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

Xingwei He[1], Qianru Zhang[1*], Pengfei Chen[2*], Guanhua Chen[3], Linlin Yu[4], Yuan Yuan[5,6,7], Siu-Ming Yiu[1]

[1]Department of Computer Science, The University of Hong Kong, Hong Kong, China
[2]School of Artificial Intelligence, Xidian University, Xi'an, China
[3]Department of Statistics and Data Science, Southern University of Science and Technology, Shenzhen, China
[4]Department of Computer Science, Augusta University, Augusta, GA, USA
[5]School of Computer Science and Engineering, Beihang University, Beijing 100191, China
[6]Qingdao Research Institute, Beihang University
[7]Hangzhou Innovation Institute, Beihang University
hexingwei15@gmail.com, qrzhang@cs.hku.hk, chenpengfei@xidian.edu.cn

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 underexplored. 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) Most proprietary LLMs exhibit strong conflict detection capabilities, whereas among open-source models, only DeepSeek-R1 demonstrates similarly strong performance. DeepSeek-R1 and Claude-4.5-Sonnet achieve the highest average F1-scores at 91.5% and 87.3%, respectively, ranking first and second overall. (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.

Code — https://github.com/NLPCode/ConInstruct

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, 2025a). 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.

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. 2025b; 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

*Corresponding authors.

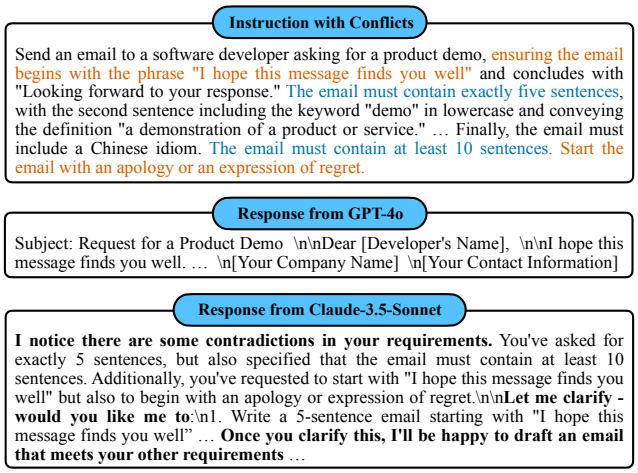


Figure 1: A conflicting instruction from ConInstruct with two responses from GPT-4o and Claude-3.5-Sonnet, where text in green and red indicate conflicts between phrase constraints and length constraints, respectively.

prior work has systematically evaluated LLMs' performance when faced with user instructions with conflicts.

To bridge this gap, we introduce **ConInstruct**, 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) (please refer to 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-4.5-Sonnet achieving the second-highest average F1-score at 87.3%. Notably, as the number of conflicts in an instruction increases, LLMs exhibit improved conflict detection ability, aligning with our intuitions.

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-4.5-Sonnet, explicitly alerts users to conflicts in only 45% of cases—either by (1) requesting further clarification (36%) or (2) resolving the conflicts autonomously and responding to the resolved instruction (9%). Moreover, as the number of conflicts in an instruction increases, strong LLMs (Claude models 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. In the following, we will 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. To be specific, ConInstruct comprises six common NLP 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*. We present the task and domain distribution for ConInstruct in Figure 7 of Appendix B. Overall, the seed instructions provide a diverse set of tasks and scenarios.

Constraint Types. Following previous work on instruction-following (Jiang et al. 2024; He et al. 2024a), we utilize six widely-used types of constraints to expand the seed instructions. **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 Appendix C.

