LEARNING SOFT CONSTRAINTS FROM REVERSED AND RECOVERABLE PREFERENCE PAIRS

Anonymous authors
Paper under double-blind review

ABSTRACT

Soft constraints shape how a response should be written, asking for a writing style, a situation to respect, or a scope of content. Unlike hard constraints, they cannot be verified by code, so existing pipelines build preference data for them by asking an LLM judge which response is better. Whether a response follows a soft constraint is a subjective call, so the judge's verdict carries its own biases, and a soft constraint does not always change the response enough to give the judge anything to compare. We present REVERB, whose principle is to plant, between the two responses of a pair, a difference that is easy to tell apart and to learn from. Reversing one soft constraint into its contrary gives two mirrored prompts, and each response is labeled as preferred under the prompt that produced it. Whether the two responses really separate along the reversed constraint is then verified: an LLM sees only the two responses and must recover the reversed constraint, a test with a known answer, and pairs that fail it are discarded. A single open-weight model runs the whole pipeline and produces 7,949 verified pairs. Direct preference optimization (DPO) on this data improves constraint following on two benchmarks by up to 16.4 points on two student models, outperforming every compared baseline, beating the judge-labeled pipeline by up to 12.0 points with fewer pairs, and preserving general abilities across five benchmarks.

1 INTRODUCTION

Large language models are increasingly used through natural-language instructions, and a real instruction rarely states a task alone. It attaches constraints, requirements on how the task should be answered: a summary must stay under 100 words, a cover letter must sound formal, an explanation must suit a reader with no prior knowledge. Formally, a constrained instruction is a pair $I = (x, C)$ of a task x and a set of constraints $C = \{c_1, \ldots, c_n\}$, and a response y is expected to satisfy every c_j in addition to completing x. Meeting these constraints is part of completing the task: a correct answer in the wrong format breaks the program that consumes it, and a correct answer in the wrong tone loses its reader. Constraints divide by how their satisfaction can be checked. A *hard* constraint admits a program that decides whether y satisfies it. Length, format, and keyword requirements are of this kind. A *soft* constraint admits no such program. Whether a response keeps a formal tone, suits a novice audience, or stays within a requested scope is a matter of degree, and it is settled by subjective judgment rather than by an objective test.

Post-training for constraint following follows this divide. For hard constraints, a program can check whether a response satisfies each constraint, so the training labels can be verified by code (Zhou et al., 2023; Dong et al., 2025). For soft constraints, no such program exists, so existing pipelines obtain the labels from an LLM judge (Ren et al., 2025; Sun et al., 2024): shown two responses to a constrained instruction, the judge picks the one that follows the constraint better, and its pick becomes the preference label. This recipe fails on two sides. On the judging side, the verdict can be wrong. Even when the two responses differ clearly, the judge can misread them. When it picks the better of two responses, position, verbosity, and self-preference biases distort the choice, so that swapping the order of the two candidates can by itself flip the verdict (Zheng et al., 2023; Wang et al., 2024). When it instead scores a single response along quality dimensions, the scores are unstable: the same response receives different ratings across repeated calls (Stureborg et al., 2024). And it mistakes fluency for compliance: when the response that violates the instruction reads better (e.g., with more polished detail), the judge often prefers it, with accuracy near or below

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

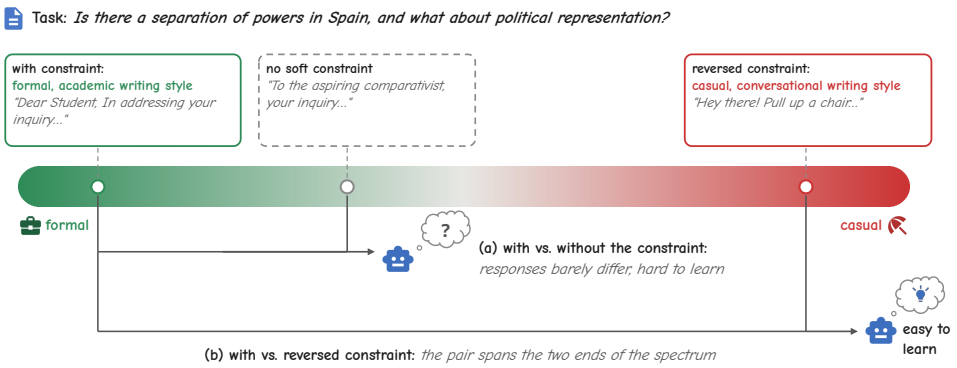


Figure 1: One task, answered three times by the same model: with the soft constraint, without it, and with the constraint reversed. (a) Existing pipelines pair the first two. Both are formal, so a judge has nothing to tell apart. (b) We pair the first and the third. They sit at opposite ends of the formal–casual axis, so the pair is easy to learn from. Positions on the axis are illustrative.

chance (Zeng et al., 2024). On the generation side, a soft constraint does not always change the response enough to be judged. Ask a model for a formal reply and its default reply is already formal: model outputs routinely satisfy stylistic and structural constraints that the instruction never states (Qi et al., 2025). And when two responses are sampled independently, the visible differences between them are often unrelated content variation rather than the constraint of interest (Cheng et al., 2024). Figure 1(a) shows a real case: with the constraint removed, the model still opens "To the aspiring comparativist, . . . ". The two responses sit at almost the same point of the formal–casual spectrum, and the judge is left to pick a winner between two equally formal texts. Whatever it picks, the pair teaches the model nothing about the constraint. The two sides fail independently: a better judge still learns nothing from a constraint that left no mark on the generation side, and a constraint that does leave a mark is still labeled by a biased judge. What the two share is that neither error can be detected, because the judging task has no verifiable answer. Soft constraints thus lack what hard constraints have, a training signal that can be checked. Narrowing this gap is the goal of this paper.

We present REVERB (REVErsal with Recoverable Back-translation). Its principle is to plant, between the two responses of a pair, a difference that is easy to tell apart and to learn from. Because we planted it, we know what the difference should be, and that knowledge does the judge's work twice: it fixes which response is preferred under which prompt, and it lets us check whether the planted difference actually shows in the two responses. The Reverse step plants it. One soft constraint of a prompt is flipped into its contrary (e.g., formal to casual), the original and the flipped prompt each generate a response, and each response is the preferred one under the prompt that produced it. This single construction answers both sides of the problem. On the generation side, the two responses are pushed toward opposite ends of the constraint, so the contrast that a removed constraint fails to produce is now built in. On the judging side, no verdict is needed, because the label follows from which prompt produced which response. The Recover step uses the planted answer to check the pair. Shown only the two responses, the LLM must name the constraint that separates them. We do not take its answer on trust. We compare it with the constraint we planted, and keep the pair only when the two match from both sides, so a reversal that changed nothing, or a response that ignored its constraint, is discarded. We instantiate the whole pipeline with one open-weight model, Qwen3.5-397B. DPO on the data constructed by our method improves the two student models by 11.3 points on FollowBench and 9.7 points on CFBench on average, measured on each benchmark's primary metric (defined in §4), and preserves the student models' general capabilities.

