# ConInstruct: Evaluating Large Language Models on Conflict Detection and Resolution in Instructions

**Anonymous ACL submission**

## Abstract

Instruction-following is a critical capability of Large Language Models (LLMs). While existing works primarily focus on assessing how well LLMs adhere to user instructions, they often overlook scenarios where instructions contain conflicting constraints—a common occurrence in complex prompts. The behavior of LLMs under such conditions remains under-explored. To bridge this gap, we introduce ConInstruct, a benchmark specifically designed to assess LLMs' ability to detect and resolve conflicts within user instructions. Using this dataset, we evaluate LLMs' conflict detection performance and analyze their conflict resolution behavior. Our experiments reveal two key findings: (1) Proprietary LLMs exhibit strong conflict detection capabilities, with Claude-3.5-Sonnet and GPT-4o achieving average F1-scores of 86.6% and 84.9%, ranking first and third, respectively. (2) Despite their strong conflict detection abilities, LLMs rarely explicitly notify users about the conflicts or request clarification when faced with conflicting constraints. These results underscore a critical shortcoming in current LLMs and highlight an important area for future improvement when designing instruction-following LLMs.

## 1 Introduction

Large Language Models (LLMs) (OpenAI et al., 2023; Touvron et al., 2023; Chowdhery et al., 2023) have witnessed significant advancements in recent years, demonstrating remarkable capabilities in reasoning (Wei et al., 2022; Wang et al., 2022), and time-series forecasting (Jia et al., 2024; Zhang et al., 2024a). A fundamental ability of LLMs is to follow instructions—generating responses that align with user-provided instructions. Instruction-following (Ouyang et al., 2022) has emerged as a key research focus, playing a critical role in enhancing the interpretability, controllability, and trustworthiness of LLMs in real-world applications.

**Instruction with Conflicts**

Send an email to a software developer asking for a product demo, ensuring the email begins with the phrase "I hope this message finds you well" and concludes with "Looking forward to your response." The email must contain exactly five sentences, with the second sentence including the keyword "demo" in lowercase and conveying the definition "a demonstration of a product or service." Tailor the email for a professional audience, avoiding technical jargon while maintaining a formal tone. Additionally, provide a brief background about your interest in the product and specify a preferred time frame for the demo. Finally, the email must include a Chinese idiom. The email must contain at least 10 sentences. Start the email with an apology or an expression of regret.

**Response from GPT-4o**

Subject: Request for a Product Demo \n\nDear [Developer's Name], \n\nI hope this message finds you well. … Looking forward to your response. \n\nBest regards, \n[Your Full Name] \n[Your Job Title] \n[Your Company Name] \n[Your Contact Information]

**Response from Claude-3.5-Sonnet**

I notice there are some contradictions in your requirements. You've asked for exactly 5 sentences, but also specified that the email must contain at least 10 sentences. Additionally, you've requested to start with "I hope this message finds you well" but also to begin with an apology or expression of regret.\n\nLet me clarify - would you like me to:\n1. Write a 5-sentence email starting with "I hope this message finds you well", or\n2. Write a 10+ sentence email starting with an apology?\n\nOnce you clarify this, I'll be happy to draft an email that meets your other requirements, including the lowercase "demo", Chinese idiom, and professional tone.

Figure 1: An instruction with conflicts from ConInstruct, where text in green and red indicate conflicts between phrase constraints and length constraints, respectively. The lower part of the figure presents two responses from GPT-4o and Claude-3.5-Sonnet for the instruction.

Existing instruction-following works primarily focus on evaluating to what extent LLMs' outputs align with user instructions using rule-based and model-based evaluation methods. For rule-based evaluation, Zhou et al. (2023a) proposed IFEval, a benchmark comprising verifiable instructions (e.g., "*Include the keyword 'useful' in your response*"), where a rule-based program can verify whether a model's output meets the given instructions. Meanwhile, recent studies suggest that LLMs can rival human annotators (He et al., 2024b) and serve as reliable evaluators (Zheng et al., 2023). Building on these findings, model-based evaluation (Chen et al., 2024; Qin et al., 2024) leverages strong LLMs to automatically assess whether LLMs' outputs adhere to user instructions. The latest research integrates rule-based and model-based evaluation approaches (Jiang et al., 2024; Zhang et al., 2024b; Wen et al., 2024). On the other hand, concurrent works (Wallace et al., 2024; Zhang et al., 2025; Geng et al., 2025) evaluate whether LLMs can follow an instruction hierarchy, where high-level instructions (e.g., system instructions) take precedence over low-level ones (e.g., user instructions).

Prior works assume that all constraints in the user instructions are coherent and non-conflicting. In practice, when users provide long or complex instructions, they may unintentionally introduce conflicting constraints—requirements that cannot be simultaneously satisfied by LLMs. Figure 1 illustrates an instruction containing two conflicts: one between phrase constraints and another involving length constraints. The presence of such conflicts poses a unique challenge for LLMs. If an LLM generates a response without notifying the user of these conflicts (as seen in GPT-4o's response in Figure 1), the user may not realize that their instruction contains conflicts and the model's output fails to fully satisfy the instruction. In such cases, a preferable conflict resolution behavior is to explicitly inform the user about the conflicts and request clarification before proceeding (as shown in Claude-3.5-Sonnet's response in Figure 1). Despite the growing interest in instruction-following, no prior work has systematically evaluated LLMs' performance when faced with user instructions with conflicts.

To bridge this gap, we introduce **ConInstruct**[1], a novel dataset designed to evaluate LLMs on **Con**flicting **Instruct**ions that contain diverse constraints. Specifically, our dataset covers six distinct tasks, with each instruction incorporating six types of constraints: content, keyword, phrase, length, format, and style constraints. Furthermore, we design 7-9 different types of conflicts per instruction, including both intra-constraint conflicts (e.g., conflicts between phrase constraints) and inter-constraint conflicts (e.g., conflicts between keyword and phrase constraints) (see conflicts in Figure 2). Using this dataset, we systematically analyze LLMs' performance in **conflict detection** and examine their behaviors in **conflict resolution**.

Conflict detection assesses how well LLMs can identify conflicts within a given instruction. To evaluate this, we introduce a new constraint into a conflict-free instruction, ensuring it conflicts with an already present constraint. We then ask LLMs to determine whether the instruction contains conflicting constraints. Our results show that proprietary LLMs exhibit strong conflict detection capabilities, with Claude-3.5-Sonnet and GPT-4o achieving average F1-scores of 86.6% and 84.9%, respectively, the best and third-best performing models. Notably, as the number of conflicts in an instruction increases, LLMs exhibit improved conflict detection ability, aligning with our intuitions.

[1]We will release our code and dataset in the future.

Conflict resolution, on the other hand, investigates how LLMs behave when faced with instructions containing conflicts. While LLMs perform well in conflict detection, our findings indicate that they often generate responses without explicitly informing the user about conflicts. For example, when an instruction contains 1–2 conflicts, GPT-4o will directly generate a response in 97.5% of cases, satisfying only a subset of the constraints but failing to notify the user of the conflicts. Even the best-performing model, Claude-3.5-Sonnet, explicitly alerts users to conflicts in only 32% of cases—either by (1) requesting further clarification (16.5%) or (2) resolving the conflicts autonomously and responding to the resolved instruction (15.5%). Moreover, as the number of conflicts in an instruction increases, strong LLMs (Claude-3.5-Sonnet, Claude-3.5-Haiku, and GPT-4o) become more likely to acknowledge the existence of conflicts in their responses.

Our contributions can be summarized as follows: (1) We introduce ConInstruct, a novel dataset designed to evaluate LLM performance in handling user instructions with conflicts. (2) We conduct an in-depth study on conflict detection, demonstrating that proprietary LLMs exhibit strong detection capabilities. (3) We analyze the conflict resolution behaviors exhibited by LLMs when encountering conflicting instructions. Our findings reveal that while proprietary LLMs exhibit strong conflict detection capabilities, they often fail to convey conflicts explicitly in their responses, highlighting an important area for future improvement in instruction-following LLMs.

## 2 ConInstruct Benchmark

### 2.1 Dataset Construction

As shown in Figure 2, the construction of ConInstruct consists of three steps: preparing seed instructions, expanding them with constraints, and introducing conflicts into the expanded instructions. Below, we provide further details on each step.

