Divide-Verify-Refine: Can LLMs Self-Align with Complex Instructions?

Xianren Zhang[1], **Xianfeng Tang**[2], **Hui Liu**[2], **Zongyu Wu**[1], **Qi He**[2]
Dongwon Lee[1], **Suhang Wang**[1]
[1]The Pennsylvania State University [2]Amazon
{xzz5508,dongwon,zzw5373,szw494}@psu.edu, {liunhu,xianft}@amazon.com

Abstract

Recent studies show LLMs struggle with complex instructions involving multiple constraints (e.g., length, format, sentiment). Existing works address this issue by fine-tuning, which heavily relies on fine-tuning data quality and is computational expensive. An alternative is leveraging LLMs' self-correction to refine responses for better constraint adherence. However, this is limited by the feedback quality, as LLMs cannot generate reliable feedback or detect errors. Moreover, its effectiveness relies on few-shot examples illustrating response modifications. As constraints in complex instructions are diverse, manually crafting such examples for each constraint type can be labor-intensive and sub-optimal. To address these two challenges, we propose the **Divide-Verify-Refine (DVR)** framework with three steps: (1) **Divide** complex instructions into single constraints and prepare appropriate tools; (2) **Verify** responses using tools that provide rigorous check and textual guidance (e.g., Python toolkit for format checks or pre-trained classifiers for content analysis); (3) **Refine**: To maximize refinement effectiveness, we propose dynamic few-shot prompting, where a refinement repository collects successful refinements, and these examples are selectively retrieved for future refinements. Recognizing the lack of complexity in existing datasets, we create a new dataset of complex instructions. DVR doubles Llama3.1-8B's constraint adherence and triples Mistral-7B's performance. The code is available here.

1 Introduction

Large language models (LLMs), like ChatGPT, have shown significant improvements across various language tasks (Touvron et al., 2023; Wang et al., 2024b,a; Zhang et al., 2025). The success of LLMs relies on the ability to execute complex instructions. Failures to follow instructions can result in unintended outputs, which may have severe consequences (Mu et al., 2023; Zhou et al., 2023; Tseng et al., 2024). This issue becomes critical when LLMs are deployed in high-stakes environments, such as legal documentation or technical writing. For example, when drafting legal contracts, LLMs must strictly adhere to constraints related to format, specific terminology, and precise language usage to avoid misinterpretations or legal liabilities. Similarly, in technical writing, adhering to strict format guidelines, word limits, and inclusion of essential technical terms is critical to ensure clarity and compliance with industry standards.

Recent studies show that LLMs, especially open-source ones, struggle to follow complex instructions with multiple constraints like response length or formatting (He et al., 2024a; Jiang et al., 2024b; Chen et al., 2024b). While this issue is well recognized, research on enhancing LLMs' constraint adherence ability is still limited, with most efforts focused on evaluating the ability (Jiang et al., 2024b; Chen et al., 2024b). Few studies improve LLMs' constraint-following via fine-tuning (He et al., 2024a; Sun et al., 2024; Li et al., 2024). For example, He et al. 2024a adopt a teacher model (e.g., GPT-4) to generate data and fine-tune a student model with the generated data to improve its multi-constraint adherence ability. While effective, it requires a large amount of computation resources and heavily depends on the generated data quality.

In contrast, the concept of "self-correction" offers an alternative approach, where LLMs autonomously correct their responses (Madaan et al., 2024; Shinn et al., 2024). Self-correction has been applied on other tasks such as question answering (Dhuliawala et al., 2023; Shinn et al., 2024) or mathematics (Madaan et al., 2024), where an LLM will evaluate its responses, give feedback, and further refine responses. However, *whether LLMs can effectively self-align with diverse and complex constraints remains an open question.* This is particularly important for agent LLMs to be deployed in high-stakes environments. For

Findings of the Association for Computational Linguistics: ACL 2025, pages 13783–13800
July 27 - August 1, 2025 ©2025 Association for Computational Linguistics

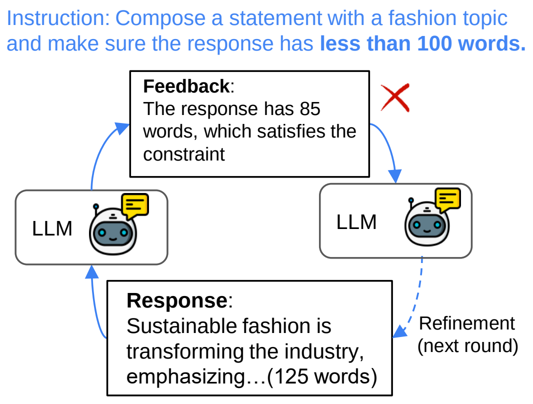

Instruction: Compose a statement with a fashion topic and make sure the response has **less than 100 words.**
Feedback: The response has 85 words, which satisfies the constraint
LLM
LLM
Response: Sustainable fashion is transforming the industry, emphasizing…(125 words)
Refinement (next round)

Figure 1: The LLMs hallucinate and struggle to give reliable feedback.

constraint-following, this self-correction process can be divided into two phases: verification and self-refinement (Fig. 1). During the verification phase, LLMs assess whether their responses align with the specified constraints. If the responses do not align with the constraints, the LLMs will give feedback that pinpoints errors and suggests adjustments. Following this, the self-refinement phase takes place where LLMs use the feedback to refine and improve their responses accordingly.

However, there are several challenges. The first one is *feedback reliability*. Studies show that the performance gains by LLM self-correction is unstable, occasionally even degrading performance in question answering (Huang et al., 2024) and code generation (Olausson et al., 2024). The bottleneck in self-correction lies in the feedback quality (Tyen et al., 2024; Gou et al., 2024; Jiang et al., 2024a). LLMs, including advanced models like GPT-4 and Claude 3, tend to have low recall in detecting LLMs errors, underperforming significantly compared to humans (Kamoi et al., 2024). On the other hand, research reveals that the self-correction performance on reasoning tasks is boosted if the error location is given, indicating LLMs have the self-correction ability given reliable feedback (Tyen et al., 2024). From the constraint-following perspective, LLMs are also not good at checking simple and easy-to-verify constraints. As shown in Fig. 1, given a response, LLMs struggle to accurately count the number of words. The second challenge is *constraint diversity* which lies in the self-refinement process. Given the response and the feedback, LLMs should refine the response according to the feedback. However, to perform this task effectively, a set of representative few-shot examples is needed to demonstrate how to appropriately modify the response (Brown et al., 2020). These constraints can vary widely, from adhering to a length limit to including specific keywords. Each type of constraint needs distinct modifications. For example, meeting a length limit might require removing content, whereas incorporating specific keywords requires adding text. Manually crafting representative few-shot examples for each constraint type is labor-intensive.

To address these challenges, we propose a novel framework named Divide-Verify-Refine (DVR) as shown in Fig. 2. To enhance feedback reliability, we observe that constraints that LLMs struggle to verify can be readily assessed using external tools. These tools include python toolkit for quantitative measures, such as Regular Expressions (re) (Friedl, 2006) and the Natural Language Toolkit (NLTK) (Bird et al., 2009) for counting the number of words, sentences, paragraphs, or bullet points, as well as pre-trained classifiers for content analysis, such as topic and sentiment analysis, which are easily accessible and widely available on hugging face (Antypas et al., 2022; Loureiro et al., 2022). We enhance LLMs by enabling interaction with external tools. First, we instruct LLMs to break down complex instructions into individual constraints and assign an appropriate tool to each. These tools then rigorously verify the LLM's response and provide textual guidance for refinement if any constraints are violated. To address the problem of constraint diversity, we propose the *dynamic few-shot prompting*. Since the verification reliability is ensured by external tools, we incorporate a novel refinement repository, which serves as a memory module to collect and store successful refinements for future use. When a new refinement task arises, we select few-shot examples with the same constraint type to maximize refinement effectiveness.

Our **main contributions** are: (i) Our framework enhances feedback reliability by integrating easily accessible tools that provide strict verification and textual guidance for LLMs. (ii) To maximize the refinement effectiveness, we propose *dynamic few-shot prompting* with a refinement repository that stores successful refinements. This enables LLMs to learn from past experiences and retrieve more similar few-shot examples for future refinements, improving refinement effectiveness. (iii) Most benchmarks only contain 1-2 constraints (Chen et al., 2024b). We construct a new complex instruction dataset with instructions containing 1-6 constraints.