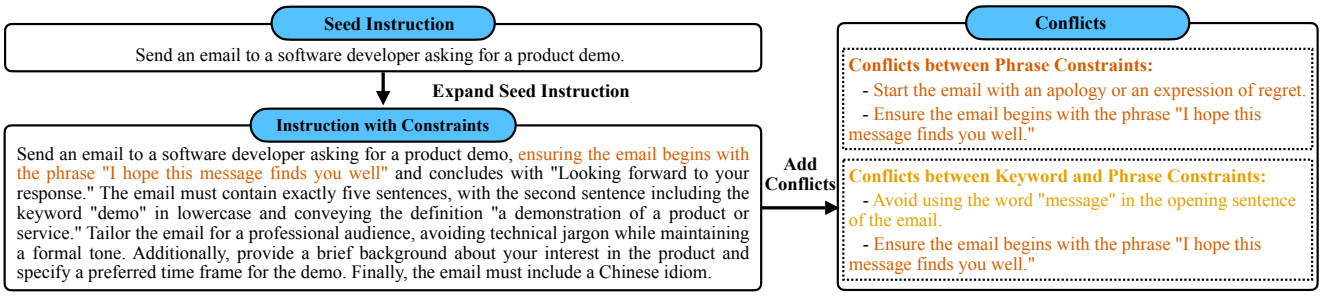


Figure 2: The construction process of the ConInstruct Benchmark: We start with a seed instruction, then add constraints to it. Finally, we introduce conflicts into the expanded instructions. Due to space limits, we show only two conflicts. In each conflict, the first constraint is newly added, while the second comes from the original instruction.

Instruction	Word	Sentence	CT	CFT
100	138.9	6.4	6	8.6

Table 1: Statistics of ConInstruct. 'Instruction' denotes the number of expanded instructions. 'Word', 'Sentence', 'CT', and 'CFT' represent the average number of words, sentences, constraint types, and conflict types per instruction.

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 4 of Appendix E.

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 are provided in Appendix D.

Adding Conflicts. We use GPT-4o to introduce conflicting constraints into the expanded instructions. 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 two conflict pairs corresponding to an expanded instruction. The prompt used to add conflicts is provided in Table 5 of Appendix E.

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[1] reviews the revised instructions and conflicts, removing unreasonable constraints or conflicts.

2.3 Dataset Statistics

Table 1 presents the basic statistics of the expanded instructions in ConInstruct. The dataset includes 94 and 70 conflict instances for the PC and KP conflict types, respectively, and 100 conflict instances for each of the remaining seven conflict types. 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.

3 Experiment Setup

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

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

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.

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 conflict 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) Seven Proprietary LLMs, including GPT-4o, GPT-4o-mini (OpenAI et al. 2023), Claude-4.5-Sonnet, Claude-3.5-Sonnet/Haiku (Anthropic 2024), Gemini-1.5-Pro/Flash-Latest (Reid et al. 2024). (2) Eleven Open-source LLMs, including DeepSeek-R1 (Guo et al. 2025), Meta-Llama-3.2-[1, 3]B-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. Experiments involving open-source LLMs were conducted using A100 GPUs with 40GB of memory, while proprietary LLMs were accessed through their official APIs.

4 Experiment Results on 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 6 of Appendix E.

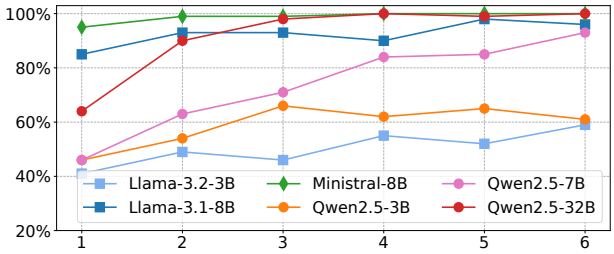


Figure 3: Conflict detection recall of LLMs for instructions in $\mathcal{I}_k$. The x-axis is the number of conflicts per instruction.

4.1 Instructions with a Single Conflict

Table 2 shows the conflict detection performance of various models. Our key findings are:
(1) **Proprietary LLMs exhibit superior performance in conflict detection, with the Claude family of models being particularly dominant.** This is evidenced by their notably high F1 scores—Claude-4.5-Sonnet (87.3%), Claude-3.5-Sonnet (86.6%), and Claude-3.5-Haiku (85.0%).
(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 faced by smaller open-source models.