**Preparing Seed Instructions.** We begin by manually curating 100 seed instructions, which serve as fundamental instructions without additional constraints. In designing these seed instructions, we prioritize task and domain diversity to ensure broad coverage across various scenarios. As shown in Figure 3, ConInstruct comprises six common NLP

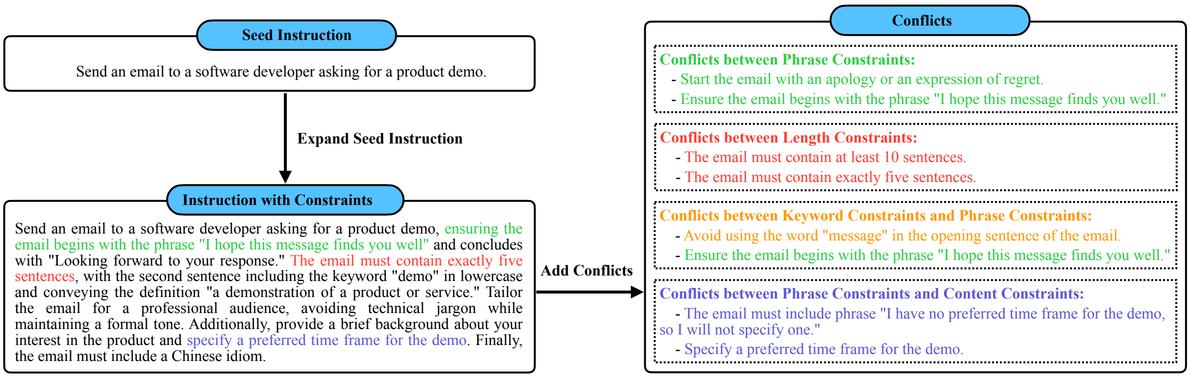



Figure 2: The construction process of the ConInstruct Benchmark: We first prepare a seed instruction, then add constraints to it. Finally, we introduce conflicts into the expanded instructions. Due to space limitations, we showcase only four conflicts. In each conflict pair, the first constraint is newly introduced, while the second comes from the original instruction. These two constraints are mutually conflicting.


Figure 3: Task and domain distribution of ConInstruct. The size of each task/domain sector reflects its proportion in the dataset.

tasks: *email writing, plan generation, story generation, open-domain question answering (QA), review writing, and article writing*. These tasks span 35 scenario-specific domains, including *travel, work, health, finance, technology, and history*. Overall, the seed instructions provide a diverse set of tasks and scenarios.

**Constraint Types.** We now introduce the constraints used to expand seed instructions. Following previous works on instruction-following (Jiang et al., 2024; He et al., 2024a), we design six widely-used constraint types: **Content Constraints** require the output to include specific details related to the content, such as reasons, purposes, topics, or background information. **Keyword Constraints** enforce the inclusion of specific keywords in the output or specify constraints on their part of speech or meaning (He and Yiu, 2022). **Phrase Constraints** mandate the presence of specific phrases or sentences in the output. **Length Constraints** impose restrictions on the length of the output, such as word count, sentence count, or paragraph count. **Format Constraints** specify the format of the output (e.g., JSON, Markdown) or its language format (e.g., requiring the output to be entirely in English). **Style Constraints** control aspects such as sentiment, readability, and overall tone of the output. Further details on these constraint types are provided in Section A.

**Expanding Seed Instructions.** We leverage GPT-4o to inject constraints into seed instructions. To enhance constraint diversity, we require GPT-4o to incorporate all six types of constraint into each seed instruction. See Figure 2 for an example of an expanded instruction. The prompt used for this expansion is detailed in Table 5.

**Conflict Types.** When designing conflicting constraints, we prioritized the feasibility of evaluating constraint satisfaction using LLMs or automated programs. To this end, we define nine types of conflicts based on six widely used constraints (Jiang et al., 2024; He et al., 2024a), categorized into six **intra-constraint conflicts** and three **inter-constraint conflicts**. Intra-constraint conflicts occur within the same constraint type, including conflicts within Content Constraints (**CC**), Keyword Constraints (**KK**), Phrase Constraints (**PP**), Length Constraints (**LL**), Format Constraints (**FF**), and Style Constraints (**SS**). Inter-constraint conflicts occur between different constraint types, including conflicts between Keyword and Phrase Constraints (**KP**), Phrase and Content Constraints (**PC**), and Phrase and Style Constraints (**PS**). Further details on these conflict types are provided in Section B.

**Adding Conflicts.** We use GPT-4o to introduce conflicting constraints into the expanded instruc-

| Basic Statistics | | | | | Conflict Distribution | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Inst. | Word | Sent. | CT | CFT | CC | KK | PP | LL | FF | SS | KP | PC | KP |
| 100 | 138.9 | 6.4 | 6 | 8.6 | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 94 | 70 |

Table 1: ConInstruct Statistics. 'Inst.', 'Word', 'Sent.', 'CT', and 'CFT' denote the number of expanded instructions, average words, sentences, constraint types, and conflict types per instruction. The right half of the table shows the number of conflicts for each conflict type.

tions. To better control the number of conflicts in each instruction, we prompt the model to generate conflict pairs rather than directly injecting conflicting constraints into the instructions. Each conflict pair consists of two constraints: one extracted from the expanded instruction and another, newly constructed by GPT-4o, that directly contradicts the former. We instruct GPT-4o to generate one conflict pair for each of the nine predefined conflict types. Figure 2 illustrates four conflict pairs corresponding to an expanded instruction. The prompt used to add conflicts is provided in Table 6.

### 2.2 Quality Control

To ensure the data quality of ConInstruct, we use a two-step verification process for each instruction. In the first step, two annotators refine the expanded instructions and conflicts generated by GPT-4o. For expanded instructions, they assess the reasonableness and correctness of constraints, correcting any unreasonable or erroneous ones. They also check whether the expanded instructions include all six types of constraints and add any missing ones. For conflicts, annotators examine whether newly introduced constraints are indisputably in conflict with the constraints in expanded instructions. Any ambiguous conflicts are revised accordingly. For example, if the constraint in an expanded instruction states that "The email should contain 150–200 words", and a new constraint states that "The email must be brief," the conflict is ambiguous because "brief" lacks a clearly defined word limit. Annotators also ensure that all types of conflicts are covered and construct any missing ones. In the second step, a third annotator[2] reviews the revised instructions and conflicts, removing any constraints or conflicts they deem unreasonable.

### 2.3 Dataset Statistics

Table 1 presents the basic statistics of the expanded instructions in ConInstruct. Each instruction contains six types of constraints and an average of 8.6 conflict types. In the conflict detection and resolution experiments, we construct conflicting instructions by combining conflicts with expanded instructions. Specifically, we append the new constraints from the conflicts directly to the end of the expanded instructions. This approach allows us to generate a sufficient number of instructions with varying numbers of conflicts. For example, when the number of conflicts is set to one, we can construct a total of 864 conflicting instructions.

[2]All annotators are college students and independent of our research.

## 3 Experiment Setup

We will introduce the common experiment setup for conflict detection and conflict resolution.

### 3.1 Preparing Instructions with Conflicts

For each task, we first evaluate LLMs on instructions with a single conflict and then analyze their behaviors on instructions with multiple conflicts.

**Instructions with a Single Conflict.** As introduced in Section 2.3, each expanded instruction contains $n$ different types of conflicts ($7 \leq n \leq 9$). For each instruction $I_i \in \mathcal{I}_0$ ($\mathcal{I}_0$ denotes the set of conflict-free expanded instructions from ConInstruct) and its corresponding conflicts $\{c_1, c_2, \ldots, c_n\}$, we append each conflict to $I_i$, constructing $n$ different instructions $\{I_{i,1}, I_{i,2}, \ldots, I_{i,n}\}$, each containing a distinct type of conflict. Based on the conflict distribution in Table 1, we generate a total of 864 instructions, each containing a single conflict. We denote the sets of instructions containing specific conflict types as $\mathcal{I}_{CC}, \mathcal{I}_{KK}, \ldots, \mathcal{I}_{KP}$, where **CC**, **KK**, and **KP** refer to the conflict types defined earlier. We then combine $\mathcal{I}_0$ with the conflicting instructions to form nine distinct experiment subsets:

$$\mathcal{S}_{CC} = \mathcal{I}_0 \cup \mathcal{I}_{CC}, \quad \ldots, \quad \mathcal{S}_{KP} = \mathcal{I}_0 \cup \mathcal{I}_{KP}.$$

Each subset consists of 100 conflict-free instructions ($\mathcal{I}_0$) and a balanced number of instructions containing a single conflict. The subset sizes are as follows: $\mathcal{S}_{CC}, \mathcal{S}_{KK}, \mathcal{S}_{PP}, \mathcal{S}_{LL}, \mathcal{S}_{FF}, \mathcal{S}_{SS}, \mathcal{S}_{KP}$ each contain 200 instructions, while $\mathcal{S}_{PC}$ and $\mathcal{S}_{KP}$ contain 194 and 170 instructions, respectively.