2 Related Work

Instruction-Following of LLMs. Recent studies show that LLMs struggle to follow complex instructions, especially as the number of constraints increases (Dubois et al., 2024; Zhou et al., 2023; Jiang et al., 2024b; Chen et al., 2024b; Zhou et al., 2023; He et al., 2024b; Lin et al., 2025). To address this challenge, some works (Chen and Wan, 2023; Sun et al., 2024; Wang et al., 2024c; He et al., 2024a; Dong et al., 2024) generate instructions and responses with advanced LLMs (e.g., GPT4) and then use the generated data to fine-tune the student LLMs. Among them, He et al. (2024a) focuses on improving LLMs' alignment with multiple constraints. They iteratively refine student model responses using GPT-4 as a teacher. The student model is fine-tuned on both intermediate modifications and final refined responses. Although effective, these methods rely heavily on the teacher model and are resource-consuming. Different from previous methods, our framework uses ins-context learning with tool interaction to effectively refine unsatisfactory responses, offering a more practical solution.

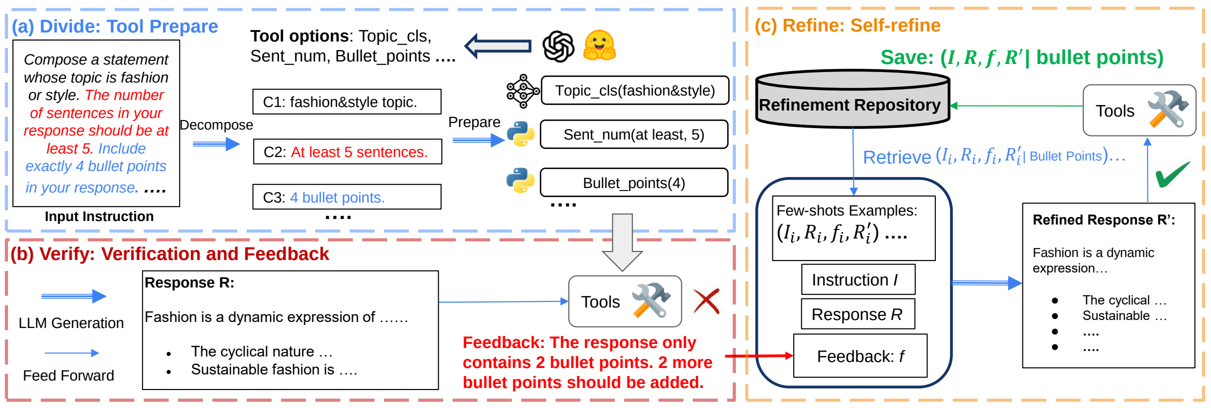
Self-Correction of LLMs. Self-correction is a framework where LLMs refine their responses during inference by reflecting on their initial responses (Shinn et al., 2024; Madaan et al., 2024). This process has two phases. Initially, LLMs are prompted to analyze and provide feedback on their responses. Subsequently, based on the feedback LLMs refine the responses to correct their mistakes. However, recent studies report negative results indicating that LLMs cannot self-correct their own mistakes (Hong et al., 2024; Tyen et al., 2024; Kamoi et al., 2024; Gou et al., 2024). A study (Kamoi et al., 2024) reveals that top LLMs like GPT-4 and Claude 3 have low recall in detecting LLM errors, with LLMs significantly underperforming compared to humans. Additionally, feedbacks provided by LLM self-correction tend to hallucinate and lack reliability. This unreliability suggests that even when errors are detected, the guidance offered for corrections may be incorrect or misleading. (Hong et al., 2024) find that LLMs struggle to accurately identify logical fallacies, casting doubt on their inherent ability to detect errors and conduct self-verification reasoning effectively. However, the self-correction performance on reasoning tasks is boosted if the error location is given (Tyen et al., 2024). All these observations indicate that LLMs are not reliable in analyzing their responses and a more reliable feedback mechanism is needed to pinpoint the mistakes. More introduction to related works is in Appendix A.8.

3 The Proposed Framework: DVR

As shown in Fig. 2, we propose the Divide-Verify-Refine (DVR) framework, which consists of three modules: (a) *Divide* instructions and prepare tools accordingly, (b) *Verify* responses and provide feedback, and (c) *Refine* and store responses in a repository. First, the tool preparation module aims to identify constraints, select appropriate tools, and fill out parameters. In this module, LLMs first decompose the complex instructions into single constraints. For each single constraint, the LLMs will prepare appropriate tools for verification. Second, the prepared tools will verify the response and give detailed textual guidance if the response does not adhere to the constraint. Third, in the self-refinement module, given the textual guidance, LLMs will refine the response to adhere to the target constraint. Since similar few-shot examples usually yield better results, DVR retrieves past refinement experience with the same constraint type few-shot examples. The successfully refined response will be stored for future use. Next, we introduce each module in detail.

3.1 Divide: Tool Preparation

To provide accurate feedback, we propose to adopt tools for verification. These tools are widely available and easily accessible (Qin et al., 2024): (i) There are abundant publicly available tools online. For instance, over 16,000 tools are accessible through RESTful API collections. Additionally, libraries like Regular Expressions (re) and the Natural Language Toolkit (NLTK) (Bird et al., 2009) are commonly used for checking text format, patterns, and length constraints. Hugging Face also provides numerous open-source models, such as topic classifiers (Antypas et al., 2022) and sentiment classifiers (Loureiro et al., 2022), which can be directly integrated and utilized by LLMs; and (ii) When no suitable tool is available, existing works have shown that LLMs can be effectively used to generate reliable tools through code synthesis (Guo et al., 2024). Since tool generation is a one-time cost, it can be efficiently handled by advanced models. In our setup, we use GPT-4o to generate tools, and as shown in Section 4.8, our



(a) Divide: Tool Prepare
Compose a statement whose topic is fashion or style. The number of sentences in your response should be at least 5. Include exactly 4 bullet points in your response. ….
Input Instruction
Decompose
Tool options: Topic_cls, Sent_num, Bullet_points ….
C1: fashion&style topic.
C2: At least 5 sentences.
C3: 4 bullet points.
Prepare
Topic_cls(fashion&style)
Sent_num(at least, 5)
Bullet_points(4)
(b) Verify: Verification and Feedback
LLM Generation
Feed Forward
Response R:
Fashion is a dynamic expression of ……
The cyclical nature …
Sustainable fashion is ….
Tools
Feedback: The response only contains 2 bullet points. 2 more bullet points should be added.
(c) Refine: Self-refine
Save: (I, R, f, R′| bullet points)
Refinement Repository
Tools
Retrieve (I_i, R_i, f_i, R′_i| Bullet Points)…
Few-shots Examples: (I_i, R_i, f_i, R′_i) ….
Instruction I
Response R
Feedback: f
Refined Response R′:
Fashion is a dynamic expression…
The cyclical …
Sustainable …

Figure 2: The DVR framework: (a) Divide: The LLMs decompose constraints and instantiate tools for each constraint, (b) Verify: Tools will give feedback on the response, (c) Refine: The refinement repository provides past refinement process as few-shot examples. The current refinement process will be stored in the repository.

experiments confirm that these generated tools are highly reliable.

Given an input instruction I, the LLM $\mathcal{M}$ first decomposes it into a series of individual constraints. We use a decomposition prompt p_{decomp} asking LLMs for decomposition. With input instruction and decomposition prompt, LLM then generates a set of decomposed constraints: $\mathcal{M}(p_{decomp}, I) \rightarrow \{c_i\}_{i=1,2,3...}$, where c_i is the i-th single constraint. For each constraint c_k, the LLM determines the appropriate tool by matching c_k to a tool t_k from the predefined toolset: $\mathcal{M}(p_{select}, c_k) \rightarrow t_k$, where $t_k \in \{t_i\}_{i=1,2,3...}$ is the selected tool for the constraint c_k. The prompts for decomposition p_{decomp} and tool selection p_{select} are in Fig. 7 in Appendix. After selecting the tools, the LLM sets the necessary parameters for each tool, such as specifying the required number of bullet points or the desired sentiment for the response. Finally, all tools relevant to instruction I are compiled into the set $T_I = \{t_i\}_{i=1,2,3...}$, ready to be utilized in the subsequent verification and feedback phase.