Our contributions are:

- We propose REVERB, a data-construction method that fixes preference labels by constraint reversal and checks data quality by back-translation recovery.
- We build a soft-constraint preference dataset of 7,949 pairs, constructed fully automatically by a single open-weight model.

- On two student models and two soft-constraint benchmarks, DPO on our data beats the judge-labeled pipeline by 8.7 points on FollowBench on average.

2 Related Work

Constraint-following post-training. Existing work splits by whether the constraint can be checked by a program. For hard constraints, IFEval (Zhou et al., 2023) defines verifiable instruction types, AutoIF (Dong et al., 2025) has the model write the checking code itself, and VerIF (Peng et al., 2025) carries the same recipe into reinforcement learning, so every label can be recomputed. For soft constraints no checker exists, and existing pipelines fall back on LLM opinion. Conifer (Sun et al., 2024) and UltraIF (An et al., 2025) compose multi-constraint instructions with strong LLMs and rank or filter the responses with LLM feedback. Suri (Pham et al., 2024) and Crab (Qi et al., 2025) back-translate constraints from existing texts, making the constraints seen in training denser and more diverse. IOPO (Zhang et al., 2025) builds input-output preference pairs over complex instructions, and Step-by-Step Mastery (Ren et al., 2025) removes constraints from an instruction one at a time and lets a judge pick the winner between the two resulting responses. SPaR (Cheng et al., 2024) reduces irrelevant variation between sampled responses by tree-search refinement, yet a judge still decides every label. RPO (Huang et al., 2025) rewrites the constraints that a judge finds violated in a sampled response into their opposites, so that the response becomes compliant. The label still rests on judge verdicts, and the method mainly targets hard constraints. In all of these pipelines, the label of a soft-constraint preference pair rests on an LLM judgment that cannot be checked. Our work instead strengthens the preference signal by active construction: for each soft constraint we generate one response under the original constraint and one under its reversal, so the two responses form a pair that contrasts exactly on that constraint and needs no judge to label, and we filter out the pairs that are invalid or whose contrast cannot be told apart.

LLM-as-judge reliability. A parallel line measures how unreliable that opinion is. LLM judges favor their own outputs, longer outputs, and whichever response appears first (Zheng et al., 2023). Swapping candidate order alone can flip a verdict (Wang et al., 2024), the same response receives different scores across repeated calls (Stureborg et al., 2024), and on pairs where the instruction-violating response is the better-written one, evaluator accuracy falls to chance (Zeng et al., 2024). Proposed remedies strengthen the judge rather than the task: averaging over swapped orders (Wang et al., 2024), multi-agent debate (Chan et al., 2024; Khan et al., 2024), panels of diverse judges (Verga et al., 2024), and evaluators fine-tuned for rubric-based grading (Kim et al., 2024). These remedies lower the error rate, but the verdict remains an opinion: for any single pair, nothing exists to grade it against, so the residual errors pass into the training data unnoticed. Our work takes a different route. The judge's task becomes recovering a constraint we planted, and its answer is graded against that known constraint.

3 Method: Reverse and Recover

REVERB builds preference data for soft constraints in five steps (Figure 2). A constrained prompt is a pair $I = (x, C)$ of a seed task x and soft constraints $C = \{c_1, \ldots, c_n\}$, $1 \le n \le 5$. Reversing the i-th constraint replaces c_i with its semantic opposite $\bar{c}_i$ and leaves everything else unchanged, giving the mirrored prompt $\bar{I}_i$. A single model M plays every role in the pipeline: it attaches the constraints, writes and checks each reversal, generates one response per side, $R \sim M(\cdot \mid I)$ and $\bar{R} \sim M(\cdot \mid \bar{I}_i)$, and answers the recovery check. Two facts hold throughout. The preference label of every exported pair is fixed by which prompt produced which response: under I the chosen response is R, and under $\bar{I}_i$ it is $\bar{R}$. And the one place where an LLM does judge, the recovery check, is graded against the planted pair $(c_i, \bar{c}_i)$.

3.1 Constrained prompt construction

The first step builds the pool of constrained prompts that every later step operates on: it takes plain tasks and attaches soft constraints to them. Every constrained prompt starts from a seed task, a plain instruction that states what the response should be about. A soft constraint only shapes how a task is answered, so it needs such a task to act on. We take our seeds from the 1,500 instruction tasks

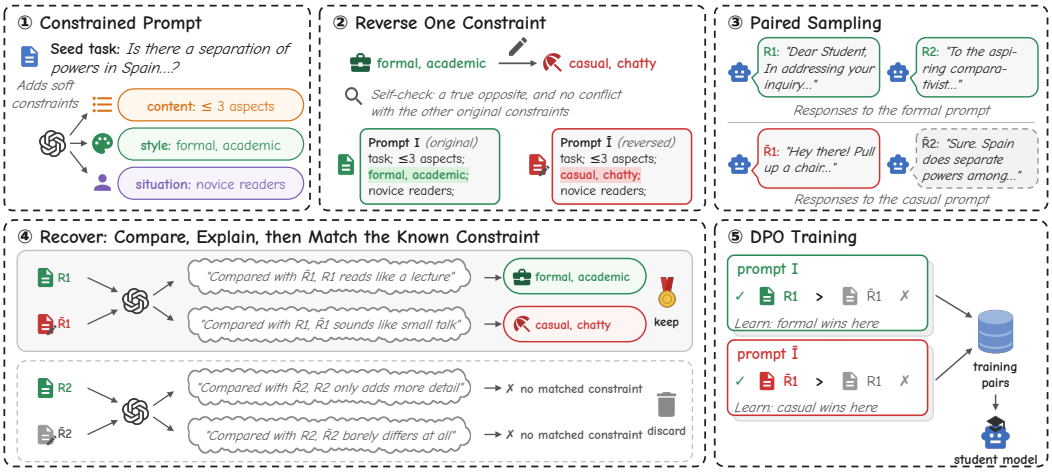


Figure 2: REVERB overview. Preference labels are fixed by construction at the reversal step. The LLM only gates pair quality by recovering the reversed constraint from the responses alone.

released by Ren et al. (2025), drawn from Open Assistant (Köpf et al., 2023), Self-Instruct (Wang et al., 2023), and Super-NaturalInstructions (Wang et al., 2022). They are a general-purpose mix of writing, analysis, coding, and factual questions, each paired with a reference answer. Following Ren et al. (2025), we drop tasks whose reference answers are shorter than 15 words, since such closed tasks (classification labels or single entities) leave a soft constraint nothing to shape. This leaves 967 seeds. To each seed, M attaches $n \in \{1, \ldots, 5\}$ soft constraints progressively, following Ren et al. (2025): at each step, M reads the instruction built so far and rewrites it to include one new constraint of a sampled category, phrased as a natural part of the instruction (prompts in Appendix A). We vary n because real prompts carry different numbers of soft constraints, and following them gets harder as the count grows. The categories come from FollowBench (Jiang et al., 2024), a benchmark that tests instruction following under constraints of several types and marks three of them as soft: a content constraint fixes what the response must cover or focus on (e.g., "focus on practical advice rather than theory"), a style constraint fixes tone, language complexity, or writing style (e.g., "use a formal, academic writing style"), and a situation constraint fixes the audience, persona, or scenario the response is written for (e.g., "write for novice readers"). Three independent constraint combinations per seed yield 2,898 constrained prompts, whose 8,664 attached constraints (about 3 per prompt on average) are the candidates for reversal.