| | Models | CC | KK | PP | LL | FF | SS | KP | PC | PS | IntraA | InterA | Average |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Random Guess | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 50.0 | 49.2 | 45.2 | 50.0 | 48.1 | 49.4 |
| Proprietary | GPT-4o (2024-11-20) | 91.9 | 91.3 | 88.7 | 88.1 | 79.8 | 89.8 | 75.1 | 83.7 | 76.1 | 88.3 | 78.3 | 84.9 |
| | GPT-4o-mini (2024-07-18) | 87.7 | 86.2 | 87.2 | 84.2 | 83.6 | 86.7 | 76.9 | 83.8 | 75.9 | 85.9 | 78.9 | 83.6 |
| | Claude-3.5-Sonnet (2024-10-22) | 95.7 | 93.1 | 93.1 | 90.5 | 90.5 | 93.1 | 60.3 | 89.8 | 73.6 | 92.7 | 74.6 | 86.6 |
| | Claude-3.5-Haiku (2024-10-22) | 92.5 | 88.2 | 91.9 | 85.4 | 81.9 | 92.5 | 70.5 | 85.1 | 77.4 | 88.7 | 77.6 | 85.0 |
| | Gemini-1.5-Pro-Latest | 73.5 | 72.6 | 73.5 | 73.5 | 72.6 | 73.5 | 73.1 | 71.3 | 65.4 | 73.2 | 69.9 | 72.1 |
| | Gemini-1.5-Flash-Latest | 68.7 | 67.8 | 68.7 | 68.3 | 67.8 | 68.3 | 67.4 | 66.4 | 60.6 | 68.3 | 64.8 | 67.1 |
| Open-source | Meta-Llama-3.2-1B-Instruct | 38.7 | 28.8 | 28.8 | 33.3 | 32.2 | 34.4 | 34.4 | 26.3 | 39.0 | 32.7 | 33.2 | 32.9 |
| | Meta-Llama-3.2-3B-Instruct | 49.5 | 46.3 | 46.3 | 36.9 | 38.7 | 39.6 | 41.3 | 52.6 | 43.2 | 42.9 | 45.7 | 43.8 |
| | Meta-Llama-3.1-8B-Instruct | 70.9 | 68.3 | 68.3 | 63.3 | 65.6 | 66.7 | 62.7 | 68.9 | 58.8 | 67.2 | 63.5 | 65.9 |
| | Ministral-8B-Instruct-2410 | 67.9 | 69.3 | 69.3 | 66.9 | 65.0 | 68.3 | 67.9 | 67.9 | 58.4 | 67.8 | 64.7 | 66.8 |
| | Qwen2.5-0.5B-Instruct | 36.2 | 42.2 | 43.1 | 48.6 | 47.7 | 41.2 | 43.1 | 32.2 | 42.2 | 43.2 | 39.2 | 41.8 |
| | Qwen2.5-1.5B-Instruct | 36.5 | 37.6 | 33.1 | 24.6 | 33.1 | 33.1 | 31.9 | 35.7 | 29.9 | 33.0 | 32.5 | 32.8 |
| | Qwen2.5-3B-Instruct | 59.7 | 56.1 | 54.6 | 45.9 | 50.0 | 48.4 | 54.6 | 62.9 | 46.9 | 52.5 | 54.8 | 53.2 |
| | Qwen2.5-7B-Instruct | 76.3 | 65.4 | 62.8 | 61.9 | 44.9 | 59.2 | 41.5 | 66.2 | 42.9 | 61.8 | 50.2 | 57.9 |
| | Qwen2.5-14B-Instruct | 90.2 | 80.2 | 79.5 | 79.5 | 65.8 | 81.6 | 57.7 | 83.7 | 64.3 | 79.5 | 68.6 | 75.8 |
| | Qwen2.5-32B-Instruct | 93.8 | 89.1 | 83.4 | 78.6 | 63.1 | 85.4 | 39.4 | 79.2 | 54.5 | 82.2 | 57.7 | 74.1 |

Table 2: Conflict detection results (%) of LLMs on different subsets, each containing instructions with a single type of conflict. Here, *conflict types* refer to subsets that contain the corresponding conflict, e.g., *CC* denotes $\mathcal{S}_{CC}$. 'IntraA' and 'InterA' denote the average performance across subsets of intra-constraint and inter-constraint conflicts, respectively. The reported metric is the F1-score (F1). The top two results among LLMs are highlighted in red and blue, respectively. Parenthesized numbers indicate specific dated snapshots of proprietary LLMs.

**Instructions with Multiple Conflicts.** To construct instructions with $k$ constraints ($k \in \{1, 2, 3, 4, 5, 6\}$), for each instruction $I_i \in \mathcal{I}_0$, we randomly select $k$ conflicts from its corresponding conflict set $\{c_1, c_2, \ldots, c_n\}$, shuffle them, and append them to $I_i$. Due to computational constraints, we generate a single instruction with $k$ conflicts for each $I_i$. This process results in the set $\mathcal{I}_k$, which contains 100 instructions, each with $k$ conflicts.

We will evaluate LLM performance on conflict detection and resolution across these subsets.

### 3.2 Evaluation Models

We evaluate a range of models for conflict detection and resolution, categorizing them into two primary groups: (1) Six Proprietary LLMs, including GPT-4o (gpt-4o-2024-11-20), GPT-4o-mini (gpt-4o-mini-2024-07-18) (OpenAI et al., 2023), Claude-3.5-Sonnet (claude-3-5-sonnet-20240620), Claude-3.5-Haiku (claude-3-5-haiku-20240620) (Anthropic, 2024), Gemini-1.5-Pro-Latest (gemini-1.5-pro-latest), and Gemini-1.5-Flash-Latest (gemini-1.5-flash-latest)[3] (Reid et al., 2024). (2) Ten Open-source LLMs, including Meta-Llama-3.2-1B-Instruct, Meta-Llama-3.2-3B-Instruct, Meta-Llama-3.1-8B-Instruct (Dubey et al., 2024), Mistral-8B-Instruct-2410 (Mistral, 2023), and Qwen2.5-[0.5, 1.5, 3, 7, 14, 32]B-Instruct (Yang et al., 2024). For all models, we set the maximum output length to 2048 tokens and use a temperature of 0 to ensure deterministic outputs.

[3]We used the Gemini-1.5 API in January 2025.

## 4 Conflict Detection

In this section, we explore the conflict detection task, which evaluates whether LLMs can identify conflicting instructions. Given an instruction $I$, the conflict detection task is formulated as a function $f(I) \in \{\text{Yes}, \text{No}\}$, where $f$ can be instantiated by an LLM. The prompt used for conflict detection is provided in Table 7.

### 4.1 Experiment Results on Instructions with a Single Conflict

Table 2 shows the conflict detection performance of various models. Our key findings are:

(1) **Proprietary models excel in conflict detection.** Claude-3.5-Sonnet and Claude-3.5-Haiku achieve the highest average F1 scores of 86.6% and 85.0%, respectively, followed closely by GPT-4o at 84.9%.

(2) **Open-source models with fewer than 7B parameters struggle with conflict detection.** Models such as Meta-Llama-3.2-3B-Instruct and Qwen2.5-1.5B-Instruct underperform relative to random guessing across most conflict types, indicating their inability to detect conflicts effectively.

(3) **Detecting intra-constraint conflicts is easier than inter-constraint conflicts.** For instance, Claude-3.5-Sonnet scores 92.7% on intra-constraint conflict subsets but only 74.6% on inter-constraint conflict subsets. This pattern is consistent with other strong models, suggesting that intra-constraint conflicts are more recognizable than inter-constraint conflicts.

These findings highlight the strength of proprietary LLMs in conflict detection and the challenges

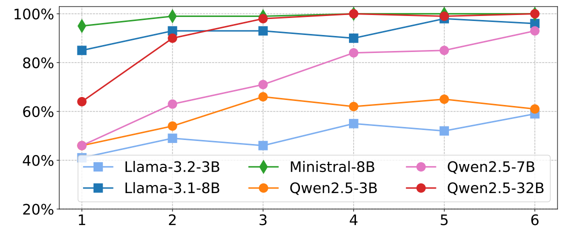



Figure 4: Conflict detection results of LLMs on different subsets $\mathcal{I}_k$, where each instruction contains $k$ conflicts. The x-axis represents the number of conflicts per instruction. The reported metric is Recall.

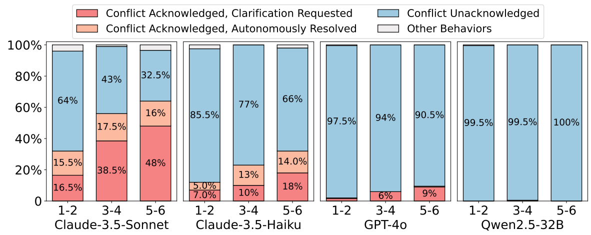



Figure 5: Distributions of conflict resolution behaviors exhibited by different LLMs when responding to instructions with varying numbers of conflicts.

faced by smaller open-source models.

### 4.2 Experiment Results on Instructions with Multiple Conflicts

Figure 4 illustrates the conflict detection performance of various LLMs as the number of conflicts in instructions increases. **As the number of conflicts within an instruction grows, models generally exhibit improved detection performance.** This trend is particularly evident in Qwen2.5-[7, 32]B. However, **smaller open-source models struggle with conflict detection.** Even when instructions contain multiple conflicts, models with fewer than 7B parameters, such as LLaMA-3.2-3B and Qwen2.5-3B, exhibit lower recall in identifying conflicts. This suggests that smaller models may lack the necessary reasoning capacity to detect conflicting constraints.