3.2 Verify: Verification and Textual Guidance

Given the instruction, the LLM will first generate the initial response $R_0 = \mathcal{M}(p_{generate}, I)$, where $p_{generate}$ is the prompt for generation (detailed in Fig. 7 in Appendix). We denote the current response as R and $R = R_0$ for the first round of refinement and will be updated to the refined response in subsequent rounds. The current response is verified by each tool in toolset T_I as follows:

$$f_i = t_i(R), \forall t_i \in T_I, \tag{1}$$

where f_i is the feedback from tool t_i for constraint c_i. If the response adheres to the constraint, the feedback is a boolean value "true". Otherwise, f_i is a textual feedback that first identifies the error in the response and then suggests modification. For example, as shown in Fig. 2, the tool "Bullet_points(4)" counts the number of bullet points in the response and outputs "true" if there are 4 bullets; while the response only contains 2 bullets. It finds that the response does not satisfy the constraint and gives out the feedback "The response only contains 2 bullet points. 2 more bullet points should be added." This detailed feedback points out the errors in the response and gives directional information for LLMs to modify the response. We collect all feedback $F_I = \{f_i\}_{i=1,2,3...}$ which will be used to refine the response R.

3.3 Self-refine with Dynamic Few-shot Prompting

In the self-refinement phase, the LLM leverages the textual feedback to refine the response. As constraints vary widely, each type of constraint requires demonstrations with similar constraint types for effective refinement. Manually creating one fixed set of few-shot examples can be sub-optimal. Instead of using a fixed set of few-shot examples, we propose dynamic few-shot prompting where few-shot examples with the same constraint type as the current refinement task are selected from the refinement repository. If the response is successfully refined, this process will be stored in the refinement repository for future use.

Specifically, the refinement process targets one unsatisfied constraint at a time, cycling through a refine-verify-refine loop until all constraints are satisfied. For a given response R and the feedback $f \in F_I, f \neq True$, the refinement response can be written as follows:

$$R' = \mathcal{M}(p_{refine}, s^t, I, R, f), \qquad (2)$$

where p_{refine} is the prompt for refinement (detailed in Fig. 7), $s^t = \{(I_i, R_i, f_i, R'_i)^t\}_{i=1,2,3...}$ is the set of refinement examples selected from the refinement repository Q, which contains refinement examples having the same constraint type associated with f. There might be many refinement examples having the same constraint type with f available in the refinement repository. Retrieval techniques like semantic similarity can be employed to select the most relevant examples. In this paper, we randomly select relevant examples for simplicity and leave more advanced techniques for future work. Some refinement examples are in Fig. 6 in the Appendix.

If the refined response adheres to the constraint, i.e., $t(R') = True$, the current successful refinement process will be stored in the repository as $Q = Q \cup \{(I, R, f, R')^t\}$.

Discussion. Our proposed DVR is a novel approach to enhancing LLMs' ability to follow complex instructions with multiple constraints. The detailed algorithm of DVR is shown in Algorithm 1 in Appendix A.1. By integrating external tools for reliable and detailed textual guidance and a refinement repository for storing successful refinement examples, we provide a scalable and robust framework for improving instruction compliance without the need for extensive retraining. Moreover, the external tools and the refinement repository work jointly. Without reliable feedback, the refinement repository would risk accumulating incorrect or noisy examples, which could deteriorate the performance of LLMs over time. The detailed feedback gives "directional" information, which guides the LLMs to adjust their responses. Compared to directly following complex instructions, decomposing these instructions and selecting the appropriate tools are simpler tasks for LLMs. This inherent advantage allows our DVR to be very effective, as it leverages these easier tasks to build a robust system that enhances LLMs' adherence to constraints.

4 Empirical Validation

In this section, we conduct experiments to answer the following research questions: (**RQ1**) Can our DVR improve the ability of LLMs to follow complex constraints? (**RQ2**) How does the performance of LLMs differ across various types of constraints, and which constraints pose the greatest challenges? (**RQ3**) How does each module of DVR (the tool-assisted verification and the few-shot self-refinement library) individually contribute to improving LLMs' ability to follow constraints?

4.1 Experimental Setup

Datasets. We conduct experiments on two datasets: (i) **CoDI** (Chen et al., 2024b): A dataset of 500 instructions, each with a topic constraint and a sentiment constraint. (ii) **ComplexInstruct**: Due to CoDI's limited complexity, we construct ComplexInstruct, a new dataset of complex instructions. Using CoDI's topic instruction set as seed instructions, we refine them by removing implicit length constraints (e.g., replacing "paragraph" or "sentence" with "text") to avoid conflicts and hidden constraints. Then, we synthesize complex instructions by adding constraints to these seed instructions (Zhou et al., 2023). To generate instructions of different levels, we generate 6,000 complex instructions across six levels (1–6 constraints per instruction, 1,000 instructions per level). The dataset includes 21 constraint types across 8 general categories (e.g., length, punctuation, case changes), with each type expressed in 8 different ways. The detailed information is in Appendix A.3.

Baselines. We compare our method with state-of-the-art baselines, which can be categorized into three main types: (i) Self-reflection based methods, which iteratively improve response via feedback from LLMs reflection, such as **Reflexion** (Shinn et al., 2024); (ii) Prompting based methods, which use different prompting strategies to get the best response, including Branch-solve-Merge (**BSM**) (Saha et al., 2024) and Universal Self-Consistency (**U-SC**) (Chen et al., 2024a); and (iii) Tool based methods, which use external tools for feedback or selection, such as Rejection sampling (**R-Sample**) (Saunders et al., 2022), **React** (Yao et al., 2023), and **CRITIC** (Gou et al., 2024). The details of these baselines are in Appendix A.2.

Implementation. We test on popular open-source models including Mistral-7B, Llama3-8B, Llama3.1-8B and Llama3.1-70B. The temperature of the model is 0.8. We set the number of few-shot demonstrations for initial response generation and self-refinement (without repository) as 5 for our method and every baseline. We use the same set of few-shot demonstrations both for baselines and our method. We also set the maximum number of few-shot demonstrations for refinement (with repository) as 8. We set the number of trials as 5 for our method and every baseline. For the refinement

Table 1: Instruction Satisfaction Rate (ISR) across levels 1 to 6 (Llama-3.1-8B-Instruct). The values in parentheses ($+xx$) indicate the improvement compared to the best performing baseline.

Method	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Vanilla	90.5	76.6	62.5	50.1	35.6	25.3
Reflexion	91.6	78.1	63.7	49.8	35.8	25.7
BSM	90.1	75.3	62.0	47.5	35.5	24.1
U-SC	90.9	76.3	62.4	47.1	36.0	25.8
R-Sample	92.1	86.7	71.1	60.4	49.8	36.3
ReAct	94.2	86.1	72.5	60.7	50.2	37.2
CRITIC	93.8	87.1	75.4	64.4	52.4	43.2
DVR$_{CS}$	94.5 (+0.7)	87.9 (+0.8)	78.4 (+3.0)	69.5 (+5.1)	**60.9** (+8.5)	49.2 (+6.0)
DVR$_{WS}$	**95.2** (+1.4)	**88.7** (+1.6)	**79.2** (+3.8)	**69.7** (+5.3)	60.5 (+8.1)	**49.6** (+6.4)

Table 2: Instruction Satisfaction Rate (ISR) across levels 1 to 6 (Mistral-7B-Instruct-v0.3).

Method	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Vanilla	77.0	55.3	34.1	19.9	12.4	6.3
Reflexion	77.2	55.8	35.1	20.1	12.0	5.8
BSM	78.1	56.2	33.8	19.3	11.3	5.2
U-SC	76.8	56.0	34.3	20.4	12.9	5.8
R-Sample	78.4	58.3	37.6	23.0	13.5	6.8
ReAct	86.0	67.8	46.0	32.5	18.2	10.7
CRITIC	88.9	72.5	55.6	43.5	28.1	18.1
DVR$_{CS}$	94.9 (+6.0)	80.2 (+7.7)	64.1 (+8.5)	49.3 (+5.8)	35.8 (+7.7)	**23.6** (+5.5)
DVR$_{WS}$	**95.0** (+6.1)	**81.3** (+8.8)	**66.6** (+11.0)	**51.4** (+7.9)	**36.4** (+8.3)	23.4 (+5.3)

repository of our DVR, we consider two variants, i.e., warm-start and cold-start. For **warm-start**, we have an additional set of instructions (6000 samples for ComplexInstruct and 500 samples for CoDI). Note that these data samples are totally independent with test set. Our framework will first run on these samples to collect examples to fill the refinement repository. For **cold-start**, since the refinement repository is empty at beginning, we use 5 fixed few-shot examples if there are no examples that can be retrieved from the repository.