3.2 REVERSE: FLIPPING ONE SOFT CONSTRAINT

The Reverse step turns one prompt into a pair of prompts that ask for the same task in two opposite ways: it picks one soft constraint and replaces it with its opposite, keeping everything else fixed. Formally, given a constrained prompt $I = (x, C)$ with $C = \{c_1, \ldots, c_n\}$, reversing the i-th constraint produces the mirrored prompt $\bar{I}_i = (x,\ C \setminus \{c_i\} \cup \{\bar{c}_i\})$, where $\bar{c}_i$ is a semantic opposite of c_i written by M. The seed task and the other $n - 1$ constraints are copied verbatim, so I and $\bar{I}_i$ differ in exactly one requirement, and a prompt with n constraints yields up to n mirrored pairs.

What counts as a semantic opposite decides whether the reversal produces a learnable contrast, so we fix it with three requirements. First, $\bar{c}_i$ must be a genuine opposite of c_i, not a paraphrase and not a bare negation. In linguistic terms, we require a contrary of c_i, the other end of the same dimension, rather than a contradictory, which merely negates it (Cruse, 1986; Horn, 1989). Take "use a formal, academic writing style". A paraphrase restates the same requirement in other words, such as "use scholarly language", so the two prompts still ask for the same thing. A bare negation is the contradictory: it forbids the original requirement without saying what should hold instead, such as "do not write formally", so generation has no target to follow and the sampled responses drift apart in no particular direction. A genuine opposite is the contrary: it states the other end of the dimension that c_i fixes as a positive requirement, here "use a casual, conversational writing style". Second, $\bar{c}_i$ must stay compatible with the rest of the prompt. Flipping "keep an optimistic tone" into "keep a

Algorithm 1 Paired sampling with a verifiable gate, for one valid flip.

Require: mirrored prompts $(I, \bar{I}_i)$, planted constraints $(c_i, \bar{c}_i)$, budget K
for $k = 1, \ldots, K$ **do**
  sample $R^k \sim M(\cdot \mid I)$ and $\bar{R}^k \sim M(\cdot \mid \bar{I}_i)$
  $P_k \leftarrow \{(R^a, \bar{R}^b) : \max(a, b) = k\}$, drop length-incomparable pairs, sort by comparability
  for $(R, \bar{R}) \in P_k$ **do**
    if $\mathrm{recover}(R, \bar{R}) = c_i$ **and** $\mathrm{recover}(\bar{R}, R) = \bar{c}_i$ **then**
      return the accepted pair $(R, \bar{R})$
    end if
  end for
end for
return no pair, abandon the flip

bleak, pessimistic tone" is a valid contrary, but it is rejected when another constraint of the same prompt asks the response to end with encouraging advice. Third, $\bar{c}_i$ must be phrased like c_i, with similar wording and length, so that the requirement it carries is the only thing the reversal changes. Flipping the formal constraint into "adopt a relaxed voice, keep sentences short, and feel free to joke" does point to the casual end, but it is longer and more specific than the original, and the responses could then differ because of that extra detail rather than because of the reversed meaning. The last two requirements are checked at this step. In the same call that writes $\bar{c}_i$, M flags flips that conflict with the remaining constraints, and a string-level check verifies that the untouched constraints still appear verbatim in $\bar{I}_i$, so the flip changed one requirement and nothing else. Together these drop 14% of the candidates, leaving 7,470 of 8,664. The first requirement is different. At this step, the definition above can only guide M in writing the flip. Whether the flip is a genuine opposite in effect shows only in the responses it later produces, so we defer its verification to the subsequent steps.

Reversing one constraint at a time keeps the contrast attributable. The two responses of a mirrored pair are pushed apart along the single dimension of c_i, so the pair teaches a model about exactly that constraint. Had we flipped several constraints at once, the responses would differ in several ways and the preference signal would spread over all of them. Editing one factor per example is what ties an observed difference to its cause, the same principle behind counterfactually augmented data and contrast sets (Kaushik et al., 2019; Gardner et al., 2020). A mirrored pair also fixes the training labels without any judging. Under I, the chosen response is R, the one generated to follow c_i, and the rejected one is $\bar{R}$. Under $\bar{I}_i$, the two swap roles. This assignment is correct only if R is actually written in the way c_i asks and $\bar{R}$ in the way $\bar{c}_i$ asks. Generation can miss a constraint, so the condition does not always hold. The reversal step thus leaves exactly two things unverified, and both are visible only in the responses: whether $\bar{c}_i$ opposes c_i in effect, and whether each response follows its own constraint. One downstream test covers both. The next two steps sample candidate pairs and keep one only when its two responses differ in precisely the way $(c_i, \bar{c}_i)$ prescribes.

3.3 RECOVER: A BACK-TRANSLATION CHECK WITH A KNOWN ANSWER

This step does the verification that the earlier steps deferred, and it is the core design of REVERB. Two things remain unverified after the reversal: (1) *whether the flip is a genuine opposite in effect*, and (2) *whether each response follows its own constraint*. Asking an LLM directly whether a response satisfies a soft constraint cannot settle either question, because that question has no checkable answer, and wrong verdicts would enter the training data invisibly, which is the failure this paper sets out to avoid. Our idea is to make the judgment gradable instead. The constraint difference between the two mirrored prompts was planted by us, so we can ask M to recover it from the two responses alone and grade its answer against what we planted. A pair is kept only when this recovery succeeds. Failed candidates are discarded, and fresh responses are sampled in their place. The third and fourth stages of Figure 2, paired sampling and recover, form a single loop, and we describe them together.

The sampling side is plain. For each valid flip, M generates responses for the two prompts, one response per side in each round, and every original-side and reversed-side response can form a candidate pair $(R, \bar{R})$. The check then decides whether a candidate is kept by asking M to recover the planted constraints from the responses alone. It runs in two calls. In the first, M sees only the two responses, is told that their prompts differ in exactly one soft constraint of the three kinds of

§3.1, and states in one sentence the constraint that the first response follows and the second does not. The prompts and the constraints stay hidden, so nothing in the input hints at the expected answer. In the second call, M matches that sentence against a shuffled list of candidates holding the target constraint, the untouched constraints of the same prompt, two constraints drawn from other prompts, and a none-of-the-above option. The probability that the matcher puts on its choice is kept as a confidence score. Splitting description from matching keeps M blind and its answer checkable: it cannot be steered toward the planted constraint, and whatever it writes is graded against candidates we control. The check runs in both directions, where the order states which response M reads as Response A. Shown $(R, \bar{R})$, so that R comes first, M must recover c_i, the constraint the first response follows. With the order swapped to $(\bar{R}, R)$, it must recover $\bar{c}_i$. A pair is accepted only when both directions succeed, and the second direction runs only after the first one hits.