4.2 Instructions with Multiple Conflicts

Figure 3 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, larger models generally exhibit improved detection performance.** This trend is particularly evident in Qwen2.5-[7, 32]B. However, smaller open-source models still 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 Experiment Results on Conflict Resolution

In the previous section, we demonstrated that LLMs, particularly proprietary ones, exhibit strong conflict detection

	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-4.5-Sonnet (2025-09-29)	88.5	88.5	88.5	86.0	86.5	88.5	88.5	86.8	83.6	87.7	86.3	87.3
	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 (2025-01)	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 (2025-01)	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	DeepSeek-R1 (2025-05-28)	93.1	**94.1**	**93.6**	**93.1**	88.1	**94.1**	**93.1**	89.1	**85.3**	**92.7**	**89.2**	**91.5**
	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 **bold** and underlined, respectively.

capabilities when faced with conflicting instructions. Building on this finding, this section further investigates how LLMs handle instructions with conflicting constraints, and whether they can respond safely—that is, by explicitly acknowledging conflicts in their responses and notifying users. Specifically, 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 users 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 3,600 responses from six LLMs (see Table 7 in Appendix E for the evaluation prompt). To check the quality of GPT-4o's assessment, we manually annotate behavior labels for 400 responses, achieving 98% agreement with GPT-4o's judgments. Besides GPT-4o, we employ Gemini-2.5-Pro as another judge. The Cohen's Kappa between both LLMs is 0.746, showing substantial agreement (> 0.6) (Landis and Koch 1977) (Please refer to Appendix F for more details). Figure 4 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, DeepSeek-R1, and Qwen2.5-32B predominantly exhibit Behavior 1: However, this does not imply that these LLMs lack the ability to detect conflicts. In Figure 3, 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 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-4.5-Sonnet and Claude-3.5-Haiku. However, despite the presence of multiple conflicts in instructions, Behavior 1 still constitutes a significant proportion of Claude models' responses.

These findings underscore the need to improve LLMs to adopt safe conflict resolution behaviors when faced with

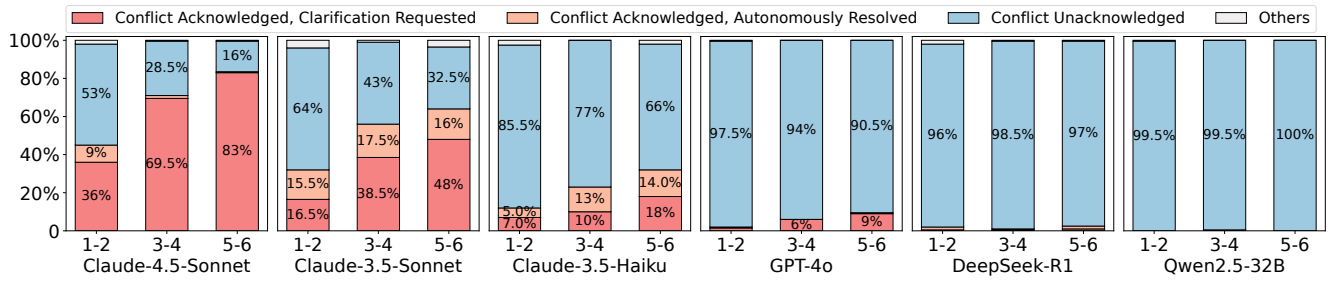


Figure 4: Distributions of conflict resolution behaviors exhibited by different LLMs when responding to instructions with varying numbers of conflicts. The x-axis denotes the number of conflicts per instruction.

Model	GP	$\mathcal{I}_1$				$\mathcal{I}_0$		F1
		B1↓	B2↑	B3	B4	B5↑	B6↓	
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–Sonnet	✗	78	7	11	4	100	0	-
	✔	**18**	**82**	0	0	72	28	78.1
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 **bold** and underlined indicate whether the behavioral changes meet or fail to meet expectations, respectively.

conflicts, which is essential to ensure reliable responses.