Evaluation Metrics. We assess the constraint-following ability by calculating the Instruction Satisfaction Rate (ISR) (Jiang et al., 2024b). Specifically, each single instruction is satisfied when all constraints in that instruction are satisfied. It is calcualted as ISR $= \frac{1}{N}\sum_{i=1}^{N}\prod_{j=1}^{m_i} c_{ij}$, where N is the total number of instructions in the dataset, m_i is the number of constraints in the i-th instruction, $c_{ij} = 1$ if the j-th constraint in i-th instruction is satisfied; otherwise $c_{ij} = 0$.

4.2 RQ1: Constraint-Following Ability

To answer RQ1, we evaluate DVR on two datasets. We evaluate structural constraints (e.g., text length, number of sections, and bullet points) on ComplexInstruct and content constraints (e.g., topic and sentiment constraints) on CoDI respectively. DVR$_{CS}$ and DVR$_{WS}$ are cold-start and warm-start.

Results are shown in Table 1 and Table 2 **(i) Single vs Multi-constraints:** As constraint complexity increases, satisfaction rates drop. While ISR at Level 1 approaches 95%, Level 6 ISR drops to 25% for Llama3.1-8B and 6.3% for Mistral-7B, showing LLMs struggle with multiple constraints even when they handle them individually. **(ii) Intrinsic self-reflection is unreliable:** Reflxion (Shinn et al., 2024), which relies on LLMs to reflect and self-correct, shows minimal improvement over Vanilla, indicating LLMs struggle to identify their own constraint violations. Similar results can be observed on Branch-Solve-Merge and Universal Self-consistence. **(iii) Textual Guidance matters:** ReAct (Yao et al., 2023) and CRITIC (Gou et al., 2024) can be viewed as two variants of DVR, where feedback is provided as boolean signals on the instruction level or specific constraint violations. Compared with ReAct and CRITIC, DVR has better performance, which means detailed analysis and textual guidance can make the refinement more effective. The refinement repository further maximizes the refinement effectiveness. The results in Figure 3 show the distribution of satisfied constraints per instruction at Level 6 difficulty. Our framework shifts the distribution rightward, indicating improved adherence to multiple constraints. Notably, for Mistral-7B, our framework moves the central tendency from satisfying 4 constraints to 5.

Table 3: Performance by Self-training (zero-shot)

Model	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Mistral-7B	60.1	41.7	23.8	14.2	8.4	3.7
Mistral-7B(DPO)	85.0	67.8	47.0	33.9	18.4	12.9
Llama3.1-8B	62.5	38.9	23.2	14.6	9.6	6.3
Llama3.1-8B(DPO)	83.7	69.8	55.6	39.3	28.0	17.0

Self-improving: To enable self-improvement without relying on labeled data or external models, we leverage the output of DVR as training data. The refined response and the original model response are selected as positive-negative a pair if their constraint satisfaction rate gap is over 0.4. These pairs are further used for Direct Preference Optimization (DPO) tuning (Rafailov et al., 2024). This allows the model to iteratively enhance its performance based on its outputs. For fine-tuning, we use Llama3.1-8B and Mistral-7B as the base models and apply LoRA (Hu et al., 2021) with a rank of 32 and an α value of 64. For DPO tuning, β is set as 0.2.

The results, presented in Table 3, demonstrate that DPO-tuned models (DPO-DVR) significantly outperform the vanilla model across all constraint levels, particularly at higher complexity levels. This highlights the effectiveness of the self-training

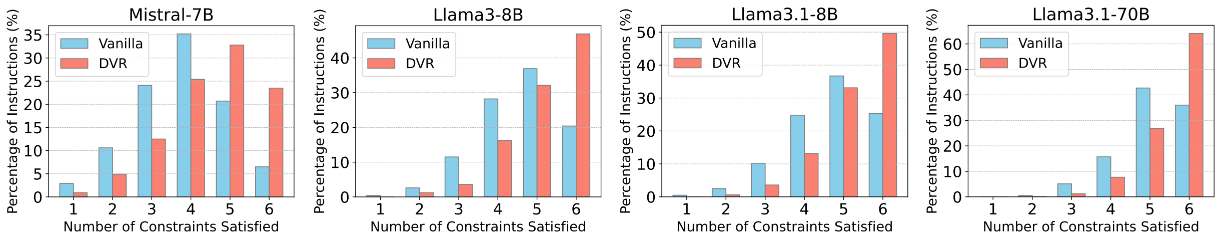


Figure 3: Distribution of satisfied constraints number per instruction (level 6).

Table 4: Comparison Across Constraints Types

	Mistral-7B		Llama3-8B		Llama3.1-8B	
Constraint Type	Vanilla	DVR	Vanilla	DVR	Vanilla	DVR
Detectable Content	76.36	88.90	84.18	96.81	86.29	96.31
Keywords	76.04	84.23	83.84	88.32	84.94	88.77
Punctuation	24.34	72.93	91.03	95.64	97.01	98.04
Case Change	70.08	81.28	81.28	93.38	82.97	90.71
Start End	81.29	90.41	84.88	90.03	84.07	91.92
Detectable Format	69.59	80.70	81.57	89.23	84.69	92.31
Language	69.11	81.80	77.06	88.38	81.96	89.76
Length Constraints	50.42	73.23	65.29	80.85	68.55	83.57

approach in improving constraint adherence.

Additional Results: Additional experimental results on models such as LLaMA 3.1-70B, LLaMA 3-8B, and GPT-4 can be found in Appendix A.4, along with some evaluations on the CoDI (Chen et al., 2024b) and IFeval (Zhou et al., 2023) benchmarks. We also test DVR efficiency by testing the time used for inference and results are in Appendix A.7. Discussion on the influence of DVR on fluency and readability can be found in A.5.

4.3 RQ2: Comparison Across Different Constraint Types

Comparison across different constraint types is shown in Table 4 (warmstart). Coldstart results are provided in Table 10 in Appendix A.4. We have the following observations. (i) Length constraints are the most challenging: Every model struggles with length constraints, which include minimum/maximum word counts, sentence counts, and exact paragraph requirements. This difficulty likely stems from a lack of such constraints in instruction-tuning datasets, making it hard for models to map output structure to length requirements. Additionally, length constraints require models to plan responses from the outset while maintaining coherence and completeness. (ii) Language constraints, which require the use of languages such as Italian, German, or Japanese, are the second most challenging. This might be due to the limited multilingual capabilities of the LLMs. (iii) Punctuation constraint which requires LLMs not to use any commas in their responses, is especially challenging for Mistral-7B. However, our framework improves it significantly and triples the performance from 24% to 73%. DVR's feedback not only verifies correctness but also explicitly highlights the locations of commas, providing precise guidance.

4.4 RQ3: Contribution of Individual Modules

We conduct an ablation study to evaluate the impact of key components. We examine three variants: (i) **w/o Detailed Feedback:**Removes detailed feedback from the tool but retains the refinement repository, which provides relevant few-shot examples showing responses before and after refinement. The repository starts empty (coldstart). (i) **w/o Repository:** Removes the refinement repository, using only five fixed examples for self-refinement. (i) **w/o both:** The refinement repository and detailed feedback are all removed. Figure 4 shows performance gaps between each method and the Vanilla. Both detailed feedback and the refinement repository are crucial. Without the repository, performance gains are limited, as fixed few-shot examples are suboptimal for diverse refinement needs. Detailed feedback is essential as it pinpoints errors and provides direction for response modification.

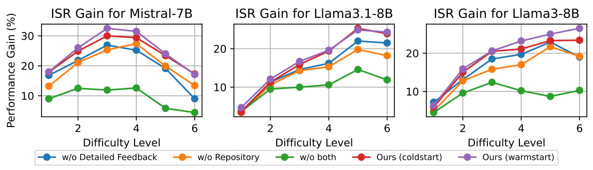


Figure 4: Ablation study on Mistral-7B, Llama3.1-8B and Llama3-8B.