## 5 Conflict Resolution

In this section, we examine how LLMs handle instructions with conflicting constraints, simulating real-world scenarios where user instructions contain mutually contradictory requirements. We first observe LLMs' behaviors in response to such conflicts, and then analyze the effect of conflicting constraints on the original conflict-free constraints.

### 5.1 Analysis on Conflict Resolution Behaviors

**Typical Conflict Resolution Behaviors.** In Section 3.1, we create six subsets $\mathcal{I}_k$, where each instruction contains $k$ conflicts. We feed these conflicting instructions into LLMs and analyze their responses, classifying their behaviors into four types:
1. **Conflict Unacknowledged**: The model does not indicate the presence of conflicts in its response and directly provides a response to the instruction.
2. **Conflict Acknowledged, Clarification Requested**: The model recognizes that the instruction contains conflicts, refuses to respond, and explicitly asks the user for clarification.
3. **Conflict Acknowledged, Autonomously Resolved**: The model identifies conflicts, resolves them on its own, and provides a response to the resolved instruction.
4. **Other Behaviors**: The model refuses to respond for reasons unrelated to conflicts.

The first behavior is particularly problematic, as the model fails to inform users of conflicts while generating a response that satisfies only a subset of constraints. This may mislead users into accepting incomplete or incorrect responses without realizing that their instruction contains conflicts. In contrast, Behaviors 2 and 3 explicitly acknowledge the conflicts. Behavior 3 autonomously resolves them, while Behavior 2 seeks clarification from users. Among these, Behavior 2 is the most desirable, as it ensures transparency and allows users to control the conflict resolution process.

**Distribution of Conflict Resolution Behaviors.** To systematically analyze LLM behavior, we use GPT-4o to assign behavior labels to 2,400 responses from four LLMs (see Table 9 for the evaluation prompt). To check the quality of GPT-4o's assessment, we manually annotate behavior labels for 100 responses, achieving 98% agreement with GPT-4o's judgments. Figure 5 presents the distribution of conflict resolution behaviors exhibited by different LLMs when responding to instructions with varying numbers of conflicts. We summarize the key findings as follows:
**(1) GPT-4o and Qwen2.5-32B Predominantly Exhibit Behavior 1**: However, this does not imply that these LLMs lack the ability to detect conflicts. As shown in Figure 4, Qwen2.5-32B can identify conflicts with near 100% accuracy when more than two conflicts are present in an instruction. **Despite their conflict detection capabilities, they fail to explicitly acknowledge conflicts in most cases.**
**(2) Claude-3.5 models exhibit conflict-aware behavior that scales with the number of conflicts.** In Claude-3.5-Sonnet, the combined proportion of

Behaviors 2 and 3 increases from 32.0% when handling instructions with 1-2 conflicts to 64.0% when handling instructions with 5-6 conflicts. A similar trend is observed in Claude-3.5-Haiku. However, despite the presence of multiple conflicts in instructions, Behavior 1 still constitutes a significant proportion of Claude-3.5 models' responses.

**These findings underscore the necessity of enhancing LLMs to adopt safe conflict resolution behaviors when faced with conflicts, which is essential for ensuring reliable responses.**

| Model | GP | $\mathcal{I}_1$ B1↓ | $\mathcal{I}_1$ B2↑ | $\mathcal{I}_1$ B3 | $\mathcal{I}_1$ B4 | $\mathcal{I}_0$ B5↑ | $\mathcal{I}_0$ B6↓ | F1 |
|---|---|---|---|---|---|---|---|---|
| GPT-4o | ✗ | 97 | 1 | 1 | 1 | 100 | 0 | - |
| | ✔ | 4 | 96 | 0 | 0 | 60 | 40 | 81.4 |
| GPT-4o-mini | ✗ | 98 | 0 | 0 | 2 | 100 | 0 | - |
| | ✔ | 4 | 96 | 0 | 0 | 47 | 53 | 77.1 |
| Claude-3.5-Haiku | ✗ | 93 | 2 | 2 | 3 | 100 | 0 | - |
| | ✔ | 16 | 84 | 0 | 0 | 78 | 22 | 81.6 |

Table 3: Distribution (%) of LLM behaviors with or without the guiding prompt (GP) designed to detect and resolve instruction conflicts using Behavior 2. Here, B refers to behavior types. $\mathcal{I}_1$ and $\mathcal{I}_0$ represent instructions with one conflict and without conflicts, respectively. For conflict-free instructions ($\mathcal{I}_0$), we report two types of model behaviors: **Behavior 5** (LLMs determine that the instruction has no conflict and executes it directly) and **Behavior 6** (LLMs incorrectly detect conflicts and unnecessarily asks for clarification). F1 denotes the F1 score of LLMs in identifying instruction conflicts when using the GP. Results highlighted in green and red indicate whether the behavioral changes meet or fail to meet expectations, respectively.

## 5.2 Prompting LLMs to Resolve Instruction Conflicts Using Desired Behaviors

LLMs often fail to explicitly acknowledge conflicting instructions. This study investigates whether prompt engineering can guide LLMs to identify and resolve such conflicts according to desired behavioral patterns. To explore this, we prepend user instructions with the prompt designed to detect and resolve instruction conflicts with Behavior 2, as detailed in Table 8. As shown in Table 3, this prompt can effectively induce LLMs to adopt the predefined desired Behavior 2 (acknowledging conflict and requesting clarification). However, it also causes LLMs to behave overly conservatively, asking for clarification even when no conflict exists (Behavior 6), thereby degrading the user experience. These findings suggest that **while prompt engineering can influence conflict resolution behavior, it alone is insufficient for achieving both desired conflict resolution and accurate execution of non-conflicting instructions.**

| Constraints | Content | Keyword | Phrase | Style | Length | Format |
|---|---|---|---|---|---|---|
| Consistency | 92% | 88% | 100% | 92% | 96% | 100% |

Table 4: Consistency between GPT-4o and human evaluations across different constraint types in the instruction-following task.

## 5.3 Analysis on Constraint Priority

**Constraint-Following Ability of LLMs on Conflict-Free Instructions.** We first feed each conflict-free instruction $I_i \in \mathcal{I}_0$ into LLMs and evaluate their **Constraint Satisfaction Rate (CSR)** in the absence of conflicting constraints. CSR is defined as follows:

$$CSR = \frac{1}{M} \sum_{i=1}^{N} \sum_{j=1}^{l_i} I_i^j, \qquad (1)$$

where $I_i^j = 1$ if the $j$-th constraint of the $i$-th instruction is satisfied and $I_i^j = 0$ otherwise. Here, $l_i$ denotes the number of constraints in $I_i$, $N$ represents the number of instructions, and $M$ is the total number of constraints across all instructions.

We use GPT-4o to evaluate whether the model's output satisfies the specified constraints (the evaluation prompt is shown in Table 10). To assess the evaluation quality of GPT-4o, we manually labeled 150 constraints and then verified whether each constraint was satisfied in LLMs' responses. Table 4 shows that automatic evaluation aligns closely with human judgment. Figure 6 presents the CSR results of seven LLMs across different constraint types, revealing a clear performance pattern: the CSR score is notably lowest for length constraints but higher for the other five constraint types.

**Impact of Conflicting Constraints on LLMs' Constraint-Following Ability.** As shown in Figure 5, when an instruction contains only a few conflicts, LLMs predominantly exhibit Behavior 1, meaning they tend to satisfy some of the constraints in the instruction. To further investigate how **N**ewly introduced conflicting **C**onstraints (NC) affect LLMs' ability to adhere to **O**riginal **C**onstraints (OC), we focus on instructions containing a single conflict. Given computational constraints, we examine three types of conflicts: CC, KK, and PP. To examine the effect of NC's position, we construct two subsets for each conflict type:

- **NCA subsets** ($\mathcal{I}_{CC}, \mathcal{I}_{KK}, \mathcal{I}_{PP}$): NC is introduced **after** OC.

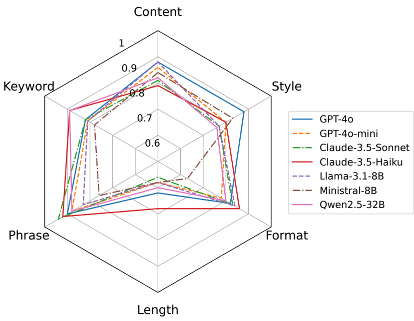
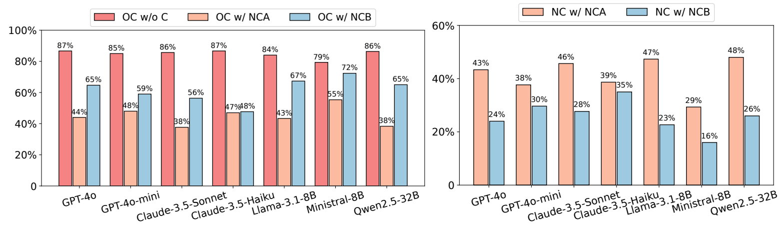



Figure 6: CSR results of various LLMs across different constraint types.