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 of Appendix E. 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.**

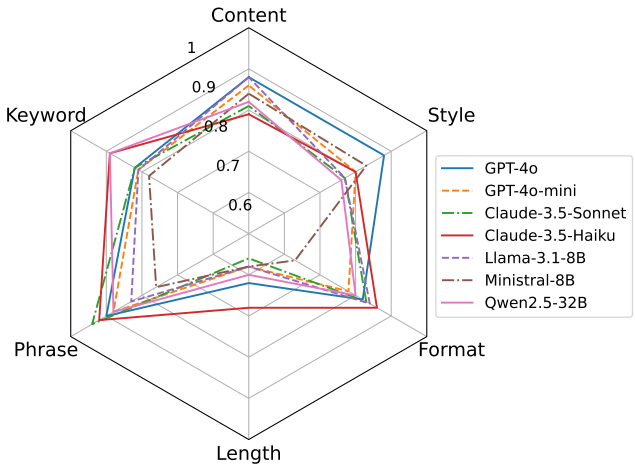


Figure 5: CSR of LLMs across different constraint types.

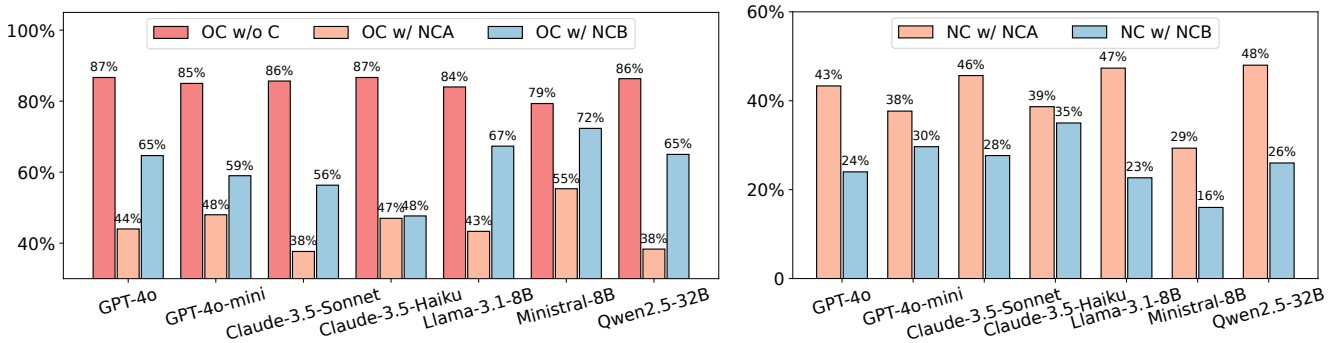
5.3 Analysis on Constraint Priority

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:

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

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 in all instructions.

We use GPT-4o to evaluate whether the model's output satisfies the specified constraints (Table 9 in Appendix E shows the evaluation prompt). To assess the evaluation quality of GPT-4o, we manually labeled 150 constraints and verified whether each constraint was satisfied in LLMs' responses. Table 11 in Appendix F shows that automatic evaluation aligns closely with human judgment. Figure 5 presents the CSR results of seven LLMs across different constraint types, revealing a clear performance pattern: the CSR score



(a) Constraint satisfaction rates of OC

(b) Constraint satisfaction rates of NC

Figure 6: 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.

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 4, 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** ($\mathcal{I}_{CC}, \mathcal{I}_{KK}, \mathcal{I}_{PP}$), where NC is introduced **after** OC, and **NCB subsets** ($\mathcal{I}'_{CC}, \mathcal{I}'_{KK}, \mathcal{I}'_{PP}$), where 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 (Table 10 in Appendix E shows 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 6(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 conflicting constraints degrade previous constraints. (2) 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**. In Figure 6(b), NC is more likely to be followed when it appears **later** (NCA) rather than earlier (NCB), further reinforcing the idea that **conflicting constraints appearing later in an instruction are more likely to be satisfied**.

6 Related Work

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, Grangier, and Auli 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 these studies, we investigate how LLMs detect and resolve conflicts when given instructions with conflicting constraints, thereby providing new insights into their instruction-following capabilities.

Conflict Detection has been extensively studied in natural language inference (Bowman et al. 2015; Williams, Nangia, and Bowman 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, Zhang, and Choi 2022; Neeman et al. 2023; Xie et al. 2024). Meanwhile, hallucination detection in LLMs (Manakul, Liusie, and Gales 2023; Min et al. 2023) investigates false 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.