4.5 Hyper-Parameter Sensitivity Analysis

We also conduct a hyper-parameter sensitivity analysis of our framework, testing different numbers of refinement few-shots and trials for successful refinement on Llama3.1-8B. As shown in Figure 5, performance improves with more trials but saturates at five, with minimal gains beyond that. Similarly, increasing the few-shot examples boosts per-

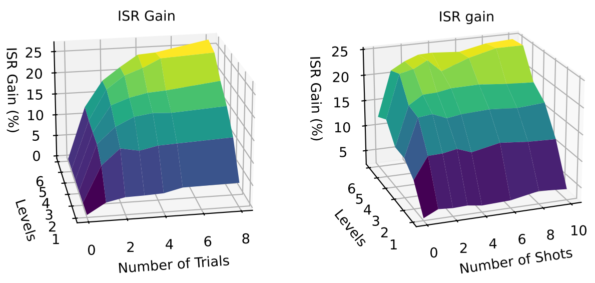


Figure 5: Parameter study on ComplexInstruct.

formance in the beginning. The performance saturates after 8 shots. This indicates that the first few numbers of trials and few-shot examples are most effective for refining the response.

Table 5: LLMs Performance on Tool Selection. (HL: Hamming Loss)

Models	HL	Acc	Precision	Recall	F1
Mistral-7B	4.13	52.85	92.98	81.39	86.80
Llama3-8B	2.64	67.60	94.39	89.48	91.87
Llama3.1-8B	2.90	61.77	94.69	87.50	90.95
Llama3.1-70B	0.86	86.40	98.41	96.38	97.38

4.6 Tool Selection Accuracy

Correctly decomposing and selecting tools are essential for feedback and refinement. We define tool selection as a multi-label prediction task for LLMs, evaluated using hamming score, accuracy, precision, recall, and F1-score. The total number of tools is 21. Results are shown in Table 18. Hamming loss, which measures the fraction of incorrect labels, is low across all models, indicating minimal mispredictions. Every model demonstrates a very high precision score, meaning that the tools they select are mostly correct, avoiding misleading feedback with incorrect tool selection. Accuracy, which measures the exact match between the selected tools and the ground truth, is the strictest metric. Despite this, all models achieve over 50% accuracy. Considering the limited performance of these models on constraint-following tasks, tool selection is a relatively easier task for LLMs. This performance gap makes it possible for our method to provide reliable feedback, collect past refinement examples and be effective in improving LLMs' constraint-following ability. We evaluate DVR's robustness to tool errors, with detailed experiments in Appendix A.6. We also evaluate the tool selection at scale in Appendix A.9.

Table 6: LLMs Self-verification Accuracy (%)

Model	Mistral-7B	Llama3-8B	Llama3.1-8B
	53.1	56.8	55.7

4.7 LLM Self-verify Ability

We evaluate the LLM ability to verify whether the responses meet the given constraints. Specifically, we present response-constraint pairs and ask LLMs to determine if the response aligns with the constraint. As shown in Table 6, the verification accuracy is around 0.5, similar to random guessing. This suggests that LLMs struggle to accurately assess responses, making them unreliable for self-feedback. In contrast, compared with LLM self-verification ability, LLMs perform significantly better in tool selection. This performance gap ensures the effectiveness of DVR. Future work can explore this further, including developing benchmarks for scenarios involving thousands of tools.

4.8 Tool Generation Accuracy

We use GPT-4o to generate Python scripts as tools, testing all 21 types of tools. Among them, 20 functioned correctly. One script designed to check the existence of a title fails for an edge case where the title is blank ("«»"). This demonstrates that tool generation is overall reliable and scalable. To balance reliability and cost, a practical strategy is to generate tools with GPT-4o and save them locally for reuse, reducing API costs. Some tool examples are shown in Appendix Fig 8.

5 Conclusion

We propose the Divide-Verify-Refine (DVR) framework to enhance LLMs' ability to follow multi-constraint instructions. DVR has three steps: (1) Divide complex instructions into single constraints and assign appropriate tools for each constraint. (2) Verify: To tackle the feedback quality problem, these tools rigorously verify the response and generate textual guidance for refinement. (3) Refine: To maximize the refinement effectiveness, we design the dynamic few-shot prompting with a refinement repository to store past refinement experiences. DVR improves LLMs' adherence to complex multi-constraint instructions. Additionally, we construct a new dataset free from hidden or conflicting constraints, providing a more comprehensive and accurate evaluation of LLM performance on multi-constraint following.

6 Limitations

There are several limitations and potential future works. (1) Currently, we consider multiple independent constraints. However, the instructions in real-world might be more complex and constraints might have dependency with each other (Wen et al., 2024). For example, the instruction can ask the response to have 4 bullet points and 2 sentences in each bullet point. In such a scenario, LLMs need to assign different priorites to these constraints. (2) Moreover, tools may not be available for new constraints. Here, we assume that we have tools for all existing constraints. However, users' requirements can be very diverse and we would not have certain tools for new constraints. (3) As shown in Appendix A.6, DVR performance would decline when tools produce errors.

7 Acknowledgment

This material is based upon work supported by, or in part by the Army Research Office (ARO) under grant number W911NF-21-10198, the Department of Homeland Security (DHS) under grant number 17STCIN00001-05-00, and Cisco Faculty Research Award.

References

Dimosthenis Antypas, Asahi Ushio, Jose Camacho-Collados, Vitor Silva, Leonardo Neves, and Francesco Barbieri. 2022. Twitter topic classification. In *Proceedings of the 29th International Conference on Computational Linguistics*.

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 1877–1901.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Xiang Chen and Xiaojun Wan. 2023. A comprehensive evaluation of constrained text generation for large language models. *arXiv preprint arXiv:2310.16343*.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. 2024a. Universal self-consistency for large language models. In *ICML 2024 Workshop on In-Context Learning*.

Yihan Chen, Benfeng Xu, Quan Wang, Yi Liu, and Zhendong Mao. 2024b. Benchmarking large language models on controllable generation under diversified instructions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17808–17816.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Shehzaad Dhuliawala, Mojtaba Komeili, Jing Xu, Roberta Raileanu, Xian Li, Asli Celikyilmaz, and Jason Weston. 2023. Chain-of-verification reduces hallucination in large language models. *arXiv preprint arXiv:2309.11495*.

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

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. 2024. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *Conference on Language Modeling*.

Jeffrey Friedl. 2006. *Mastering regular expressions*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Nan Duan, Weizhu Chen, et al. 2024. Critic: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Qianyu He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. 2024a. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. *arXiv preprint arXiv:2404.15846*.

Qianyu He, Jie Zeng, Wenhao Huang, Lina Chen, Jin Xiao, Qianxi He, Xunzhe Zhou, Jiaqing Liang, and Yanghua Xiao. 2024b. Can large language models understand real-world complex instructions? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18188–18196.

Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. 2024. A closer look at the self-verification abilities of large language models in logical reasoning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2024)*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*.

Dongwei Jiang, Jingyu Zhang, Orion Weller, Nathaniel Weir, Benjamin Van Durme, and Daniel Khashabi. 2024a. Self-[in] correct: Llms struggle with refining self-generated responses. *arXiv preprint arXiv:2404.04298*.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024b. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (ACL 2024)*.

Ryo Kamoi, Sarkar Snigdha Sarathi Das, Renze Lou, Jihyun Janice Ahn, Yilun Zhao, Xiaoxin Lu, Nan Zhang, Yusen Zhang, Ranran Haoran Zhang, Sujeeth Reddy Vummanthala, et al. 2024. Evaluating llms at detecting errors in llm responses. In *Proceedings of the 2024 Conference on Language Modeling (COLM 2024)*.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. 2019. Generalization through memorization: Nearest neighbor language models. In *International Conference on Learning Representations*.

Ming Li, Han Chen, Chenguang Wang, Dang Nguyen, Dianqi Li, and Tianyi Zhou. 2024. Ruler: Improving llm controllability by rule-based data recycling. *arXiv preprint arXiv:2406.15938*.

Minhua Lin, Hui Liu, Xianfeng Tang, Jingying Zeng, Zhenwei Dai, Chen Luo, Zheng Li, Xiang Zhang, Qi He, and Suhang Wang. 2025. How far are llms from real search? a comprehensive study on efficiency, completeness, and inherent capabilities. *arXiv preprint arXiv:2502.18387*.

Daniel Loureiro, Francesco Barbieri, Leonardo Neves, Luis Espinosa Anke, and Jose Camacho-Collados. 2022. Timelms: Diachronic language models from twitter. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 251–260.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36.