The sampling loop feeds this check on demand (Algorithm 1). In round k, M generates one fresh response on each side. Every pairing of an original-side and a reversed-side response that involves a new sample becomes a candidate. Candidates whose lengths differ by a factor of more than 1.5 are dropped before judging, since a large length gap draws the comparison toward length itself and feeds the known length bias of preference training (Park et al., 2024). The remaining candidates are checked in order of length comparability, and the first accepted pair settles the flip. A flip with no accepted pair after $K{=}5$ rounds is abandoned. About half of the flips settle in the first round, and the loop costs 5.3 generations per flip on average, roughly half of the $2K{=}10$ worst case.

Key property. This one check covers both questions left open by the Reverse step (§3.2). If the flip is not a genuine opposite, or a response ignores its constraint, there is no planted difference to recover, and the pair is discarded. An accepted pair's two responses separate along the planted constraint, in the same direction as the labels. That is what preference learning needs, because a preference label makes a relative claim only: it says one response is better than the other, and between two responses that separate along c_i, the one closer to c_i is indeed the better answer.

3.4 Pair export and DPO fine-tuning

The last step turns accepted pairs into training examples, each a prompt with a chosen and a rejected response. Every accepted flip yields one example under its reversed prompt: given $\bar{I}_i$, the chosen response is $\bar{R}$ and the rejected one is R. For the original prompt we export a single example even when several of its flips are accepted, so that no prompt appears in the data twice. Among the accepted flips of I we pick the one whose recovery had the highest matcher confidence, breaking ties toward the closest response lengths, and export its R as chosen and $\bar{R}$ as rejected. A source prompt with n accepted flips thus contributes $n{+}1$ examples. Our run yields 5,373 accepted flips across 2,576 of the 2,898 source prompts, 7,949 examples in total.

This export is deliberately symmetric. Each original-prompt example reuses the two responses of one reversed-prompt example with the labels swapped, so the same two responses appear twice, ranked in opposite ways. Which response wins is then decided by the prompt alone. A model trained on such twins cannot explain the data through properties of the responses themselves, such as length or polish, and is pushed to read the constraint in the prompt, the behavior that Figure 1 calls for. In effect this closes the shortcut, akin to reward hacking in preference training, of winning by response-surface features such as length (Park et al., 2024).

Tracking the data through the pipeline, the self-check of §3.2 keeps 7,470 valid flips out of the 8,664 candidate constraints, and the back-translation check of §3.3 accepts 5,373 of them, so 28% of the valid flips are discarded before export. Every pair that reaches the training data has therefore passed a check with a known answer.

We train on the exported pairs with direct preference optimization (DPO; Rafailov et al., 2023). DPO takes a prompt x with a chosen response y_w and a rejected response y_l and raises the likelihood margin between them relative to a frozen reference model π_{ref}, minimizing

$$\mathcal{L}_{\mathrm{DPO}} = -\,\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\mathrm{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\mathrm{ref}}(y_l \mid x)}\right)\right], \tag{1}$$

where σ is the logistic function and β controls how far the trained model π_θ may move from the reference. DPO suits our data well: it learns directly from binary pairs with fixed labels, exactly

what the pipeline produces, and its contrastive objective exploits the mirrored structure of our twins, rewarding the same response on one side of the pair and penalizing it on the other (details in §4).

4 EXPERIMENTAL SETUP

Pipeline implementation. Every role of the pipeline, attaching constraints, writing and checking the reversals, generating responses, and running the recovery check, is played by Qwen3.5-397B-A17B (Qwen Team, 2026), an open-weight mixture-of-experts model served locally with vLLM (Kwon et al., 2023) in FP8 precision. The whole dataset is produced without any closed-source API.

Students and training. We train two student models, Qwen3.5-9B and Llama-3.1-8B-Instruct (Grattafiori et al., 2024). The first shares its family with the data-generating model while the second does not, which tests whether the data transfers across model families. Both student models are trained with LoRA (Hu et al., 2021) and the DPO objective of §3.4 for 3 epochs at learning rate 5×10^{-6} with β=0.1, on one NVIDIA H200 GPU (about 15 hours per run for Qwen3.5-9B and 6 for Llama-3.1-8B). Every dataset in our experiments is trained with this identical recipe.

Baselines. We compare four settings on each student model. *Base* is the untrained student model. *SFT* fine-tunes the student model on the chosen responses of our pairs, which tests whether the gains come from the preference contrast or from simply imitating verified responses. *SbSM* reproduces the judge-labeled paradigm of Ren et al. (2025) on the same constrained prompts with the same generator. For each prompt, one constraint is removed, the generator answers both the full and the weakened prompt, and an LLM judge reads the two responses and declares the one that serves the full instruction better the winner. The winner becomes the chosen response no matter which prompt produced it, so the response to the weakened prompt can be preferred over the response to the full one. REVERB trains on the 7,949 pairs of §3.4.

Benchmarks and judging. We evaluate constraint following on two benchmarks. FollowBench (Jiang et al., 2024) organizes instructions into five difficulty levels, where a level-k instruction carries k added constraints. We use its LLM-judged subset of 470 prompts (FB-all) together with the 285 prompts whose constraints are of the content, situation, and style kinds (FB-soft). We report two rates. Under the hard satisfaction rate (HSR), an instruction counts as satisfied only when every one of its constraints is followed. The soft satisfaction rate (SSR) instead grades constraints one by one and reports the fraction of satisfied constraints. Both are averaged over the five levels. CFBench (Zhang et al., 2024) contains 1,000 instructions whose constraints are annotated as primary or secondary needs, and also probes how the data transfers across languages, since our training pairs are entirely in English. It reports three rates. The constraint satisfaction rate (CSR) is the fraction of satisfied constraints, averaged over instructions. The instruction satisfaction rate (ISR) is the fraction of instructions whose constraints are all satisfied. The priority satisfaction rate (PSR) counts an instruction as satisfied when its primary needs are all met and its secondary needs are mostly met, following the benchmark's official scoring rule. Both benchmarks score responses with an LLM judge, for which we use Claude Sonnet 4.6 with the official judging prompts. This is the native evaluation protocol of both benchmarks. To reduce the influence of a single judge model's bias, Appendix C additionally rescores every system with a different judge. To further validate the judge, two annotators, each with over three years of experience using AI systems and familiar with prompt writing and evaluation, re-judged a blind random sample of 50 judged responses covering all compared systems (141 constraint-level verdicts) and agreed with the judge on 92.2% of them, with disagreements resolved by discussion.

5 RESULTS AND ANALYSIS

5.1 MAIN RESULTS

REVERB delivers the largest gains of all methods on both student models. As shown in Table 1, REVERB improves every metric of both benchmarks on both student models, and its margin over the base model is the largest of the three trained variants on every single metric. On Qwen3.5-9B, FB-soft HSR rises from 82.5 to 88.8 and PSR from 56.0 to 58.9. On Llama-3.1-8B, FB-soft HSR rises from 65.6 to 81.8 and PSR from 20.8 to 37.2. Although our pairs are built around soft constraints,

Table 1: Main results on constraint following (%).

