, Bosi Wen, Andrew Feng, Xiao Liu, Xuanyu Lei, Jiale Cheng, Shengyuan Wang, Aohan Zeng, Yuxiao Dong, Hongning Wang, Jie Tang, and Minlie Huang. 2024. Critiquellm: Towards an informative critique generation model for evaluation of large language model generation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 13034–13054. Association for Computational Linguistics.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R Bowman, Tim Rocktäschel, and Ethan Perez. 2024. Debating with more persuasive llms leads to more truthful answers. In *Forty-first International Conference on Machine Learning*.

Seungone Kim, Jamin Shin, Yejin Choi, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Seongjin Shin, Sungdong Kim, James Thorne, and Minjoon Seo. 2024a. Prometheus: Inducing fine-grained evaluation capability in language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Seungone Kim, Juyoung Suk, Shayne Longpre, Bill Yuchen Lin, Jamin Shin, Sean Welleck, Graham Neubig, Moontae Lee, Kyungjae Lee, and Minjoon Seo. 2024b. Prometheus 2: An open source language model specialized in evaluating other language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 4334–4353. Association for Computational Linguistics.

Neema Kotonya, Saran Krishnasamy, Joel Tetreault, and Alejandro Jaimes. 2023. Little giants: Exploring the potential of small llms as evaluation metrics in summarization in the eval4nlp 2023 shared task. In *Proceedings of the 4th Workshop on Evaluation and Comparison of NLP Systems*, pages 202–218.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2024a. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *CoRR*, abs/2411.16594.

Junlong Li, Shichao Sun, Weizhe Yuan, Run-Ze Fan, Hai Zhao, and Pengfei Liu. 2024b. Generative judge for evaluating alignment. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Zongjie Li, Chaozheng Wang, Pingchuan Ma, Daoyuan Wu, Shuai Wang, Cuiyun Gao, and Yang Liu. 2023. Split and merge: Aligning position biases in large language model based evaluators. *arXiv preprint arXiv:2310.01432*.

Yen-Ting Lin and Yun-Nung Chen. 2023. Llm-eval: Unified multi-dimensional automatic evaluation for open-domain conversations with large language models. In *Proceedings of the 5th Workshop on NLP for Conversational AI (NLP4ConvAI 2023)*, pages 47–58.

Minqian Liu, Ying Shen, Zhiyang Xu, Yixin Cao, Eunah Cho, Vaibhav Kumar, Reza Ghanadan, and Lifu Huang. 2024. X-eval: Generalizable multi-aspect text evaluation via augmented instruction tuning with auxiliary evaluation aspects. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 8560–8579. Association for Computational Linguistics.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2021. Webgpt: Browser-assisted question-answering with human feedback. *CoRR*, abs/2112.09332.

OpenAI. 2024. Hello GPT-4o. `https://openai.com/index/hello-gpt-4o/`.

Qwen. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

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.

Vyas Raina, Adian Liusie, and Mark Gales. 2024. Is llm-as-a-judge robust? investigating universal adversarial attacks on zero-shot llm assessment. *arXiv preprint arXiv:2402.14016*.

Swarnadeep Saha, Omer Levy, Asli Celikyilmaz, Mohit Bansal, Jason Weston, and Xian Li. 2024. Branch-solve-merge improves large language model evaluation and generation. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 8352–8370. Association for Computational Linguistics.

Tu Vu, Kalpesh Krishna, Salaheddin Alzubi, Chris Tar, Manaal Faruqui, and Yun-Hsuan Sung. 2024. Foundational autoraters: Taming large language models for better automatic evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17086–17105.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023a. Large language models are not fair evaluators. *arXiv preprint arXiv:2305.17926*.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. 2024a. MINT: evaluating llms in multi-turn interaction with tools and language feedback. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yidong Wang, Zhuohao Yu, Wenjin Yao, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, Wei Ye, Shikun Zhang, and Yue Zhang. 2024b. Pandalm: An automatic evaluation benchmark for LLM instruction tuning optimization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

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

Tinghao Xie, Xiangyu Qi, Yi Zeng, Yangsibo Huang, Udari Madhushani Sehwag, Kaixuan Huang, Luxi He, Boyi Wei, Dacheng Li, Ying Sheng, Ruoxi Jia, Bo Li, Kai Li, Danqi Chen, Peter Henderson, and Prateek Mittal. 2024. Sorry-bench: Systematically evaluating large language model safety refusal behaviors. *CoRR*, abs/2406.14598.

Zhenran Xu, Senbao Shi, Baotian Hu, Jindi Yu, Dongfang Li, Min Zhang, and Yuxiang Wu. 2023. Towards reasoning in large language models via multi-agent peer review collaboration. *arXiv preprint arXiv:2311.08152*.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, et al. 2024. Justice or prejudice? quantifying biases in llm-as-a-judge. In *Neurips Safe Generative AI Workshop 2024*.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2024. Evaluating large language models at evaluating instruction following. In *The Twelfth International Conference on Learning*

Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net.