Norman Mu, Sarah Chen, Zifan Wang, Sizhe Chen, David Karamardian, Lulwa Aljeraisy, Dan Hendrycks, and David Wagner. 2023. Can llms follow simple rules? *arXiv preprint arXiv:2311.04235*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Theo X Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? In *The Twelfth International Conference on Learning Representations*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2024. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations*.

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.

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)*, pages 8345–8363.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Marco Siino. 2024. All-mpnet at semeval-2024 task 1: Application of mpnet for evaluating semantic textual relatedness. In *Proceedings of the 18th International Workshop on Semantic Evaluation (SemEval-2024)*, pages 379–384.

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

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

PeiYu Tseng, ZihDwo Yeh, Xushu Dai, and Peng Liu. 2024. Using llms to automate threat intelligence analysis workflows in security operation centers. *arXiv preprint arXiv:2407.13093*.

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Yuanzhu Peter Chen, and Tony Mak. 2024. Llms cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 13894–13908.

Fali Wang, Runxue Bao, Suhang Wang, Wenchao Yu, Yanchi Liu, Wei Cheng, and Haifeng Chen. 2024a. Infuserki: Enhancing large language models with knowledge graphs via infuser-guided knowledge integration. In *Findings of the Association for Computational Linguistics: EMNLP*, pages 3675–3688. Association for Computational Linguistics.

Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, et al. 2024b. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *arXiv preprint arXiv:2411.03350*.

Fei Wang, Chao Shang, Sarthak Jain, Shuai Wang, Qiang Ning, Bonan Min, Vittorio Castelli, Yassine Benajiba, and Dan Roth. 2024c. From instructions to constraints: Language model alignment with automatic constraint verification. *arXiv preprint arXiv:2403.06326*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxin Xu, et al. 2024. Benchmarking complex instruction-following with multiple constraints composition. *arXiv preprint arXiv:2407.03978*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Zhiwei Zhang, Fali Wang, Xiaomin Li, Zongyu Wu, Xianfeng Tang, Hui Liu, Qi He, Wenpeng Yin, and Suhang Wang. 2025. Catastrophic failure of llm unlearning via quantization. In *The International Conference on Learning Representations*.

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. In *The Twelfth International Conference on Learning Representations*.

A Appendix

A.1 Algorithm

DVR algorithm is shown in Algorithm 1. The process begins with the LLM selecting the appropriate tools for each instruction. The selected tools assess whether the generated response meets the constraints and provide textual feedback if any violation is detected. To improve adherence, few-shot examples with the same constraint type are retrieved from a refinement repository. If the response is successfully refined and passes the tool-based validation, this refined version is stored in the repository for future use. The response is output when it passes all tools or the budget is met.

A.2 Baseline Details

We conduct experiments on 6 baselines, encompassing reflexion-based approaches, prompting strategies, and tool-assisted techniques. The details are as follows:

- Reflexion (Shinn et al., 2024): This method allows LLMs to self-reflect on their own responses and provide valuable feedback for future outputs. With the feedback, LLMs will refine their responses.

- Branch-solve-Merge (BSM) (Saha et al., 2024): BSM uses a "Divide and Conquer" approach to break complex instructions as individual branches. Then the LLMs will merge

Algorithm 1 Algorithm for DVR

```
Input: LLM $\mathcal{M}$, Instructions $X$, Toolset $T$
Output: Response set $Y$
Select: Trial number $n$
1:  Initialize the refinement repository $Q = \{\}$
2:  for $I \in X$ do
3:      Generate response: $R_0 = \mathcal{M}(p_{generate}, I)$.
4:      Initialize the toolset for $I$: $T_I = \{\}$.
5:      Decompose:$\mathcal{M}(p_{decomp}, I) \to \{c_i\}_{i=1,2...}$
6:      for $c \in \{c_i\}_{i=1,2,3...}$ do
7:          $\mathcal{M}(p_{select}, c) \to t$, where $t \in T$.
8:          $\mathcal{M}$ sets parameters for $t$.
9:          $T_I = T_I \cup t$.
10:     end for
11:     $R = R_0$, $a = n$.
12:     while $a > 0$ do
13:         $a = a - 1$
14:         Verify and get feedback from tools: $F_I = \{f_i\}_{i=1,2,3...}$, where $f_i = t(R)$.
15:         if $f = True, \forall f \in F_I$ then
16:             return $R$
17:         end if
18:         Retrieve: $s^t = \{(I_i, R_i, f_i, R_i')^t\}_{i=1,2...}$, where $s^t \subseteq Q$.
19:         Refine: $R' = \mathcal{M}(p_{refine}, s^t, I, R, f)$, where $f \in F_i$ and $f \neq True$.
20:         if $t(R') = True$ then
21:             Save: $Q = Q \cup \{(I, R, f, R')^t\}$
22:             Update the response: $R = R'$
23:             $a = n$
24:         end if
25:     end while
26:     $Y = Y \cup R$
27: end for
```

the responses from branches as the final answer. Similarly, in our experiment, we use LLMs to generate a response for each single constraint and then merge them together.

- Universal Self-Consistency (U-SC) (Chen et al., 2024a): This study extends the idea of Self-Consistency (Wang et al., 2023) to free-form generation. It first generates several candidate responses and then asks LLMs to select the most consistent one.

- Rejection Sampling (Saunders et al., 2022): Since we have tools for reliable verification, the most simple method is to select the best one from a set of responses. Here, we give the maximum number of trials as 5.

- ReAct (Yao et al., 2023): In ReAct, LLMs take actions based on the observation of the environment. Here, we adopt this method by letting the tools as the environment and giving LLMs boolean signals indicating whether the generated response adheres to all constraints in the instruction.

- CRITIC (Gou et al., 2024): CRITIC uses external API to evaluate the toxicity score of a generated response, focusing on a single predefined task. We adopt this method as a variant of our DVR framework, where tools will pinpoint which constraint of the instruction is not satisfied.

A.3 ComplexInstruct

We have 21 types of constraints which can be divided into 8 general categories (Zhou et al., 2023) as shown below:

- Keywords:

  (1) Include keyword,
  (2) Include keyword at least/less than certain frequency,
  (3) Forbidden word,
  (4) At least/less than certain frequency of letters.

- Length:

  (1) At least/less than certain number of words,
  (2) At least/less than certain number of sentences,
  (3) Exact number of paragraphs.

- Detectable Content:
  (1) postscript,
  (2) Exact number of placeholders.

- Detectable Format:
  (1) Number of bullet points,
  (2) Add title,
  (3) Answer from options,
  (4) Minimum of highlighted sections,
  (5) Json format.

- Change Cases:
  (1) All uppercase,
  (2) All lowercase,
  (3) At least/less than certain number of all-capital words.

- Startend:
  (1) End the text with a certain sentence,
  (2) Wrap whole response in double quotation.

- Punctuation:
  (1) No commas in response.

- Language:
  (1) Respond with certain language.

A.4 Detailed experiments

Additional experiments on Llama3-8B and Llama3.1-70B are shown in Table 7 and Table 8 respectively. We can observe that our methods consistently outperform baselines on different LLMs. In Table 9, we also observe that LLMs perform overall good on the CoDI dataset (Chen et al., 2024b). There are two reasons. The first reason is that instructions are relatively simple, and only contain two constraints. Additionally, another study also shows that LLMs perform relatively better on sentiment and topic constraints (Chen et al., 2024b) compared with format constraints. The LLMs inherently have better performance on semantic constraints over structural constraints. Our methods also outperform baselines and successfully improve the instruction satisfaction rate on CoDI.