		FB-all		FB-soft		CFBench		
Model	Method	HSR	SSR	HSR	SSR	CSR	ISR	PSR
Qwen3.5-9B	Base	73.8	90.2	82.5	93.6	78.8	43.6	56.0
	SFT	76.4	90.8	87.0	95.7	80.1	45.4	57.9
	SbSM	75.5	90.1	83.5	93.6	79.3	45.3	57.6
	REVERB	**82.1**	**93.0**	**88.8**	**96.0**	**80.4**	**46.2**	**58.9**
Llama-3.1-8B	Base	57.2	81.3	65.6	84.6	53.2	13.1	20.8
	SFT	65.7	83.6	75.1	88.1	59.8	19.5	28.2
	SbSM	62.6	84.1	69.8	87.4	57.8	16.4	26.0
	REVERB	**72.3**	**88.9**	**81.8**	**92.6**	**66.3**	**26.1**	**37.2**

the gains carry over to FB-all, whose instructions also contain hard constraints: REVERB adds 8.3 HSR points there on Qwen and 15.1 on Llama, again the largest margins in the table.

REVERB improves a strong base model, and improves a weak one far more. The two student models start from very different places. Qwen3.5-9B already follows constraints well, its base scores are high on both benchmarks, and headroom is accordingly small, yet REVERB still buys 6.3 points of FB-soft HSR and 2.9 of PSR on top of it. Llama-3.1-8B starts clearly below Qwen on every metric, and for such a model the sharp contrast in our mirrored pairs pays off most. Its FB-soft HSR gains 16.2 points and its PSR 16.4, roughly twice what the next best method obtains, and far ahead of the other three variants on all metrics.

Why REVERB outperforms SbSM and SFT. The two baselines improve the base by similar and modest amounts on most metrics. SbSM builds a pair by removing one soft constraint from the prompt, answering both the full and the weakened prompt, and letting a judge pick the response that serves the full instruction better. Such pairs do teach some soft-constraint following. But the judge is free to prefer the response that was generated without the constraint, and it tends to do so exactly when the constraint shifts the response too little to tell the two apart, so a share of its pairs carries a vague or reversed lesson. SFT trains on the chosen halves of our pairs as separate examples: the response to the original prompt and the response to the reversed prompt each appear under their own prompt. Every one of these responses has passed the recovery check, so SFT imitates verified compliant behavior under the original constraints and their opposites alike, and this verified, constraint-dense supervision is enough to match the judge-labeled pairs. What neither baseline provides is a trustworthy contrast, the difference between a response that follows a constraint and one that follows its opposite, under the same prompt. REVERB trains on both sides of that verified contrast, and that is where its consistent extra margin comes from.

5.2 A CLOSER LOOK AT THE PREFERENCE DATA

More than a third of SbSM's preference labels prefer the response that ignores the constraint. The SbSM reproduction lets a judge pick the winner between a response to the full prompt and a response to the prompt with one soft constraint removed. Of its 8,520 pairs, 3,339 (39%) chose the response generated without the constraint, and for another 128 (1.5%) the judge declared no preference between the two, in which case the protocol falls back to the full-prompt response as the default winner. A weak-side win is not necessarily an error, since the weakened prompt can produce the genuinely better response. But it often points at a problem one step earlier. If adding a constraint does not make the response that honors it preferable, the constraint itself was likely written too vaguely to steer the response, and the judge-labeled pipeline has no stage that assesses constraint quality or catches such cases, so these unmeasurable judgment calls flow straight into 39% of the labels. The same observation explains why REVERB checks the flipped constraint so many times. The reversal is screened at construction time in §3.2, and a pair enters training only after the recovery check of §3.3 confirms that the constraint difference is visible in the responses, so a vague or ineffective constraint is discarded instead of silently turning into a label.

The recovery check rejects the pairs that carry no signal, at a stable rate. The check accepts 5,373 of the 7,470 valid flips (72%). The acceptance rate is flat in the number of constraints on

Table 2: General-capability results (%, MT-Bench scored from 1 to 10).

Model	Method	MMLU	GSM8K	HumanEval	AlpacaEval	MT-Bench
Qwen3.5-9B	Base	78.6	**87.6**	**71.3**	77.1	7.75
	REVERB	**78.8**	87.1	68.3	**78.0**	**7.96**
Llama-3.1-8B	Base	**68.3**	70.9	62.2	29.2	**6.08**
	REVERB	67.8	**71.2**	**62.8**	**43.4**	6.02

the prompt, 71.2% for single-constraint prompts and 72.2% for five-constraint prompts. Across constraint kinds the spread is wider. Content flips pass at 76.5% and style flips at 70.6%, while situation flips pass at 67.4%, matching the intuition that a changed situation often leaves the surface of the response nearly unchanged. Of the roughly 2,100 rejected flips, 868 never produced a length-comparable candidate pair within the sampling budget, and the rest were judged but never recovered, meaning their two responses did not separate along the planted constraint.

A human audit of the accepted pairs. The same annotators audited 50 randomly sampled accepted pairs. In all 50, the response generated under the original prompt was judged closer to the original constraint than its counterpart, so the label direction held in every audited pair, and in 49 of the 50 the reversed constraint was judged a genuine opposite of the original.

5.3 GENERALIZATION

Preference training that targets one narrow skill can tax the rest of the model, so we check the general abilities of the trained student models on five benchmarks. MMLU (Hendrycks et al., 2020), GSM8K (Cobbe et al., 2021), and HumanEval (Chen et al., 2021) measure knowledge, mathematics, and coding, and are scored by rules through lm-eval-harness (Gao et al., 2024) with no judge involved. AlpacaEval 2.0 (Dubois et al., 2024) and MT-Bench measure general instruction following. We report the AlpacaEval win rate against the official GPT-4-turbo responses and the MT-Bench score on a 1 to 10 scale, both judged by the same Claude Sonnet 4.6 as in §4.

The constraint gains come at no general cost. As shown in Table 2, the trained student model scores above its base on six of the ten model and metric combinations. The largest gain is the AlpacaEval win rate of Llama-3.1-8B, up 14.2 points. Our pairs train the model to read the prompt for what it actually asks, and Llama-3.1-8B, starting from a win rate of only 29.2, has the most room to benefit. A few metrics remain on par with the base model, with differences within about half a point. The one visible decrease is HumanEval on Qwen3.5-9B, which gives up 3.0 points. With only 164 problems, each worth 0.6 points, and with Llama-3.1-8B improving on the same benchmark, the decrease reads as fluctuation rather than a systematic cost. Overall, training on our pairs sharpens constraint following while general instruction following improves and knowledge, mathematics, and coding stay at the level of the base model.

6 CONCLUSION

Soft constraints shape how a response should be written, but whether a response truly follows is hard to verify. For this reason, unlike for hard constraints, there is no established practice for writing them well, so they are often vaguely defined, and whether one takes effect is unknown until tried. This paper presented REVERB, which flips exactly one soft constraint into its contrary, so that the two responses of a pair separate along that constraint and the preference labels follow from construction rather than from a judge. A back-translation check with a known answer then verifies each pair before it enters training, and a single open-weight model runs the whole pipeline. The resulting pairs substantially improve soft-constraint following on two student models, beat every compared baseline, including a judge-labeled pipeline built from the same prompts and generator with more pairs, and leave general abilities intact. We see in REVERB a recipe that reaches beyond soft constraints. When the effect of an instruction is too subtle to stand out in ordinary data, reverse the instruction and build the pair across its two extremes: there the effect stands out, and the model can learn it directly. Checking such pairs still relies on a model, but it answers a question with a known answer instead of giving an opinion, so its judgment can be graded.