Acknowledgments

This work is supported by HKU-SCF FinTech Academy, Shenzhen-Hong Kong-Macao Science and Technology Plan Project (Category C Project: SGDX20210823103537030), and Theme-based Research Scheme of RGC, Hong Kong (T35-710/20-R). Linlin Yu did not receive any financial support for this work and contributed only by developing the research ideas, participating in discussions, and providing feedback on the manuscript.

References

Anthropic, A. 2024. Claude 3.5 Sonnet. *Anthropic AI*.

Bowman, S. R.; Angeli, G.; Potts, C.; and Manning, C. D. 2015. A large annotated corpus for learning natural language inference. In Màrquez, L.; Callison-Burch, C.; and Su, J., eds., *EMNLP*, 632–642.

Chen, H.-T.; Zhang, M.; and Choi, E. 2022. Rich Knowledge Sources Bring Complex Knowledge Conflicts: Recalibrating Models to Reflect Conflicting Evidence. In Goldberg, Y.; Kozareva, Z.; and Zhang, Y., eds., *EMNLP*, 2292–2307.

Chen, Y.; Xu, B.; Wang, Q.; Liu, Y.; and Mao, Z. 2024. Benchmarking Large Language Models on Controllable Generation under Diversified Instructions. In *AAAI*, volume 38, 17808–17816.

Chowdhery, A.; Narang, S.; Devlin, J.; et al. 2023. PaLM: Scaling Language Modeling with Pathways. *Journal of Machine Learning Research*, 24(240): 1–113.

Dathathri, S.; Madotto, A.; Lan, J.; Hung, J.; Frank, E.; Molino, P.; Yosinski, J.; and Liu, R. 2020. Plug and Play Language Models: A Simple Approach to Controlled Text Generation. In *ICLR*.

Dubey, A.; Jauhri, A.; Pandey, A.; Kadian, A.; Al-Dahle, A.; Letman, A.; Mathur, A.; Schelten, A.; Yang, A.; Fan, A.; et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Fan, A.; Grangier, D.; and Auli, M. 2018. Controllable Abstractive Summarization. In Birch, A.; Finch, A.; Luong, T.; Neubig, G.; and Oda, Y., eds., *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, 45–54.

Geng, Y.; Li, H.; Mu, H.; Han, X.; Baldwin, T.; Abend, O.; Hovy, E.; and Frermann, L. 2025. Control illusion: The failure of instruction hierarchies in large language models. *arXiv preprint arXiv:2502.15851*.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

He, Q.; Zeng, J.; Huang, W.; Chen, L.; Xiao, J.; He, Q.; Zhou, X.; Liang, J.; and Xiao, Y. 2024a. Can Large Language Models Understand Real-World Complex Instructions? In *AAAI*, volume 38, 18188–18196.

He, X. 2021. Parallel Refinements for Lexically Constrained Text Generation with BART. In Moens, M.-F.; Huang, X.; Specia, L.; and Yih, S. W.-t., eds., *EMNLP*, 8653–8666.

He, X.; Gong, Y.; Jin, A.-L.; Qi, W.; Zhang, H.; Jiao, J.; Zhou, B.; Cheng, B.; Yiu, S.; and Duan, N. 2022. Metric-guided Distillation: Distilling Knowledge from the Metric to Ranker and Retriever for Generative Commonsense Reasoning. In Goldberg, Y.; Kozareva, Z.; and Zhang, Y., eds., *EMNLP*, 839–852.

He, X.; and Li, V. O. 2021. Show me how to revise: Improving lexically constrained sentence generation with xlnet. In *AAAI*, volume 35, 12989–12997.

He, X.; Lin, Z.; Gong, Y.; Jin, A.-L.; Zhang, H.; Lin, C.; Jiao, J.; Yiu, S. M.; Duan, N.; and Chen, W. 2024b. AnnoLLM: Making Large Language Models to Be Better Crowdsourced Annotators. In Yang, Y.; Davani, A.; Sil, A.; and Kumar, A., eds., *NAACL*, 165–190.

He, X.; and Yiu, S. M. 2022. Controllable Dictionary Example Generation: Generating Example Sentences for Specific Targeted Audiences. In *ACL*, 610–627.