Table 7: Performance of methods across levels 1 to 6 (Llama-3-8B-Instruct)

Method	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Vanilla	89.1	71.7	56.4	44.2	30.4	20.4
Reflexion	88.8	72.1	57.5	41.5	30.0	20.9
BSM	89.2	71.9	56.0	41.3	28.8	19.5
U-SC	89.5	71.8	56.7	45.1	31.2	20.6
R-Sample	90.8	80.9	64.5	52.9	39.8	31.0
ReAct	93.6	81.3	68.8	54.4	39.1	30.7
CRITIC	94.1	85.8	74.4	61.2	51.1	41.5
DVR$_{CS}$	95.0	86.7	76.9	65.3	53.7	43.8
DVR$_{WS}$	95.4	87.6	77.0	67.4	55.4	46.9

Table 8: Performance of methods across levels 1 to 6 (Llama-3.1-70B-Instruct-AWQ-INT4)

Method	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Vanilla	95.5	83.7	72.4	63.2	51.3	35.9
Reflexion	95.3	83.5	72.8	63.0	51.6	36.1
BSM	95.0	84.5	72.6	64.8	49.6	34.2
U-SC	96.0	83.3	71.8	64.0	52.5	36.3
R-Sample	97.3	90.8	83.2	72.2	63.8	50.7
ReAct	97.5	91.0	83.5	72.4	65.0	52.1
CRITIC	98.1	93.2	87.6	79.1	73.7	61.3
DVR$_{CS}$	98.0	94.3	88.2	82.0	75.7	63.1
DVR$_{WS}$	98.2	94.6	88.7	82.2	76.0	64.2

Experiments on IFEval: We conduct experiments on IFEval (Zhou et al., 2023) which is an instruction-following benchmark widely used for industry. The IFEval dataset evaluates the instruction-following ability and is one of the core benchmarks used in the Open LLM Leaderboard (Hugging Face). We conduct experiments on Mistral-7B-v0.3 and the results are shown in Table 11. DVR outperforms all other baselines on IFEval benchmark.

Table 9: Instruction Satisfaction Rate (ISR) on CoDI

Method	Mistral 7B	Llama3 8B	Llama3.1 8B
Vanilla	68.8	68.8	68.6
Reflexion	69.4	70.0	69.8
BSM	68.2	68.4	68.6
USC	69.2	70.2	69.6
Reject Sample	79.8	80.8	81.4
ReAct	80.4	81.0	81.8
CRITIC	88.6	93.0	91.2
DVR (coldstart)	92.0	94.2	**94.6**
DVR (warmstart)	**93.2**	**94.4**	**94.6**

Table 10: Comparison Across Different Constraints Types (coldstart)

	Mistral-7B		Llama3-8B		Llama3.1-8B		Llama 3.1-70B	
Constraint Type	Vanilla	DVR	Vanilla	DVR	Vanilla	DVR	Vanilla	DVR
Detectable Content	76.36	88.49	84.18	95.82	86.29	96.19	97.06	98.14
Keywords	76.04	84.32	83.84	87.53	84.94	88.77	87.88	92.05
Punctuation	24.34	71.31	91.03	95.47	97.01	98.29	98.38	98.38
Case Change	70.08	80.15	81.28	93.20	82.97	89.96	80.23	96.24
Start End	81.29	88.71	84.88	88.71	84.07	91.78	89.37	95.94
Detectable Format	69.59	81.30	81.57	89.29	84.69	92.38	90.52	95.56
Language	69.11	82.72	77.06	86.24	81.96	89.30	90.83	94.34
Length Constraints	50.42	72.61	65.29	79.66	68.55	83.93	80.50	90.17

Table 11: ISR (%) for IFEval Dataset

Method	Vanilla	Reflexion	BSM	U-SC	R-Sample	ReAct	CRITIC	DVR
ISR	47.32	47.13	47.87	46.95	53.23	53.97	55.53	60.44

Experiments on GPT4: We conduct experiments on GPT-4-turbo. Shown in Table 12, we can observe that GPT-4-turbo performs better than open-source models (Mistral and Llama). Surprisingly, applied on Llama3.1-8B, DVR can still outperform GPT-4-turbo, indicating that DVR exploits the potential of the open-source model.

Table 12: Performance Comparison to GPT4-turbo (zero shot).

Model	Level 1	Level 2	Level 3	Level 4	Level 5	Level 6
Mistral-7B	77.0	55.3	34.1	19.9	12.4	6.3
DVR (Mistral-7B)	95.0	81.3	66.6	51.4	36.4	23.4
Llama3.1-8B	90.5	76.6	62.5	50.1	35.6	25.3
DVR (Llama3.1-8B)	95.2	88.7	79.2	69.7	60.5	49.6
GPT4-turbo(zero shot)	95.3	88.4	78.8	65.2	53.7	42.6

A.5 Fluency and Readability

In this subsection, we investigate if our framework would sacrifice comprehensibility and fluency in order to follow complex-constraints. We evaluate key metrics such as readability, perplexity, and coherence. These metrics assess the comprehensibility and fluency of the responses. Results of ComplexInstruct and CoDI are shown in Table 13 and Table 14. They both show that our framework has performance comparable to those of Vanilla, indicating that it does not degrade fluency and readability. The reason is that our method does not change any weights in LLMs, which maintains their ability in generating fluent and comprehensible text.

Table 13: Descriptive Statistics of Responses (ComplexInstruct), where Coherence_or1 is first order coherence and Coherence_2 is the second order coherence.

Model	Method	Readability ↑	Perplexity ↓	Co_or1 ↑	Co_or2 ↑
mistral7B	Vanilla	62.24	18.22	0.61	0.59
	DVR	61.93	18.95	0.59	0.57
llama3-8B	Vanilla	63.77	18.49	0.57	0.57
	DVR	63.58	18.08	0.59	0.56
llama3.1-8B	Vanilla	63.43	19.68	0.62	0.61
	DVR	62.75	18.30	0.62	0.60
llama3.1-70B	Vanilla	61.96	17.99	0.64	0.63
	DVR	63.51	18.04	0.63	0.62

Table 14: Descriptive Statistics of Responses (CoDI)

Model	Method	Readability ↑	Perplexity ↓	Co_or1 ↑	Co_or2 ↑
mistral7B	Vanilla	63.62	15.27	0.82	0.81
	DVR	63.53	15.98	0.83	0.82
llama3-8B	Vanilla	63.79	14.25	0.79	0.76
	DVR	64.14	16.07	0.80	0.77
llama3.1-8B	Vanilla	62.18	16.27	0.81	0.83
	DVR	62.02	17.58	0.81	0.80

A.6 Robustness of DVR

We also conduct experiments to assess DVR's performance in the presence of tool errors. Two types of errors are introduced: random noise and systematic bias. Specifically, we evaluate the framework on instructions with length constraints, using 600 samples (from ComplexInstruct) for word count control and another 600 for sentence count control. A constraint example: "The response needs to be less than (or at least) x number of words/sentences." where x ranges from 10 to 100 for words and 3 to 5 for sentences. We add two types of noises to tools:

Noise: Gaussian noise with a mean of 0 is added to the counted number of words (or sentences) to simulate random errors. The DVR's performance is then measured across different deviation levels.

Bias Errors: A fixed bias is added to the counted values of words (or sentences) to introduce systematic errors. The tables below demonstrate DVR's performance under different bias values.

Observations: We have several observations in Table 15 and Table 16. (1) The performance will decrease as the noise levels (deviation, bias values) increase. (2) As the errors become large, the performance degradation will saturate. (3) Overall, DVR will not perform much worse than vanilla even if the bias and errors are large (20 for word count and 4 for sentence count). (4) The impact of noise on the overall instruction satisfaction rate is less severe compared to its influence on specific constraints.

Table 15: Satisfaction Rate for Word Number Constraints (%)

Deviation	0	5	10	20	Vanilla
Words Number Satisfaction Rate	88.00	87.17	82.83	81.50	68.16
Instruction Satisfaction Rate	48.17	45.00	43.67	43.67	10.17

Bias	0	5	10	20	Vanilla
Words Number Satisfaction Rate	88.00	87.17	84.33	82.67	68.16
Instruction Satisfaction Rate	48.17	47.83	47.00	45.67	10.17

Table 16: Satisfaction Rate for Sentence Number Constraints (%)

Deviation	0	1	2	4	Vanilla
Sentences Number Satisfaction Rate	74.50	68.17	62.50	60.50	56.33
Instruction Satisfaction Rate	42.83	38.17	35.33	34.33	10.17

Bias	0	1	2	4	Vanilla
Sentences Number Satisfaction Rate	74.50	64.67	58.67	56.50	56.33
Instruction Satisfaction Rate	42.83	40.50	32.67	31.33	10.17

A.7 Computation Time

We conducted experiments with 20 instructions, each containing 6 constraints, using Mistral-7B. The number of trials was set to 5, consistent with the paper's settings. The average running time is summarized below:

Table 17: The Average Running Time for One Sample

Method	Vanilla	Reflexion	U-SC	BSM	RS	ReAct	CRITIC	DVR
Time (s)	5.91	20.53	41.34	46.21	32.32	36.48	37.98	33.91

As shown in Table 17, our method does not exhibit significantly higher running time compared to other baselines. Considering the performance gains (Table 2), our method demonstrates a balance between efficiency and effectiveness.