AI USE STATEMENT

In accordance with the ICLR policy, we used generative AI tools for two tasks with required disclosure. First, this paper studies LLM-generated preference data, so an LLM was used to generate a synthetic dataset: the pipeline of §3 attaches constraints, writes and checks the reversals, generates responses, and runs the recovery check, exactly as described in the main text. Second, generative AI tools assisted in implementing parts of the experiment code. We have not used generative AI tools for developing the conceptual framework, proposing or refining hypotheses, designing the methodology or experiments, or interpreting results, and the remaining tasks with required disclosure are not applicable to this work. Additionally, we used generative AI tools for tasks with recommended disclosure, namely editing the paper for grammar, fluency, and clarity, and polishing LaTeX table and figure code. We have reviewed all AI-assisted work. AI-assisted code was reviewed and tested by the authors, the generated dataset is audited by the verifiable checks described in the paper and by the authors, and all AI-edited text was revised by the authors. We take full responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We provide an anonymized repository containing the full data pipeline together with the training and evaluation scripts that reproduce all reported results: `https://anonymous.4open.science/r/REVERB-FBEF`. Every prompt used by the pipeline is included verbatim in Appendix A. The exact models and versions used in each experiment, the training hyperparameters, and the evaluation protocol are specified in §4. The pipeline depends on no proprietary service, since a single open-weight model served locally plays every role, so both the data and the experiments can be reproduced with open components alone.

REFERENCES

Kaikai An, Li Sheng, Ganqu Cui, Shuzheng Si, Ning Ding, Yu Cheng, and Baobao Chang. Ultraif: Advancing instruction following from the wild. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18722–18737, 2025.

Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. In *International conference on learning representations*, volume 2024, pp. 9079–9093, 2024.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Jiale Cheng, Xiao Liu, Cunxiang Wang, Xiaotao Gu, Yida Lu, Dan Zhang, Yuxiao Dong, Jie Tang, Hongning Wang, and Minlie Huang. Spar: Self-play with tree-search refinement to improve instruction-following in large language models. *arXiv preprint arXiv:2412.11605*, 2024.

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

David A Cruse. *Lexical semantics*. Cambridge university press, 1986.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. Self-play with execution feedback: Improving instruction-following capabilities of large language models. In *International Conference on Learning Representations*, volume 2025, pp. 39286–39313, 2025.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang

Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. The language model evaluation harness, 07 2024. URL https://zenodo.org/records/12608602.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, et al. Evaluating models' local decision boundaries via contrast sets. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1307–1323, 2020.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Laurence R Horn. A natural history of negation. 1989.

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

Xiang Huang, Ting-En Lin, Feiteng Fang, Yuchuan Wu, Hangyu Li, Yuzhong Qu, Fei Huang, and Yongbin Li. Reverse preference optimization for complex instruction following. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 20700–20723, 2025.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4667–4688, 2024.

Divyansh Kaushik, Eduard Hovy, and Zachary C Lipton. Learning the difference that makes a difference with counterfactually-augmented data. *arXiv preprint arXiv:1909.12434*, 2019.

Akbir Khan, John Hughes, Dan Valentine, Laura Ruis, Kshitij Sachan, Ansh Radhakrishnan, Edward Grefenstette, Samuel R Bowman, Tim Rocktäschel, and Ethan Perez. Debating with more persuasive llms leads to more truthful answers. *arXiv preprint arXiv:2402.06782*, 2024.

Seungone Kim, Jay Shin, Joel Jang, Shayne Longpre, Hwaran Lee, Sangdoo Yun, Ryan Shin, Sungdong Kim, James Thorne, Minjoon Seo, et al. Prometheus: Inducing fine-grained evaluation capability in language models. In *International Conference on Learning Representations*, volume 2024, pp. 29927–29962, 2024.

Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in neural information processing systems*, 36:47669–47681, 2023.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. Disentangling length from quality in direct preference optimization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 4998–5017, 2024.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. Verif: Verification engineering for reinforcement learning in instruction following. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 30312–30327, 2025.

Chau Minh Pham, Simeng Sun, and Mohit Iyyer. Suri: Multi-constraint instruction following in long-form text generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 1722–1753, 2024.

Yunjia Qi, Hao Peng, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. Constraint back-translation improves complex instruction following of large language models. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 2388–2398, 2025.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Qingyu Ren, Jie Zeng, Qianyu He, Jiaqing Liang, Yanghua Xiao, Weikang Zhou, Zeye Sun, and Fei Yu. Step-by-step mastery: Enhancing soft constraint following ability of large language models. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 19581–19596, 2025.

Rickard Stureborg, Dimitris Alikaniotis, and Yoshi Suhara. Large language models are inconsistent and biased evaluators. *arXiv preprint arXiv:2405.01724*, 2024.

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

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. *arXiv preprint arXiv:2404.18796*, 2024.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, et al. Large language models are not fair evaluators. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 9440–9450, 2024.

Yizhong Wang, Swaroop Mishra, Pegah Alipoormolabashi, Yeganeh Kordi, Amirreza Mirzaei, Atharva Naik, Arjun Ashok, Arut Selvan Dhanasekaran, Anjana Arunkumar, David Stap, et al. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 5085–5109, 2022.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pp. 13484–13508, 2023.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. Evaluating large language models at evaluating instruction following. In *International Conference on Learning Representations*, volume 2024, pp. 40193–40219, 2024.

Tao Zhang et al. Cfbench: A comprehensive constraints-following benchmark for llms. *arXiv preprint arXiv:2408.01122*, 2024.

Xinghua Zhang, Haiyang Yu, Cheng Fu, Fei Huang, and Yongbin Li. Iopo: Empowering llms with complex instruction following via input-output preference optimization. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22185–22200, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

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

A PIPELINE PROMPTS

This appendix lists, verbatim, every prompt template used by the REVERB pipeline and by the SbSM reproduction. Placeholders in curly braces are filled in by the pipeline at run time.

A.1 CONSTRAINT ATTACHMENT

Following Ren et al. (2025), one soft constraint is added per call with the shared rewriting template below, and the {requirement} slot selects the constraint category. The five category-specific requirement texts follow the template.