Jia, F.; Wang, K.; Zheng, Y.; Cao, D.; and Liu, Y. 2024. GPT4MTS: Prompt-based Large Language Model for Multimodal Time-series Forecasting. In *AAAI*, volume 38, 23343–23351.

Jiang, Y.; Wang, Y.; Zeng, X.; Zhong, W.; Li, L.; Mi, F.; Shang, L.; Jiang, X.; Liu, Q.; and Wang, W. 2024. FollowBench: A Multi-level Fine-grained Constraints Following Benchmark for Large Language Models. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *ACL*, 4667–4688.

Jiayang, C.; Chan, C.; Zhuang, Q.; Qiu, L.; Zhang, T.; Liu, T.; Song, Y.; Zhang, Y.; Liu, P.; and Zhang, Z. 2024. ECON: On the Detection and Resolution of Evidence Conflicts. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *EMNLP*, 7816–7844.

Keskar, N. S.; McCann, B.; Varshney, L. R.; Xiong, C.; and Socher, R. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv preprint arXiv:1909.05858*.

Kikuchi, Y.; Neubig, G.; Sasano, R.; Takamura, H.; and Okumura, M. 2016. Controlling Output Length in Neural Encoder-Decoders. In Su, J.; Duh, K.; and Carreras, X., eds., *EMNLP*, 1328–1338.

Landis, J. R.; and Koch, G. G. 1977. The measurement of observer agreement for categorical data. *Biometrics*, 33(1): 159–174.

Manakul, P.; Liusie, A.; and Gales, M. 2023. SelfCheckGPT: Zero-Resource Black-Box Hallucination Detection for Generative Large Language Models. In Bouamor, H.; Pino, J.; and Bali, K., eds., *EMNLP*, 9004–9017.

Min, S.; Krishna, K.; Lyu, X.; Lewis, M.; Yih, W.-t.; Koh, P.; Iyyer, M.; Zettlemoyer, L.; and Hajishirzi, H. 2023. FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. In Bouamor, H.; Pino, J.; and Bali, K., eds., *EMNLP*, 12076–12100.

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

Neeman, E.; Aharoni, R.; Honovich, O.; Choshen, L.; Szpektor, I.; and Abend, O. 2023. DisentQA: Disentangling Parametric and Contextual Knowledge with Counterfactual

Question Answering. In Rogers, A.; Boyd-Graber, J.; and Okazaki, N., eds., *ACL*, 10056–10070.

OpenAI; Achiam, J.; Adler, S.; Agarwal, S.; et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; Schulman, J.; Hilton, J.; Kelton, F.; Miller, L.; Simens, M.; Askell, A.; Welinder, P.; Christiano, P. F.; Leike, J.; and Lowe, R. 2022. Training language models to follow instructions with human feedback. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *NIPS*, volume 35, 27730–27744.

Qin, Y.; Song, K.; Hu, Y.; Yao, W.; Cho, S.; Wang, X.; Wu, X.; Liu, F.; Liu, P.; and Yu, D. 2024. InFoBench: Evaluating Instruction Following Ability in Large Language Models. In Ku, L.-W.; Martins, A.; and Srikumar, V., eds., *Findings of ACL*, 13025–13048.

Reid, M.; Savinov, N.; Teplyashin, D.; et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *ArXiv*, abs/2403.05530.

Sun, J.; Tian, Y.; Zhou, W.; Xu, N.; Hu, Q.; Gupta, R.; Wieting, J.; Peng, N.; and Ma, X. 2023. Evaluating Large Language Models on Controlled Generation Tasks. In Bouamor, H.; Pino, J.; and Bali, K., eds., *EMNLP*, 3155–3168.

Thorne, J.; Vlachos, A.; Christodoulopoulos, C.; and Mittal, A. 2018. FEVER: a Large-scale Dataset for Fact Extraction and VERification. In Walker, M.; Ji, H.; and Stent, A., eds., *NAACL*, 809–819.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*.

Wallace, E.; Xiao, K.; Leike, R.; Weng, L.; Heidecke, J.; and Beutel, A. 2024. The instruction hierarchy: Training llms to prioritize privileged instructions. *arXiv preprint arXiv:2404.13208*.