(a) Constraint satisfaction rates of OC (b) Constraint satisfaction rates of NC

Figure 7: The impact of the order of NC on the constraint satisfaction rates of both OC and NC. 'w/o C' denotes the absence of conflicts, while 'NCA' and 'NCB' indicate that NC appears after and before OC, respectively.

- **NCB subsets** ($\mathcal{I}'_{CC}, \mathcal{I}'_{KK}, \mathcal{I}'_{PP}$): NC is introduced **before** OC.

Each subset contains 100 single-conflict instructions. We input these into LLMs and use GPT-4o to evaluate whether their responses satisfy OC or NC (see Table 11 for the evaluation prompt). To validate the reliability of GPT-4o's assessment, we manually annotated 100 conflicting cases, achieving 90% agreement with GPT-4o's judgments.

Figure 7(a) shows the impact of NC on LLMs' ability to satisfy OC. The results reveal the following key observations: (1) **NC significantly reduces OC satisfaction rates**, suggesting that newly introduced constraints interfere with previously given ones. (2) **The order of NC matters.** OC is more likely to be followed when NC appears before OC rather than after it. This suggests that the **later a constraint appears in an instruction, the more likely it is to be followed**. As shown in Figure 7(b), NC is more likely to be followed when it appears **later** (NCA) rather than earlier (NCB), further reinforcing the idea that **constraints appearing later in an instruction are more likely to be satisfied**.

# 6 Related Work

## 6.1 Controllable Text Generation

Controllable text generation focuses on guiding language models to generate text with specific attributes, such as sentiment (Keskar et al., 2019; Dathathri et al., 2020), lexical constraints (He, 2021; He and Li, 2021; He et al., 2022), length (Kikuchi et al., 2016; Fan et al., 2018). Recent studies have constructed data based on these controllable tasks to evaluate (Zhou et al., 2023a; Sun et al., 2023) or enhance the instruction-following ability of LLMs (Zhou et al., 2023b). Unlike prior work, which assumes that all constraints within instructions are consistent, we assess LLMs' ability to detect and resolve conflicting constraints, offering new insights into their behavior when handling instructions with conflicts.

## 6.2 Conflict Detection

Conflict detection has been extensively studied in natural language inference (Bowman et al., 2015; Williams et al., 2018) and fact verification (Thorne et al., 2018), aiming to detect contradictions between two statements or between claims and external evidence sources. More recently, research has expanded to detecting conflicts among retrieved documents (Jiayang et al., 2024), or discrepancies between LLMs' parametric knowledge and retrieved documents (Chen et al., 2022; Neeman et al., 2023; Xie et al., 2024). Meanwhile, hallucination detection in LLMs (Manakul et al., 2023; Min et al., 2023) investigates false or misleading content generated by LLMs. While these studies explore different aspects of conflict detection, they do not focus on conflicting instructions where multiple constraints contradict each other. Our work extends beyond these domains by systematically evaluating how LLMs detect and resolve explicit conflicts within user instructions.

# 7 Conclusion

We introduce ConInstruct, a benchmark designed to evaluate LLMs' ability to detect and resolve conflicting constraints within instructions. Our findings reveal that while proprietary LLMs demonstrate strong conflict detection capabilities, they often fail to explicitly communicate conflicts to users, instead generating responses that only partially satisfy the given constraints. This highlights a critical gap in instruction-following: despite recognizing conflicts, LLMs struggle to transparently convey them. Future research should focus on enhancing LLMs' ability to explicitly notify users of conflicts and seek clarification, improving their reliability in real-world applications that demand precise adherence to instructions.

## 8 Limitations

Despite the insights provided by ConInstruct, our study has several limitations. While our benchmark covers a diverse range of constraints and conflicts, it may not fully encompass all possible forms of instruction inconsistencies. In designing conflicting constraints, we prioritized the feasibility of evaluating constraint satisfaction using LLMs or automated programs. This consideration led us to avoid overly complex constraints and ambiguous conflicts. Another limitation is that our dataset primarily focuses on text-based instructions, restricting its applicability to multimodal scenarios where conflicts may arise in image, audio, or video-based instructions. Future work should explore extending our benchmark to assess how LLMs handle conflicting constraints in multimodal settings.

## References


AI Anthropic. 2024. Claude 3.5 sonnet. *Anthropic AI*.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Hung-Ting Chen, Michael Zhang, and Eunsol Choi. 2022. Rich knowledge sources bring complex knowledge conflicts: Recalibrating models to reflect conflicting evidence. In *EMNLP*, pages 2292–2307, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

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

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. 2023. Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski, and Rosanne Liu. 2020. Plug and play language models: A simple approach to controlled text generation. In *ICLR*.

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

Angela Fan, David Grangier, and Michael Auli. 2018. Controllable abstractive summarization. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 45–54, Melbourne, Australia. Association for Computational Linguistics.

Yilin Geng, Haonan Li, Honglin Mu, Xudong Han, Timothy Baldwin, Omri Abend, Eduard Hovy, and Lea Frermann. 2025. Control illusion: The failure of instruction hierarchies in large language models. *arXiv preprint arXiv:2502.15851*.

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

Xingwei He. 2021. Parallel refinements for lexically constrained text generation with BART. In *EMNLP*, pages 8653–8666, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xingwei He, Yeyun Gong, A-Long Jin, Weizhen Qi, Hang Zhang, Jian Jiao, Bartuer Zhou, Biao Cheng, Sm Yiu, and Nan Duan. 2022. Metric-guided distillation: Distilling knowledge from the metric to ranker and retriever for generative commonsense reasoning. In *EMNLP*, pages 839–852, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Xingwei He and Victor OK Li. 2021. Show me how to revise: Improving lexically constrained sentence generation with xlnet. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12989–12997.

Xingwei He, Zhenghao Lin, Yeyun Gong, A-Long Jin, Hang Zhang, Chen Lin, Jian Jiao, Siu Ming Yiu, Nan Duan, and Weizhu Chen. 2024b. AnnoLLM: Making large language models to be better crowdsourced annotators. In *NAACL*, pages 165–190, Mexico City, Mexico. Association for Computational Linguistics.

Xingwei He and Siu Ming Yiu. 2022. Controllable dictionary example generation: Generating example sentences for specific targeted audiences. In *ACL*, pages 610–627, Dublin, Ireland. Association for Computational Linguistics.

Furong Jia, Kevin Wang, Yixiang Zheng, Defu Cao, and Yan Liu. 2024. Gpt4mts: Prompt-based large language model for multimodal time-series forecasting. In *AAAI*, volume 38, pages 23343–23351.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. 2024. FollowBench: A multi-level fine-grained constraints following benchmark for large language models. In *ACL*, pages 4667–4688, Bangkok, Thailand. Association for Computational Linguistics.

Cheng Jiayang, Chunkit Chan, Qianqian Zhuang, Lin Qiu, Tianhang Zhang, Tengxiao Liu, Yangqiu Song, Yue Zhang, Pengfei Liu, and Zheng Zhang. 2024. ECON: On the detection and resolution of evidence conflicts. In *EMNLP*, pages 7816–7844, Miami, Florida, USA. Association for Computational Linguistics.

Nitish Shirish Keskar, Bryan McCann, Lav R Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Yuta Kikuchi, Graham Neubig, Ryohei Sasano, Hiroya Takamura, and Manabu Okumura. 2016. Controlling output length in neural encoder-decoders. In *EMNLP*, pages 1328–1338, Austin, Texas. Association for Computational Linguistics.

Potsawee Manakul, Adian Liusie, and Mark Gales. 2023. SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. In *EMNLP*, pages 9004–9017, Singapore. Association for Computational Linguistics.

Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained atomic evaluation of factual precision in long form text generation. In *EMNLP*, pages 12076–12100, Singapore. Association for Computational Linguistics.

AI Mistral. 2023. Mixtral of experts: A high quality sparse mixture-of-experts. *Mistral AI*.

Ella Neeman, Roee Aharoni, Or Honovich, Leshem Choshen, Idan Szpektor, and Omri Abend. 2023. DisentQA: Disentangling parametric and contextual knowledge with counterfactual question answering. In *ACL*, pages 10056–10070, Toronto, Canada. Association for Computational Linguistics.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, et al. 2023. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *NIPS*, volume 35, pages 27730–27744. Curran Associates, Inc.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. 2024. InFoBench: Evaluating instruction following ability in large language models. In *Findings of ACL*, pages 13025–13048, Bangkok, Thailand. Association for Computational Linguistics.

Machel Reid, Nikolay Savinov, Denis Teplyashin, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *ArXiv*, abs/2403.05530.

Jiao Sun, Yufei Tian, Wangchunshu Zhou, Nan Xu, Qian Hu, Rahul Gupta, John Wieting, Nanyun Peng, and Xuezhe Ma. 2023. Evaluating large language models on controlled generation tasks. In *EMNLP*, pages 3155–3168, Singapore. Association for Computational Linguistics.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *NAACL*, pages 809–819, New Orleans, Louisiana. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Eric Wallace, Kai Xiao, Reimar Leike, Lilian Weng, Johannes Heidecke, and Alex Beutel. 2024. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NIPS*, volume 35, pages 24824–24837. Curran Associates, Inc.