Constraint attachment: shared rewriting template

```
You are an Instruction Rewriting Expert. You need to rewrite #Given Instruction# based on
     #Rewriting Requirement#, in order to obtain a #Rewritten Instruction#.
Basically, #Rewritten Instruction# should adhere to the following guidelines:
1. Your rewriting cannot omit the non-text parts such as the table and code in #Given
     Instruction#.
2. #Rewritten Instruction# must be reasonable and must be understood and responded to by
     humans.
3. You should try your best not to make the #Rewritten Instruction# become verbose.
     #Rewritten Instruction# can only add 10 to 20 words into #Given Instruction#.
#Given Instruction#
{instruction}
#Rewriting Requirement#
{requirement}
Please output in JSON format with the fields 'modified_instruction' for the modified
     instruction and 'added_constraint' for the added constraint. Output the JSON object
     only.
```

Constraint attachment: the five category requirements

```
[Content_open] Please add one proper content constraint to the #Given Instruction#. The
     content constraints include but are not limited to:
1. Add a Subtask or Another Related Question.
2. Narrow Down the Topic: Instead of a general theme or topic, provide a more specific
     subset.
3. Set a Higher Standard: Raise the bar for what's considered acceptable or successful.
4. Limit Resources: Restrict the number or type of resources someone can use.
5. Introduce Specific Criteria: Mandate particular components or features that must be
     included.
6. Specifying Sequence: Dictate the order in which certain steps or actions should be
     taken.

[Content_language] Please add one proper content constraint to the #Given Instruction#.
     The content constraints include but are not limited to:
1. Specify Language Complexity: Determine whether the text should use simple,
     intermediate, or advanced language.
2. Restrict Vocabulary: Include or exclude specific words or phrases, or limit the range
     of vocabulary.

[Situation_suggest] Please add one proper situation constraint to the #Given
     Instruction#: describe a specific situation, audience, or circumstance under which
     the response should be produced (e.g. advising a particular kind of person, reacting
     to a specific scenario, or addressing a certain context).

[Situation_role] Please add one proper role-play constraint to the #Given Instruction#:
     require the response to be written from the perspective of a specific role,
     profession, or persona.

[Style] Please add one proper style constraint to the #Given Instruction#. The style
     constraints include but are not limited to:
1. Tone: formal, informal, empathetic, critical, humorous, serious, etc.
2. Emotional coloring: positive, optimistic, cautious, skeptical, sad, etc.
3. Writing style: concise, detailed, academic, conversational, persuasive, etc.
```

A.2 REVERSAL

The reversal template of §3.2. The model rewrites exactly one target constraint into its contrary and reports its own validity check in the same call.

Reversal

```
You are an Instruction Rewriting Expert. The #Given Instruction# below contains several
     constraints. Your task is to rewrite EXACTLY ONE target constraint into a constraint
     with the OPPOSITE meaning, and leave every other part of the instruction untouched.

Requirements for the opposite constraint:
1. It must be a genuine semantic opposite of the target constraint - not a paraphrase,
     not a vague negation like "do not X" (prefer stating what SHOULD hold instead), and
     not a synonym.
2. It must remain compatible with the rest of the instruction: it must not contradict any
     other constraint and must keep the base task achievable.
3. Keep the wording style and length similar to the target constraint.

#Given Instruction#
{instruction}

#Target Constraint#
{constraint}

Please output in JSON format with the fields:
- 'flipped_constraint': the opposite-meaning constraint text
- 'modified_instruction': the full instruction with ONLY the target constraint replaced
     by the flipped constraint
- 'is_valid_opposite': true/false - your own check that the flip is a genuine opposite
     and conflict-free
- 'note': one short remark if is_valid_opposite is false, else ""
Output the JSON object only.
```

A.3 RECOVERY CHECK: DESCRIBING THE DIFFERENCE

The first call of the recovery check of §3.3. The model sees only the two responses and states the differing constraint in one sentence. The check runs once per direction, with the two responses swapped for the backward direction.

Recovery check: difference description

```
Below are two responses, A and B, produced for two instructions that differ in exactly
     one constraint. You are NOT shown the instructions.

The differing constraint is a SOFT constraint of one of these kinds:
- content: involves restrictions on the scope or depth of the response (what must be
     covered, focused on, or included);
- style: limits the manner of expression (tone, emotional coloring, language complexity,
     writing style);
- situation: refers to background limitations (the audience, persona, or scenario the
     response is tailored to).

Compare the two responses. In AT MOST ONE sentence, state the constraint that Response A
     follows which differs from Response B (describe A relative to B).

Response A:
{response_a}

Response B:
{response_b}

Your one-sentence answer:
```

A.4 RECOVERY CHECK: MATCHING THE DESCRIPTION

The second call of the recovery check. The description is matched against a shuffled list holding the target constraint, the untouched constraints of the same prompt, two constraints from other prompts,

and a "None of the above" option. The probability the model places on its chosen letter is kept as the confidence score.

Recovery check: candidate matching

```
A difference between two responses (A vs B) was described as:

"{description}"

Which of the following candidate constraints does this description correspond to? The
     description states what held in Response A as opposed to Response B; a candidate
     matches if Response A following THAT constraint (while Response B does not) would
     explain the described difference.

{options}

Answer with the single letter of the best-matching option, or the letter of "None of the
     above" if none matches. Answer with the letter only.
```

A.5 SbSM REPRODUCTION

The judge-labeled baseline of §4 removes one constraint with the template below and then lets a judge pick the winner. The judge template is ported near verbatim from the official SbSM repository and evaluates against the whole instruction. The verdict [[A]] selects the weakened-prompt response, [[B]] the full-prompt response, and ties [[C]] default to the full-prompt side.

SbSM reproduction: constraint removal

```
You are an Instruction Rewriting Expert. The #Given Instruction# below contains several
     constraints. Your task is to rewrite the instruction with EXACTLY ONE target
     constraint removed, leaving every other part untouched.

Requirements:
1. Remove only the target constraint; all other constraints and the base task must remain
     verbatim as far as grammar allows.
2. The rewritten instruction must stay fluent and natural.

#Given Instruction#
{instruction}

#Target Constraint to remove#
{constraint}

Please output in JSON format with the field:
- 'modified_instruction': the full instruction with ONLY the target constraint removed
Output the JSON object only.
```

SbSM reproduction: judge

```
You are a helpful assistant who reviews a debate between two other assistants in
     evaluating the quality of the outputs for a given instruction.The two assistants,
     Assistant (a) and Assistant (b), are given an instruction. Output (a) and Output (b)
     are generated by two different AI chatbots respectively. Assistant (a) and Assistant
     (b) have conflicting evaluations. Your goal is to review their evaluations and give
     your final decision on which output is better.

Here are some rules of the evaluation:
1. You should prioritize evaluating whether the output honestly/precisely/closely
     executes the instruction, then consider its helpfulness, accuracy, level of detail,
     harmlessness, etc.
2. Outputs should NOT contain more/less than what the instruction asks for, as such
     outputs do NOT precisely execute the instruction.
3. You should avoid any potential bias and your judgment should be as objective as
     possible. For example, the order in which the outputs were presented should NOT
     affect your judgment, as Output (a) and Output (b) are **equally likely** to be the
     better.

Output your final verdict by strictly following this format: "[[A]]" if Output (a) is
     better, "[[B]]" if Output (b) is better, and "[[C]]" for a tie.
```

```
[Given instruction]
{instruction}

[The Start of Output (a)]
{output_a}
[The End of Output (a)]

[The Start of Output (b)]
{output_b}
[The End of Output (b)]
```

B TEACHER-SOURCE ROBUSTNESS: CLAUDE-GENERATED DATA

To study how the choice of the teacher model affects the constraint-following ability that the student models learn from the data, we ran the identical pipeline with a different teacher family. Claude Sonnet 4.6 attaches the constraints, writes and checks the reversals, and generates the responses, while the recovery check is judged by Claude Haiku 4.5, a far smaller and cheaper model. A cheap judge is viable precisely because of the design of §3.3: the judge only answers a question with a known answer, so a weaker judge at most lowers the yield. From the same 967 seeds this track produces 6,392 pairs, on which both student models are trained with the recipe of §4.