Qiyuan Zhang, Yufei Wang, Tiezheng Yu, Yuxin Jiang, Chuhan Wu, Liangyou Li, Yasheng Wang, Xin Jiang, Lifeng Shang, Ruiming Tang, et al. 2024. Reviseval: Improving llm-as-a-judge via response-adapted references. *arXiv preprint arXiv:2410.05193*.

Yidan Zhang and Zhenan He. 2024. Large language models can not perform well in understanding and manipulating natural language at both character and word levels? In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 11826–11842. Association for Computational Linguistics.

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. 2023a. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li13, Eric P Xing35, et al. 2023b. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

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. *CoRR*, abs/2311.07911.

Lianghui Zhu, Xinggang Wang, and Xinlong Wang. 2023. Judgelm: Fine-tuned large language models are scalable judges. *CoRR*, abs/2310.17631.

Terry Yue Zhuo. 2024. Ice-score: Instructing large language models to evaluate code. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2232–2242.

A Training Settings

We train Qwen2.5-7B-Instruct[2] to perform as the Analyzer. The number of training samples in the Composite Analysis Corpus is around 25K, including 7.7K evaluation question generation samples, 6K code-driven analysis samples, and 11K text-based analysis samples. The corpus is constructed based on instructions from Auto-J[3] (Li et al., 2024b). We train it for 2 epochs with a global batch size of 96 and we save checkpoints for every 50 steps. The learning rate is set to 1e-5. We use DeepSpeed ZeRO3 and FlashAttention to reduce computational memory usage. The training is implemented on 6 computing devices. We use Pytorch with the 2.4.0 version, Transformers with the 4.44.2 version, and deepspeed with the 0.14.4 version.

[2]https://huggingface.co/Qwen/Qwen2.5-7B-Instruct

[3]https://github.com/GAIR-NLP/auto-j

B Prompt Templates

Prompt templates used for dataset construction are shown in Figure 6, Figure 7, Figure 8, Figure 9, and Figure 10.

Prompt templates used for the Analyzer and Refiner of our ARJudge are shown in Figure 11, Figure 12, and Figure 13.

Prompt for evaluation question geneartion

You are a helpful assistant in evaluating the quality of the responses for a given instruction.
Please propose at most three concise evaluation questions about whether a potential response is a good response for a given instruction.
Another assistant will evaluate different aspects of the response by answering all the questions.

Rules of the evaluation:
1. You should prioritize evaluating whether the response honestly/precisely/closely executes the instruction.
2. Responses should NOT contain more/less than what the instruction asks for, as such responses do NOT precisely execute the instruction.

Requirements for Your Output:
The evaluation questions should **specifically** target the given instruction instead of some general standards, so the questions may revolve around key points of the instruction. Questions are presented from most important to least important. You should directly give the questions without any other words. Format is "Questions:\n1. {question1}\2. {question2}...".

Instruction:
{instruction}

Reference Response 1:
{response1}

Reference Response 2:
{response2}

Reference Response 3:
{response3}

Figure 6: Prompt template for evaluation question generation.

Prompt for constraint geneartion

You are an expert for writing constraints. These constraints can be clearly and objectively check whether they have been followed correctly.

Examples of Verifiable Constraint Types:
1. Keywords (Include Keywords: Include keywords {keyword1}, {keyword2} in your response; Keyword Frequency: In your response, the word word should appear {N} times; Forbidden Words: Do not include keywords {forbidden words} in the response; Letter Frequency: In your response, the letter {letter} should appear {N} times; etc.)
2. Language (Response Language: Your ENTIRE response should be in {language}, no other language is allowed; etc.)
[...]
7. Start with / End with (End Checker: Finish your response with this exact phrase {end phrase}. No other words should follow this phrase; Quotation: Wrap your entire response with double quotation marks; etc.)
8. Punctuation (No Commas: In your entire response, refrain from the use of any commas; etc.)

Examples of Instruction and Generated Constraints:
Instruction:
Write a limerick about Hannah, a college student, doing an internship at a coffee company. Make sure that her father would love the limerick.
Constraints:
1. Include the words "intern" and "grow".
2. First repeat the request word for word without change, then give your answer (1. do not say any words or characters before repeating the request; 2. the request you need to repeat does not include this sentence)
[...]

Requirements for Your Output:
Please write additional different 8 verifiable constraints for the following instruction. You should randomly select verifiable constraint types from the above examples of verifiable constraint types. The constraint form can be arbitrary like examples of instruction and generated constraints. The constraints should be tailored to the context of the instruction. Format is "Constraints:\n1. {constraint1}\2. {constraint2}...". Don't state the type name in constraints.

Instruction:
{instruction}

Figure 7: Prompt template for objective constraint generation.

Prompt for text-based evaluation

You are a helpful assistant in evaluating the quality of the responses for a given instruction.
Select the better response, Response 1 or Response 2, based on the evaluation questions. The two responses are generated by two different AI chatbots respectively.