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*.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Jian Xie, Kai Zhang, Jiangjie Chen, Renze Lou, and Yu Su. 2024. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In *ICLR*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan

Qiu. 2024. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*.

Qianru Zhang, Haixin Wang, Cheng Long, Liangcai Su, Xingwei He, Jianlong Chang, Tailin Wu, Hongzhi Yin, Siu Ming Yiu, Qi Tian, and Christian S. Jensen. 2024a. A survey of generative techniques for spatial-temporal data mining. *arXiv preprint arXiv:2405.09592*.

Tao Zhang, Yanjun Shen, Wenjing Luo, Yan Zhang, Hao Liang, Fan Yang, Mingan Lin, Yujing Qiao, Weipeng Chen, Bin Cui, et al. 2024b. Cfbench: A comprehensive constraints-following benchmark for llms. *arXiv preprint arXiv:2408.01122*.

Zhihan Zhang, Shiyang Li, Zixuan Zhang, Xin Liu, Haoming Jiang, Xianfeng Tang, Yifan Gao, Zheng Li, Haodong Wang, Zhaoxuan Tan, Yichuan Li, Qingyu Yin, Bing Yin, and Meng Jiang. 2025. IHEval: Evaluating language models on following the instruction hierarchy. In *In NAACL*, pages 8374–8398, Albuquerque, New Mexico. Association for Computational Linguistics.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *NIPS*, volume 36, pages 46595–46623. Curran Associates, Inc.

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

Wangchunshu Zhou, Yuchen Eleanor Jiang, Ethan Wilcox, Ryan Cotterell, and Mrinmaya Sachan. 2023b. Controlled text generation with natural language instructions. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pages 42602–42613. PMLR.

# A Constraint Dimensions

## A.1 Content Constraints

Content constraints involve incorporating specific details related to the content. These details may encompass various aspects, such as reasons, purposes, topics, background information, budgets, targets, and more.

## A.2 Keyword Constraints

**1. Keywords Inclusion:** This constraint specifies the inclusion of specific keywords. Examples of this constraint:

- Include all of the following keywords in the response: {keyword list}.
- Include at least/at most/exactly {N} of the following keywords in the response: keyword list.
- Include either {keyword1} or {keyword2}, but not both, in the response.

**2. Forbidden Words:** This constraint specifies keywords that must not be included. Examples of this constraint:

- Do not include the following forbidden keywords in the response: {forbidden keyword list}.

**3. Keyword Frequency:** This constraint specifies the frequency of keywords. Examples of this constraint:

- The keyword {keyword} must appear at least/at most/exactly {N} times in the response.

**4. Letter Frequency:** This constraint specifies the frequency of letters. Examples of this constraint:

- The letter {letter} must appear at least/at most/exactly {N} times in the response.

**5. Keyword Order:** This constraint specifies the order of keywords. Examples of this constraint:

- The keyword {keyword1} must appear before/after the keyword {keyword2} in the response.

**6. Keyword Proximity:** This constraint specifies the distance between keywords. Examples of this constraint:

- The keyword {keyword1} must appear at least/at most/exactly {N} words/sentences/paragraphs away from the keyword {keyword2}.
- The keywords {keyword list} must/must not appear in the same paragraph/sentence.

**7. Keyword Position:** This constraint specifies the positions of keywords. Examples of this constraint:

- The keywords {keyword list} must/must not appear in the first/last/n-th paragraph/sentence of the response.

**8. Keyword Part-of-speech:** This constraint specifies the part-of-speech tag for keywords, which possess multiple part-of-speech tags in the Oxford Dictionary. Do not apply this constraint to keywords with only a single part-of-speech tag. Examples of this constraint:

- The keyword {keyword} must appear in the response and used as {part-of-speech tag in the Oxford Dictionary}.

**9. Keyword Definition:** This constraint specifies the definition for keywords, which possess multiple definitions in the Oxford Dictionary. This constraint can only be used to verbs or adjectives. Do not apply this constraint to keywords with fewer than three definitions. Examples of this constraint:

- The keyword {verb or adjective} must appear in the response and convey the specified definition {definition in the Oxford Dictionary}.

## A.3 Phrase Constraints

**1. Phrase Inclusion:** This constraint specifies the inclusion of specific phrases. The specific phrase must contain at least four words. Examples of this constraint:

- Include the phrase {phrase} in the response.

**2. Phrase Frequency:** This constraint specifies the frequency of phrases. The specific phrase must contain at least four words. Examples of this constraint:

- The phrase {phrase} must appear at least/at most/exactly {N} times in the response.

**3. Phrase Position:** This constraint specifies the positions of keywords. The specific phrase must contain at least four words. Examples of this constraint:

- Start/Finish the response/n-th paragraph with the phrase {phrase}.
- Include the phrase {phrase} in n-th paragraph.

### A.4 Length Constraints

**1. Number of Paragraphs:** This constraint specifies the required number of paragraphs in the response. Examples of this constraint:

- The response must contain at least/at most/exactly {N} paragraphs.

**2. Number of Sentences:** This constraint specifies the required number of sentences in the response or within specific paragraphs. Examples of this constraint:

- The response must contain at least/at most/exactly {N} sentences.
- The n-th paragraph must contain at least/at most/exactly {N} sentences.
- Each paragraph must contain at least/at most/exactly {N} sentences.

**3. Number of Words:** This constraint specifies the required number of words in the response, or within specific paragraphs or sentences. Examples of this constraint:

- The response must contain at least/at most/exactly {N} words.
- The n-th paragraph/sentence must contain at least/at most/exactly {N} words.
- Each paragraph/sentence must contain at least/at most/exactly {N} words.

### A.5 Format Constraints

**1. JSON Format:** This constraint requires the entire response to be wrapped in JSON format and follow specific JSON structure.

- The response must include the following keys: {key list}.
- The response must include at least/at most/exactly {N} of the following types: Number, String, Boolean, Array, or Object.
- The value of the key {key} must be a Number/String/Boolean/Array/Object.
- The value of the key {key} must be an integer equal to/less than/greater than {N}.
- The value of the key {key} must be an Array/Object containing at least/at most/exactly {N} elements.

**2. Markdown Format:** This constraint requires the entire response to follow specific Markdown formats. Examples of this constraint:

- The response must include at least/at most/exactly {N} headers at level {M}.
- The response must include at least/at most/exactly {N} ordered/unordered lists. Each list must include at least/at most/exactly {M} items.
- The response must include at least/at most/exactly {N} code blocks formatted with triple backticks (```) and a specified language (e.g., ```python).
- The response must include at least/at most/exactly N horizontal rules, formatted as – or ***.
- The response must include at least/exactly N hyperlinks formatted as [text](URL).

**3. Bullet Format:** This constraint specifies the requirements for bullet points. Examples of this constraint:

- Format specific content (e.g., reasons, contributions, purposes, and names) into a bulleted list containing at least/at most/exactly {N} points.
- Each bullet point must include at least/at most/exactly {N} words/sentences.
- Each bullet point must begin/end with a specific keyword/phrase: {keyword/phrase}.

**4. Language Format:** This constraint specifies the language requirements. Examples of this constraint:

- The entire response must be written exclusively in {language, such as Chinese, English}.
- The response must include a {language, such as Chinese, English} idiom/ancient poem.

**5. Case Sensitivity:** This constraint defines the required case for words. Examples of this constraint:

- Write all words in lowercase/uppercase case.
- The response must include at least/at most/exactly {N} lowercase/uppercase words.

### A.6 Style Constraints

**1. Rhetorical Style Constraints:** This constraint specifies the rhetorical style to be used. Examples of this constraint:

- Include rhetorical questions to engage the audience.
- Conclude with a strong call to action.

**2. Tone and Emotion Constraints:** This constraint specifies the tone or emotion of the response. Examples of this constraint:

- Write the response in a {tone/emotion, e.g., positive/neutral/negative/academic/persuasive/humorous/sarcastic} style suitable for a {field/topic, e.g., motivational speech}.
- Use short, punchy sentences to create urgency and excitement.
- Convey empathy/sincerity/urgency in the response.
- Use a neutral/optimistic/pessimistic tone throughout the response.
- The response must be academic/persuasive/humorous/sarcastic.

**3. Voice Constraints:** This constraint specifies whether the response should use active or passive voice. Examples of this constraint:

- Write the response in active/passive voice.
- The response must include at least/at most/exactly {N} sentences in passive/active voice.

**4. Sentence Structure Constraints:** This constraint specifies the complexity or structure of sentences. Examples of this constraint:

- The response/n-th paragraph must include at least/at most/exactly {N} simple/compound/complex sentences.

**5. Sentence Type Constraints:** This constraint specifies the type of sentences. Examples of this constraint:

- The response/n-th paragraph must include at least/at most/exactly {N} declarative/interrogative/exclamatory/imperative sentences.