Table 3: DPO with Claude-generated data against DPO with the self-generated data of the main text (%). The pipeline, student models, and evaluation are identical, only the data-generating models change.

		FB-all		FB-soft		CFBench		
Model	Method	HSR	SSR	HSR	SSR	CSR	ISR	PSR
Qwen3.5-9B	Base	73.8	90.2	82.5	93.6	78.8	43.6	56.0
	REVERB (Qwen3.5-397B data)	**82.1**	**93.0**	88.8	96.0	80.4	46.2	58.9
	REVERB (Claude data)	80.6	92.9	**89.8**	**96.4**	**80.8**	**47.1**	**59.8**
Llama-3.1-8B	Base	57.2	81.3	65.6	84.6	53.2	13.1	20.8
	REVERB (Qwen3.5-397B data)	72.3	88.9	81.8	92.6	**66.3**	**26.1**	**37.2**
	REVERB (Claude data)	**72.6**	**89.0**	**83.2**	**93.1**	64.9	24.1	34.5

Table 3 places the two REVERB datasets side by side. The models trained on Claude data land within about one point of the self-generated track on most metrics, and the differences run in both directions: on Qwen3.5-9B the Claude track is slightly higher on FB-soft (89.8 against 88.8) but lower on FB-all (80.6 against 82.1), and on Llama-3.1-8B it is higher on FB-soft (83.2 against 81.8) but lower on CFBench PSR (34.5 against 37.2). The gains over the base models are of nearly identical size on both tracks, even though the Claude track trains on fewer pairs (6,392 against 7,949) and grades its recovery check with a far cheaper judge. REVERB is therefore not tied to a particular data-generating model, and the recovery check does not require the generator to grade its own outputs.

C JUDGE ROBUSTNESS: A SECOND EVALUATOR

The benchmark scores in §5 come from one LLM judge. To check that the conclusions do not depend on this choice, we rescore every system of Table 1 with Qwen3.5-397B-A17B as the judge, using the same official judging prompts and the same responses. Table 4 shows the result. The absolute numbers move, as this judge is more lenient on FollowBench and stricter on CFBench, but the picture of §5.1 repeats. REVERB stays the best method on every metric for both student models. On Qwen3.5-9B it lifts FB-soft HSR from 94.0 to 97.8 and FB-all HSR from 86.1 to 91.3 over the base model, and leads SbSM by 4.0 and 3.2 points on the two. On Llama-3.1-8B the gains are again the largest of the table, 14.9 points of FB-soft HSR and 13.9 of FB-all HSR over the base model, with PSR nearly doubling from 14.5 to 27.0 and a lead of 8.5 PSR points over SbSM. The per-model orderings of the four methods match those under the original judge.

Table 4: Main results rescored with Qwen3.5-397B-A17B as the judge (%). Rows and protocol are identical to Table 1, only the judge changes.

		FB-all		FB-soft		CFBench		
Model	Method	HSR	SSR	HSR	SSR	CSR	ISR	PSR
Qwen3.5-9B	Base	86.1	95.0	94.0	98.0	75.4	41.7	53.3
	SFT	88.6	95.5	94.2	98.4	75.2	41.8	53.1
	SbSM	88.1	95.4	93.8	97.9	76.0	43.5	54.6
	REVERB	**91.3**	**96.3**	**97.8**	**99.0**	**76.7**	**45.1**	**55.1**
Llama-3.1-8B	Base	72.9	87.3	76.4	89.5	44.9	9.0	14.5
	SFT	80.4	90.3	86.2	93.1	52.3	15.6	22.7
	SbSM	77.9	90.7	83.8	94.2	48.8	12.2	18.5
	REVERB	**86.8**	**93.5**	**91.3**	**96.7**	**56.6**	**20.0**	**27.0**

D QUALITATIVE EXAMPLES

We illustrate the recovery check with one accepted and one rejected flip from our data.

Accepted flip: the reversal separates the responses

Task: summarize a group conversation about restaurant tips for a London visit.

Target constraint: Write the summary in a formal tone.

Reversed constraint: Write the summary in a casual tone.

Response R (original side): *"Ingmar announced his upcoming visit to London and requested recommendations for restaurants and local attractions favored by residents. In response, Kevin suggested Dino's pizza place located at Bottom Alley 50, while Miranda noted that although Soho is an excellent area, it is prohibitively expensive. [...]"*

Response $\bar{R}$ (reversed side): *"Ingmar's hitting up London this week and asked the crew for some local spots to eat and hang out. Kevin kept it low-key since he's not going out much lately, but he still threw out a recommendation for Dino's pizza place over on Bottom Alley 50. Miranda vouched for Soho, though she warned that it's seriously expensive. [...]"*

Judge description (sees only the two responses): *"Response A adheres to a formal style constraint with neutral, professional language, whereas Response B employs a casual, colloquial tone with slang and informal phrasing."*

Matcher: selects the target constraint in both directions (confidence 1.0). **Pair accepted.**

Rejected flip: the reversal leaves no trace

Target constraint: You are acting as a quiz designer for a history of art course.

Reversed constraint: You are acting as a quiz designer for a history of architecture course.

Response R (original side): *"Did the Art Deco movement originate as a direct stylistic reaction to the organic forms of Art Nouveau within the British Arts and Crafts tradition prior to the outbreak of the First World War?"*

Response $\bar{R}$ (reversed side): *"Did the Art Deco movement, characterized by its emergence in 1920s France, originate as a direct stylistic continuation of the organic forms and asymmetrical lines that defined the preceding Art Nouveau period?"*

Judge description (sees only the two responses): *"Response A adheres to a content constraint requiring the inclusion of the British Arts and Crafts tradition and a pre-WWI timeframe, whereas Response B focuses on 1920s France and describes the relationship as a continuation rather than a reaction."*

Matcher: selects "None of the above". **Flip abandoned.**

The two cases show the check doing exactly what §3.3 promises. In the accepted flip, the formal and the casual constraint pull the two summaries to the two ends of one style dimension, the blind judge describes exactly that difference, and the matcher maps the description onto the planted constraint in both directions. In the rejected flip, the reversal is linguistically genuine, an art course and an architecture course are indeed different situations, but Art Deco sits in both fields, so a quiz question

looks the same under either persona. Accordingly, the judge does not tell the art course and the architecture course apart at all. What it reports instead is the incidental difference it does see, one question anchored in the British Arts and Crafts tradition before the First World War and the other in 1920s France, and the matcher correctly finds that no planted constraint corresponds to such a description. A judge-labeled pipeline would still have scored such a pair and turned an accidental difference into a preference label. The recovery check instead discards it, which is the failure mode of Figure 1(a) caught in the wild.