Rules of the evaluation:
1. You should prioritize evaluating whether the response honestly/precisely/closely executes the instruction.
2. Responses should NOT contain more/less than what the instruction asks for, as such responses do NOT precisely execute the instruction.
3. You should avoid any potential bias and your judgment should be as objective as possible. For example, the order in which the responses were presented should NOT affect your judgment, as Response 1 and Response 2 are **equally likely** to be the better.

Requirements for Your Output:
1. You should do the evaluation based on thinking about the evaluation questions.
2. You should first provide an analysis of each response based on the comparison between them, and then end your output with either "Therefore, Response 1 is better" or "Therefore, Response 2 is better," verbatim.
3. Do NOT say both/neither are good. Do NOT say "Response 1 is better" or "Response 2 is better" at the beginning.
4. You should do reasoning and thinking **before** claiming which is better.
5. Format is "Analysis of Question 1: {analysis}\n\nAnalysis of Question 2: {analysis}\n\nAnalysis of Question 3: {analysis}\n\n..Therefore, Response {idx} is better." Don't use comment symbols like "*".

Instruction:
{instruction}

Response 1:
{response1}

Response 2:
{response2}

Evaluation Questions:
{criteria}

Figure 8: Prompt template for text-based evaluation.

Prompt for code-driven evaluation

You are a helpful assistant in writing evaluation functions in Python to evaluate whether a response aligns with the given evaluation question.

Reference Response 1:
{response1}

Reference Response 2:
{response2}

Reference Response 3:
{response3}

Evaluation question:
{question}

Requirements for Your Output:
1. You should write a Python function named "evaluate" to evaluate whether an input string "response" aligns with the given evaluation question. Use ".format()" instead of f-string. Use escape character "\\n" if necessary.
2. You should comprehensively understand the question and write Python function suitable for evaluating all responses.
3. "Starts with" or "Ends with" mean the first or last several characters of the response should meet some requirements. The whitespaces at the beginning or end of sentences can be ignored.
4. The Python function must print not only the main results but also intermediate calculations or additional information that helps to understand the logic of the code.
5. DO NOT print out the original responses.
6. The execution lines are as follows:

```python
# Example usage
responses = [response1, response2, response3]
for i, response in enumerate(responses, 1):
  print("\nEvaluating Response {}:".format(i))
  evaluate(response)
```

Figure 9: Prompt template for code-driven evaluation.

Prompt for reverse validation

You are an expert in converting Python evaluation function code into the corresponding explanation text. I will provide the evaluation function code. Please strictly follow the code to convert it into the corresponding explanation text.

Example:
Evaluation Function:
```python
{function}
```

Explanation:
{explanation}

Please convert the following evaluation function into explanation stored in a list:
Evaluation Function:
```python
{function}
```

Figure 10: Prompt template for reverse validation.

Prompt for the Analyzer: question

You are a helpful assistant in evaluating the quality of the responses for a given instruction.
Propose 3-6 concise evaluation questions about whether a potential response is a good response for a given instruction.

Requirements for Your Output:
The evaluation questions should **specifically target the given instruction instead of some general standards, so the questions may revolve around key points of the instruction.**

Instruction:
{instruction}

Figure 11: Prompt template for question generation of the Analyzer.

Prompt for the Analyzer: multi-faceted evaluation

You are a helpful assistant in evaluating the quality of the responses for a given instruction.
Evaluate the following two responses based on the given evaluation question.

Requirements for Your Output:
1. You should do the evaluation based on thinking about the evaluation question.
2. You should provide an analysis of each response based on the comparison between them.
3. You can write Python code to do the analysis for some verifiable evaluation question.

Instruction:
{instruction}

Response 1:
{response1}

Response 2:
{response2}

Evaluation Question:
{criteria}

Figure 12: Prompt template for multi-faceted analysis of the Analyzer.

Prompt for the Refiner

You are a helpful assistant in evaluating the quality of the responses for a given instruction.
Select the better response, Response 1 or Response 2. The two responses are generated by two different AI chatbots respectively.

Rules of the evaluation:
1. You should prioritize evaluating whether the response honestly/precisely/closely executes the instruction.
2. Responses should NOT contain more/less than what the instruction asks for, as such responses do NOT precisely execute the instruction.
3. You should avoid any potential bias and your judgment should be as objective as possible. For example, the order in which the responses were presented should NOT affect your judgment, as Response 1 and Response 2 are **equally likely to be the better.**

Requirements for your output:
1. You should provide a detailed explanation of your analysis, and then always end your response with either "Therefore, Response 1 is better." or "Therefore, Response 2 is better." verbatim.
2. Do NOT say both/neither are good. Do NOT say "Response 1 is better" or "Response 2 is better" at the beginning.
3. You should do reasoning and thinking **before claiming which is better.**

Instruction:
{instruction}

Response 1:
{response1}

Response 2:
{response2}

Analysis by preliminary evaluators:
{analysis}

Your analysis (Give a detailed explanation: step 1: what do you think of the analysis by other evaluators?; step 2: can you refine the analysis by reconsidering the requirements of the instruction?; step 3: which response is better?):

Figure 13: Prompt template for refinement of the Refiner.