**6. Readability Constraints:** This constraint specifies the readability of the response. Examples of this constraint:

- Tailor the response for specific audience (e.g., children, laypersons, professionals, experts).
- Use at least/at most/exactly {N} technical terms related to {field/topic}.
- The response must simplify technical jargon, providing explanations for terms.
- Avoid using jargon/slang/archaic words.

**7. Person Constraints:** This constraint specifies the narrative perspective to be used. Examples of this constraint:

- The response must be written in the first/second/third person.
- Avoid using personal pronouns.
- Include at least {N} sentences addressing the reader directly.

**8. Miscellaneous Style Constraints:** Covers specific stylistic choices not covered above. Examples of this constraint:

- Mimic the writing style of {author/speaker}.
- Include at least {N} metaphors/similes in the response.

## B Conflict Types

### 1. Conflicts between Content Constraints (CC)

- Definition: Conflicts occur when two content requirements contradict each other.
- Example 1: The itinerary must exclude any mention of national parks vs. The itinerary must include national parks.
  Explanation: One constraint says not to mention national parks, while the other requires them to be included.

### 2. Conflicts between Keyword Constraints (KK)

- Definition: Conflicts arise when keyword-related rules are in opposition.
- Example 1: Include the keyword "like" vs. Avoid using the keyword "like."
  Explanation: The instructions directly contradict each other, as one demands the use of the keyword and the other forbids it.
- Example 2: The keyword "resignation" must appear at least three times vs. Do not include the keyword "resignation."
  Explanation: One rule requires the keyword "resignation" to appear multiple times, while the other explicitly bans it.
- Example 3: The keyword "strategy" must appear before the keyword "quality" vs. The keyword "quality" must appear before the keyword "strategy."
  Explanation: This is a conflict of word order, where one rule demands "strategy" precedes "quality" and the other dictates the opposite.
- Example 4: Use the keyword "bank" as a verb vs. Use the keyword "bank" as a noun.
  Explanation: The word "bank" is given different roles in each constraint, making them incompatible.
- Example 5: The keywords "transaction" and "clarification" must appear in the same paragraph, with no more than five words separating them vs. The keywords "transaction" and "clarification" must appear in the same paragraph, with at least six words separating them.
  Explanation: The two rules conflict in terms of the allowable distance between the two keywords.

### 3. Conflicts between Phrase Constraints (PP)

- Definition: Conflicts where different rules dictate how phrases should appear.
- Example 1: The first paragraph starts with the phrase "Embark on an unforgettable journey" vs. The first paragraph starts with the phrase "Begin an unforgettable journey."
  Explanation: The rules conflict in terms of how the first paragraph should begin, requiring different phrases.
- Example 2: The first paragraph starts with the phrase "Embark on an unforgettable journey" vs. Do not include the phrase "Embark on an unforgettable journey."
  Explanation: One rule mandates the phrase to appear at the beginning, while the other forbids its use.
- Example 3: The first paragraph starts with the phrase "Embark on an unforgettable journey" vs. The first paragraph should not start with the phrase "Embark on."
  Explanation: The first rule dictates that the paragraph must start with "Embark on an unforgettable journey," while the second rule prohibits starting the paragraph with any phrase that begins with "Embark on."

### 4. Conflicts between Length Constraints (LL)

- Definition: Conflicts occur when constraints are in opposition regarding the length or size of elements (e.g., word count, number of sentences, number of paragraphs).
- Example 1: The email must contain exactly five paragraphs, with each paragraph consisting of at least 80 words vs. The email must contain at most 300 words.
  Explanation: If there are exactly five paragraphs with a minimum of 80 words each, the total word count exceeds 300, making the two rules incompatible.
- Example 2: The email must contain exactly five paragraphs vs. The email must contain at most four paragraphs.
  Explanation: The first rule requires five paragraphs, while the second limits it to four.
- Example 3: Each paragraph consists of at least 100 words vs. The first paragraph contains

four sentences, with each consisting of at most 20 words.
Explanation: If each sentence in the first paragraph has at most 20 words, the total word count will not exceed 80. This directly conflicts with the rule requiring each paragraph to contain at least 100 words.

- Example 4: Each paragraph consists of at least 100 words vs. The first paragraph has between 50 and 80 words.
Explanation: One rule requires the paragraph to have at least 100 words, while the other limits it to a smaller word count.

- Example 5: The email must contain exactly five paragraphs, with each paragraph consisting of at least five sentences vs. The email must contain at most 20 sentences.
The first rule requires at least 25 sentences (5 paragraphs × 5 sentences), which conflicts with the second rule, which limits the total to 20 sentences.

**5. Conflicts between Format Constraints (FF)**

- Definition: Conflicts between different formatting requirements.

- Example 1: The response must include at least two level-1 headers vs. The response can only use level-2 headers.
Explanation: One rule requires level-1 headers, while the other forbids them, limiting the response to only level-2 headers.

- Example 2: The email must be formatted in JSON with the following keys: "subject", "body", and "signature" vs. The email must be formatted in JSON with two keys.
Explanation: One rule requires three keys, while the other allows only two.

- Example 3: Present the fine dining recommendations in a bulleted list of exactly three points vs. Avoid using bullet points.
Explanation: The first rule requires bullet points, while the second forbids them.

- Example 4: The email must be written exclusively in English vs. The email must include a Chinese idiom.
Explanation: One rule requires the email to be only in English, while the other mandates the inclusion of a Chinese idiom, presented in Chinese.

- Example 5: The email should include at least five uppercase words vs. The email must be written in lowercase.
Explanation: One rule requires uppercase words, while the other specifies that everything must be in lowercase.

**6. Conflicts between Style Constraints (SS)**

- Definition: Conflicts arise when different stylistic rules are at odds with each other.

- Example 1: The response must be written in the first person vs. The response must be written in the second person.
Explanation: The two rules conflict because one requires a first-person perspective, while the other demands a second-person perspective.

- Example 2: Ensure the email is written in a formal tone vs. Ensure the email is written in an informal tone.
Explanation: One rule demands a formal tone, while the other requires an informal one.

- Example 3: Tailor the response for laypersons vs. Tailor the response for experts.
Explanation: The two rules conflict because they require the response to be suitable for different audiences: laypersons and experts.

**7. Conflicts between Keyword Constraints and Phrase Constraints (KP)**

- Definition: Conflicts between specific keywords and larger phrases.

- Example 1: Refrain from using the keyword "unforgettable" vs. The first paragraph starts with the phrase "Embark on an unforgettable journey."
Explanation: The first rule forbids the use of "unforgettable", while the second requires it as part of a phrase.

- Example 2: The first paragraph starts with the phrase "Embark on an unforgettable journey" vs. The keyword "unforgettable" must appear before "embark."
Explanation: The first rule specifies that the phrase "Embark on an unforgettable journey" should be used, with "embark" coming first. However, the second rule requires that the word "unforgettable" must precede "embark," creating an ordering conflict.

**8. Conflicts between Phrase Constraints and Content Constraints (PC)**

- Definition: Conflicts arise when specific phrase requirements contradict broader content or thematic requirements, making it impossible to adhere to both at the same time.

- Example 1: The email must include the phrase "Thank you for your business" vs. The email should not express gratitude or appreciation
  Explanation: The first rule requires a specific phrase expressing gratitude, while the second rule prohibits any expression of gratitude, making these content and phrase constraints incompatible.

- Example 2: The response must contain the phrase "A visit to Yellowstone National Park is a must" vs. The itinerary must exclude any mention of national parks
  Explanation: The first rule mandates mentioning Yellowstone National Park, while the second rule explicitly forbids mentioning any national parks, creating a direct conflict.

- Example 3: The introduction must include the phrase "We guarantee the lowest prices" vs. The content must not make any guarantees or promises
  Explanation: The first rule demands a specific phrase that guarantees low prices, while the second rule forbids making guarantees, creating a contradiction.

**9. Conflicts between Phrase Constraints and Style Constraints (PS)**

- Definition: Conflicts arise when specific phrase requirements contradict stylistic requirements, such as tone, perspective, or formality.

- Example 1: The response must include the phrase: "I strongly believe this is the best approach." vs. The response must be written in the second person
  Explanation: The phrase uses first-person perspective ("I strongly believe"), but the style constraint requires the response to be in the second person.

- Example 2: The response must include the phrase: "Hey buddy, this is gonna be awesome!" vs. The response must be written in a formal tone
  Explanation: The required phrase is informal and conversational, but the style constraint demands a formal tone.

- Example 3: The response must include the phrase: "The stochastic process adheres to a Markovian property." vs. The response must be tailored for laypersons
  Explanation: The phrase contains technical jargon suited for experts, but the style constraint requires the response to be accessible to laypersons.

- Example 4: The response must include the phrase: "This is the worst decision ever made." vs. The response must maintain a neutral and unbiased tone
  Explanation: The required phrase expresses a strong negative opinion, contradicting the neutrality constraint.