A.8 Related Work Details

Instruction Following. (1) Evaluation: Recent studies evaluate instruction-following capability of LLMs from various perspectives (Dubois et al., 2024; Zhou et al., 2023; Jiang et al., 2024b; Chen et al., 2024b; Zhou et al., 2023; He et al., 2024b). They evaluate LLMs' instruction-following ability by testing on length (Dubois et al., 2024), format (Zhou et al., 2023), semantic and topic constraints (Chen et al., 2024b). Most works only test LLMs on simple instructions with only 1-2 constraints. Recently, some works test on instructions with multiple constraints (He et al., 2024a; Jiang et al., 2024b). They find that LLMs struggle to follow complex instructions as the number of constraints increases. Moreover, there is a big performance gap between the open-source models and the closed-source models on instruction-following. **(2) Methods:** Upon finding these problems, some works (Chen and Wan, 2023; Sun et al., 2024; Wang et al., 2024c; He et al., 2024a) use various prompting strategies to generate instructions and responses with advanced LLMs (e.g., GPT4) and then use the generated data to fine-tune open-source LLMs. While most methods consider only a few constraints, He et al. (2024a) focus on improving LLMs' adherence to multiple constraints. They generate complex instruction datasets by merging instructions with external constraints and iteratively refine student model responses using GPT-4 as a teacher. The student model is tuned on both intermediate modifications and final refined responses.

LLMs Using Tools. Tools have been extensively employed to enhance the capabilities of LLMs across various domains. For instance, retrievers are used to augment the response generation of LLMs by fetching relevant information (Khandelwal et al., 2019; Gou et al., 2024), while search engines enhance the model's access to real-time data (Nakano et al., 2021). Similarly, calculators are adopted to support math reasoning of LLMs (Cobbe et al., 2021), interpreters are used to facilitate accurate code generation (Chen et al., 2022; Gao et al., 2023) and mathematical provers help in verifying theoretical proofs (Jiang et al., 2023). CRITIC (Gou et al., 2024) uses external API to evaluate the toxicity score of a generated response, focusing on a single predefined task. In contrast, DVR involves preparing and selecting multiple tools, allowing for greater flexibility. Moreover, while CRITIC provides feedback as a single numerical score, DVR offers textual guidance, which has more detailed analysis for response. Another key distinction is that DVR incorporates dynamic few-shot prompting, further enhancing refinement effectiveness.

A.9 Tool Selection at Scale

Currently, there is no large-scale benchmark for instruction-following with such an large toolset. To evaluate tool selection at scale, we expand the tool pool to 2,000 entries by combining our original tools descriptions with fictitious tool descriptions generated by GPT-4-Turbo. For example, a tool description is like: "Citation Format Validator: It ensures that all citations in the text adhere to a specified citation style like APA, MLA, or Chicago."

We then design a retrieval method. All tool descriptions are embedded using the all-mpnet-base-v2 model (Siino, 2024). The LLM (Llama-3.1-8B-Instruct) generates a tool query, which is also embedded. We use cosine similarity to match the query against the tool pool. The top 10 most similar tools are retrieved. The LLM selects the final tool from this top-10 set. We test on 1,000 constraints and report both the recall of retrieval and final accuracy of the tool selection.

The results are shown in Table 18. As the tool pool size increases, recall and final accuracy gradually decrease. Overall recall remains high, indicating that the correct tool is still very likely to be selected, even with a large candidate pool.

Tool Numbers	**100**	**200**	**500**	**1000**	**2000**
Recall@10	0.982	0.980	0.977	0.963	0.961
Final ACC	0.931	0.931	0.925	0.920	0.917

Table 18: Tool selection performance.

Refinement Example

Prompt: Excuse me, could you please provide text that focuses on the music-topic? use the keywords artists. highlight at least 1 key point by wrapping it with *. For example: *highlighted key point*. The number of sentences in your response should be less than 5. be sure the letter e appears at least 4 times in your response. Make sure to include at least 3 placeholder represented by square brackets, such as [address], [name]. the total number of words in your response should be at least 23.

Response:

«Music Unites» From festivals to online communities, music continues to bridge cultural and geographical gaps like never before. *Electronic music has especially seen a surge in popularity among the younger generation*. Talented artists from all over the world have been coming together to create beautiful melodies and harmonies. [name] is one such artist who has been making waves in the industry with his eclectic beats. His music can be found on various platforms, including [address] and [website]. Many emerging artists, including [another name], are also experimenting with different genres.

Unsatisfied Constraint: The number of sentences in your response should be less than 5.

Feedback: The response contains 6 sentences, which is 2 more than allowed. Please remove at least 2 sentences.

Refined Response: «Music Unites» From festivals to online communities, music continues to bridge cultural and geographical gaps like never before. *Electronic music has especially seen a surge in popularity among the younger generation*. Talented artists from all over the world have been coming together to create beautiful melodies and harmonies. [name] is one such artist who has been making waves in the industry with his eclectic beats and his music can be found on various platforms, including [address] and [website].

Figure 6: The refinement process example

Decomposition Prompt

You are an advanced assistant specializing in identifying and listing output constraints from provided instructions. The instructions typically include a task related to generating content on a specific topic and one (or multiple) format constraint(s). Your goal is to focus only on extracting and listing all the format constraints required for the output, ignoring the content-related task.
Instruction:
Generate a few lines of text that touch on the topic of tv. Put your entire answer in JSON format...
Format Constraints:
#1. Put your entire answer in JSON format.
#2. The word 'show' should not appear in your response.
... (more examples)
Instruction:
{current instruction}
Format Constraints:

Tool Selection Prompt

You will be given a list of constraints. Each constraint belongs to a specific category. Your task is to recognize and categorize each constraint. Only output the category from the following options: postscript, placeholder, include keyword, exclude keyword, letter frequency, keyword frequency, sentence count constraint, word count constraint, *** separator, bullet points, fixed responses, highlighted, JSON format, title format, quoted response, end phrase, no commas, all capital letters, all lowercase, capital word frequency, language restriction
Please ensure to categorize each constraint accurately according to its description. There is definitely a valid category option for each constraint. Here are examples:
Prompt: Make sure to include the word 'mutations'.
Category: include keyword
...(more examples)
Prompt: {Current Prompt}
Category:

Refinement Prompt

You are an AI assistant responsible for refining a given response. Given a prompt, its response, and the analysis of the response, your task is to modify the response according to the analysis.
#Prompt: I'm looking for text that explores arts or culture, can you assist? There should be no commas in your reply......
#Original Response: Art has the power to bring people together and transcend cultural boundaries. It can evoke emotions and spark conversations that might not be possible through other means. *At the [address] museum, ...
#It does not satisfy the constraint: There should be no commas in your reply.
#Analysis: The response contains 4 comma(s). Here are the detected commas: (museum, visitors) (installations, each)... Please remove all commas.
#Modified Response: Art has the power to bring people together and transcend ...(more examples)
#Prompt: current prompt
#Original Response: current response
#It does not satisfy the constraint: current unsatisfied constraint
#Analysis: current feedback
#Modified Response:

Figure 7: The prompts used in DVR

Word Counting

This example is obtained through GPT4-o with zero-shot. It demonstrates that reliable tools can be easily created. The details are as follows:

```
def feedback(response, max_words=None, min_words=None):
    # Count the number of words in the response
    word_count = len(response.split())

    # Check for maximum word constraint
    if max_words is not None and word_count > max_words:
    return f"Response failed because it has {word_count} words,
    exceeding the maximum allowed limit of {max_words} words."

    # Check for minimum word constraint
    if min_words is not None and word_count < min_words:
    return f"Response failed because it has only {word_count}
    words, fewer than the minimum required {min_words} words."

    # If all constraints are satisfied
    return True
```

Lowercase Validation

This example validates whether a given text is entirely in lowercase. If any word contains uppercase letters, it provides feedback on which words need correction. The implementation is as follows:

```
class LowercaseLetter:
    def __init__(self):
        pass

    def feed_back(self, value):
        # Split the input string into words
        words = value.split()

        # Find words that are not fully in lowercase
        upper_case_words = [word for word in words if
        any(char.isupper() for char in word)]

        if value.islower():
            return True
        else:
            return f"The response contains words that are not in
            all lowercase letters: {', '.join(upper_case_words)}.
            Please lowercase all of them."
```

Figure 8: The tool examples