Wang, X.; Wei, J.; Schuurmans, D.; Le, Q.; Chi, E.; and Zhou, D. 2022. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; ichter, b.; Xia, F.; Chi, E.; Le, Q. V.; and Zhou, D. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In Koyejo, S.; Mohamed, S.; Agarwal, A.; Belgrave, D.; Cho, K.; and Oh, A., eds., *NIPS*, volume 35, 24824–24837.

Wen, B.; Ke, P.; Gu, X.; Wu, L.; Huang, H.; Zhou, J.; Li, W.; Hu, B.; Gao, W.; Xu, J.; et al. 2024. Benchmarking complex instruction-following with multiple constraints composition. *arXiv preprint arXiv:2407.03978*.

Williams, A.; Nangia, N.; and Bowman, S. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In Walker, M.; Ji, H.; and Stent, A., eds., *NAACL*, 1112–1122.

Xie, J.; Zhang, K.; Chen, J.; Lou, R.; and Su, Y. 2024. Adaptive Chameleon or Stubborn Sloth: Revealing the Behavior of Large Language Models in Knowledge Conflicts. In *ICLR*.

Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; Lin, H.; Yang, J.; Tu, J.; Zhang, J.; Yang, J.; Yang, J.; Zhou, J.; Lin, J.; Dang, K.; Lu, K.; Bao, K.; Yang, K.; Yu, L.; Li, M.; Xue, M.; Zhang, P.; Zhu, Q.; Men, R.; Lin, R.; Li, T.; Xia, T.; Ren, X.; Ren, X.; Fan, Y.; Su, Y.; Zhang, Y.; Wan, Y.; Liu, Y.; Cui, Z.; Zhang, Z.; and Qiu, Z. 2024. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*.

Zhang, Q.; Wang, H.; Long, C.; Su, L.; He, X.; Chang, J.; Wu, T.; Yin, H.; Yiu, S. M.; Tian, Q.; and Jensen, C. S. 2024a. A Survey of Generative Techniques for Spatial-Temporal Data Mining. *arXiv preprint arXiv:2405.09592*.

Zhang, Q.; Wen, H.; Li, M.; Huang, D.; Yiu, S.-M.; Jensen, C. S.; and Liò, P. 2025a. AutoHFormer: Efficient Hierarchical Autoregressive Transformer for Time Series Prediction. *arXiv preprint arXiv:2506.16001*.

Zhang, T.; Shen, Y.; Luo, W.; Zhang, Y.; Liang, H.; Yang, F.; Lin, M.; Qiao, Y.; Chen, W.; Cui, B.; et al. 2024b. Cfbench: A comprehensive constraints-following benchmark for llms. *arXiv preprint arXiv:2408.01122*.

Zhang, Z.; Li, S.; Zhang, Z.; Liu, X.; Jiang, H.; Tang, X.; Gao, Y.; Li, Z.; Wang, H.; Tan, Z.; Li, Y.; Yin, Q.; Yin, B.; and Jiang, M. 2025b. IHEval: Evaluating Language Models on Following the Instruction Hierarchy. In Chiruzzo, L.; Ritter, A.; and Wang, L., eds., *NAACL*, 8374–8398. ISBN 979-8-89176-189-6.

Zheng, L.; Chiang, W.-L.; Sheng, Y.; Zhuang, S.; Wu, Z.; Zhuang, Y.; Lin, Z.; Li, Z.; Li, D.; Xing, E.; Zhang, H.; Gonzalez, J. E.; and Stoica, I. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *NIPS*, volume 36, 46595–46623.

Zhou, J.; Lu, T.; Mishra, S.; Brahma, S.; Basu, S.; Luan, Y.; Zhou, D.; and Hou, L. 2023a. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

Zhou, W.; Jiang, Y. E.; Wilcox, E.; Cotterell, R.; and Sachan, M. 2023b. Controlled Text Generation with Natural Language Instructions. In Krause, A.; Brunskill, E.; Cho, K.; Engelhardt, B.; Sabato, S.; and Scarlett, J., eds., *ICML*, volume 202 of *Proceedings of Machine Learning Research*, 42602–42613.