## C Prompt Templates

I currently have a simple seed instruction (i.e., [Seed Instruction]). Your task is to make it more complex by adding additional constraints. To assist you in completing this task, I will provide six types of constraints for your reference (i.e., [Reference Constraints]), which include ‘Content Constraints’, ‘Keyword Constraints’, ‘Phrase Constraints’, ‘Length Constraints’, ‘Format Constraints’, and ‘Style Constraints’. Each type of constraint includes several different sub-constraints. For example, ’Format Constraints’ consist of five sub-constraints: JSON Format, Markdown Format, Bullet Format, Language Format, and Case Sensitivity. For each sub-constraint, we will first provide a definition followed by example templates. You may choose a suitable template from these examples or create your own, as long as it satisfies the sub-constraint’s definition. Ensure that you strictly apply all sub-constraints from each type without prioritizing any particular one.

Below is [Reference Constraints].
**[Reference Constraints]**
{*Constraints in §A*}.

Below is the requirements for modifying the seed instruction.
**[Requirements]:**

1. You must use all constraints from [Reference Constraints].

2. Feel free to use any constraints other than [Reference Constraints] that you deem appropriate.

3. When adding constraints to the seed instruction, you are free to combine and paraphrase the selected constraints as needed. Seamlessly integrate these constraints into the seed instruction without omitting any key information, and avoid directly listing the selected constraints.

4. If the seed instruction is a question, please do not modify the seed instruction. Add constraints after the seed instruction.

5. Directly output the modified instruction (the instruction with added constraints in plain text format, i.e., [Modified Instruction]), without any analysis. The modified instruction must not contain line breaks.

Below is the seed instruction.
**[Seed Instruction]:** {*Seed Instruction*}
**[Modified Instruction]:**

Table 5: Prompt template for expanding seed instructions.

I currently have an instruction (i.e., [Instruction]) that includes multiple constraints, all of which can be satisfied simultaneously. Your task is to add new constraints to this instruction. These new constraints should conflict with the existing ones in the given instruction, meaning they cannot be satisfied at the same time. However, the new constraints themselves must not conflict with one another. To assist you in completing this task, I will provide six types of constraints for your reference (i.e., [Reference Constraints]), which include 'Content Constraints', 'Keyword Constraints', 'Phrase Constraints', 'Length Constraints', 'Format Constraints', and 'Style Constraints'. Each type of constraint includes several different sub-constraints. For example, 'Format Constraints' consist of five sub-constraints: JSON Format, Markdown Format, Bullet Format, Language Format, and Case Sensitivity.

Below is [Reference Constraints].
**[Reference Constraints]**
{*Constraints in §A*}.

Below are the types of conflicts and their corresponding examples:
**[Conflict Types]**
{*Conflict types in §B*}.

When adding conflicting constraints into the given instruction, you must adhere to the following requirements:
**[Requirements]:**

1. Select constraints exclusively from [Reference Constraints].
2. The selected constraints must conflict with at least one constraint in the provided instruction. However, there must be no internal conflicts among the selected constraints. Do not include constraints that do not conflict with the given instruction's existing constraints.
3. The conflicts between the selected constraints and the existing constraints in the given instruction must be explicit and unambiguous.
4. The conflicts must cover all nine conflict types in [Conflict Types], with each conflict type containing one pair of conflicting constraints.
5. The conflicting examples in [Conflict Types] and the example below are provided to aid your understanding. However, I would prefer that you not rely solely on these examples. Please come up with a wider variety of conflict scenarios.
6. The output must strictly adhere to the format shown in the example below.

Here is an example:
{*In-Context Example*}

**[Instruction]:** {*Instruction*}

Table 6: Prompt template for adding conflicting instructions.

I currently have an instruction (i.e., [Instruction]) that includes multiple constraints. Please help me judge whether the instruction contains conflicting constraints, which cannot be satisfied simultaneously.

Answer "YES" if you think the [Instruction] contains conflicting constraints or requirements; otherwise, answer "NO."

Please first provide an explanation and then give the answer. Your output should follow the format below:
**[Explanation]**: Your explanation
**[Answer]**: YES / NO

**[Instruction]:** {*Instruction*}

Table 7: Prompt template for conflict detection.

I currently have an instruction (i.e., [Instruction]) that includes multiple constraints. Please analyze whether any of these constraints are in conflict and cannot be satisfied simultaneously. If no such conflicts are found, proceed to execute the instruction. However, if conflicting constraints are detected, refrain from executing the instruction. Instead, clearly indicate the presence of conflicting constraints and request further clarification.

**[Instruction]:** {*Instruction*}

Table 8: Prompt template for instruction-following with an expected conflict resolution behavior.

You will be given an Instruction containing multiple constraints, some of which are conflicting and cannot be simultaneously satisfied. You will also be provided with a Model Response, which is generated based on the given Instruction.

Your task is to evaluate the Model Response and determine whether the model has recognized the conflicting constraints within the Instruction when generating its response. Follow these evaluation rules:

1. If the model does not acknowledge the conflicting constraints and directly provides a response to the instruction, output 1.

2. If the model identifies the conflicting constraints, does not provide a response to the instruction, and explicitly asks the user to clarify the instruction, output 2.

3. If the model identifies the conflicting constraints, resolves them on its own, and then provides a response to the resolved instruction, output 3.

4. If the model's response does not fit into any of the above categories, output 4.

Your output must strictly follow this format:
**## Output Format**
**Analysis**: A brief analysis of the Model Response.
**Answer**: The evaluation result (1, 2, 3, or 4) without any additional content.

**## Evaluation Input**
**Model Response**
{*Model Response*}

**Instruction**
{*Instruction*}

**## Evaluation Output**

Table 9: Prompt template for evaluating LLM behavior when handling conflicting instructions.

Your task is to act as a quality evaluator, analyzing the content of the Model Response to determine whether it fully satisfies the requirements outlined in the Instruction. When evaluating, you should adhere to the following judgment criteria:
1. Answer "YES" if the Model Response entirely fulfills all the requirements specified in the instruction.
2. Answer "NO" if the Model Response fails to meet all the requirements or provides no relevant information for the given instruction.

**## Evaluation Steps**
Please analyze the Model Response and Instruction carefully, adhering to the following steps:
Step 1: Analyze the Instruction, then extract relevant content from the Model Response. Copy sentences from the Model Response exactly as they are, without any modification.

1. If the instructions include constraints related to keywords or phrases (e.g., keyword definitions, keyword frequency, or phrase frequency), extract the sentences containing the specified keywords or phrases. Record the positions of these keywords or phrases, if necessary.

2. If the instructions include constraints related to specific information or topics, extract segments containing the relevant information or topics.

3. If the instructions include constraints related to output formats or styles, extract segments that reflect the specified formats or styles.

Step 2: Analyze whether the Instruction's constraints are fully satisfied.
Step 3: Provide your evaluation answer ("YES" or "NO"), without adding extra content.

**## Output Format**
**Step 1**: The extracted content from the Model Response.
**Step 2**: A brief analysis of the Instruction.
**Step 3**: YES / NO

**## Evaluation Input**
**Model Response**
{*Model Response*}

**Instruction**
{*Instruction*}

**## Evaluation Output**

Table 10: Prompt template for instruction-following evaluation.

You will be provided with the following:

1. Two instructions, Instruction 1 and Instruction 2.

2. A Model Response, generated by a model using Instruction 1 and Instruction 2.

Your task is to act as a quality evaluator, analyzing the content of the Model Response to determine which instruction's all constraints is fully satisfied based on the following rules:

1. If all constraints in Instruction 1 are fully satisfied, output 1.

2. If all constraints in Instruction 2 are fully satisfied, output 2.

3. If neither instruction's constraints are fully satisfied, output -1.

Note that these two instructions contain conflicting constraints, making it impossible for the Model Response to fully satisfy both simultaneously.

**## Evaluation Steps**
Please analyze the Model Response, Instruction 1 and Instruction 2 carefully, adhering to the following steps:
Step 1: Analyze Instruction 1 and Instruction 2, and then extract relevant content from the Model Response. You should copy sentences from the Model Response exactly as they are, without any modification.

1. If the instructions include constraints related to keywords or phrases (e.g., keyword definitions, keyword frequency, or phrase frequency), extract the sentences containing the specified keywords or phrases. Record the positions of these keywords or phrases, if necessary.

2. If the instructions include constraints related to specific information or topics, extract segments containing the relevant information or topics.

3. If the instructions include constraints related to output formats or styles, extract segments that reflect the specified formats or styles.

Step 2: Analyze which instruction's constraints are fully satisfied.
Step 3: Directly give your evaluation answer (1, 2, or -1) without any additional content.

**## Output Format**
**Step 1**: The extracted content from the Model Response.
**Step 2**: A brief analysis of Instruction 1 and Instruction 2.
**Step 3**: The evaluation result (1, 2, or -1) without any additional content.

**## Evaluation Input**
**Model Response**
{*Model Response*}

**Instruction 1**
{*Instruction 1*}

**Instruction 2**
{*Instruction 2*}

**## Evaluation Output**

Table 11: Prompt template used to evaluate which of the two mutually conflicting instructions is satisfied by a model's output.