Gradient-Guided Multi-Judge Prompt Optimization

Anonymous ACL submission

Abstract

Automatic prompt optimization is a practical alternative to fine-tuning for adapting large language models (LLMs), yet existing approaches often trade off signal quality against computational cost. Methods that rely on generative feedback can be informative but expensive to scale, while sampling-based optimization typically requires many evaluations and exhibits high variance. Even loss-driven prompt optimization remains limited by costly segment attribution that scales with prompt length and by overfitting to a single evaluator, which weakens transfer across model families and domains. We propose Gradient-guided Multi-judge Prompt Optimization (GMPO), a scalable framework that improves both efficiency and robustness. GMPO uses a first-order gradient approximation to score segment importance in a continuous masking direction, requiring only one forward and one backward pass. GMPO further employs a generate multi-judge design in which candidate prompt edits are proposed by a generator and selected using cross-entropy losses aggregated from multiple lightweight judge models, reducing evaluator bias and improving generalization. Experiments across math, reasoning, instruction-following evaluation, and safety robustness benchmarks demonstrate consistent gains with substantially lower optimization overhead. We will publicly release our code.

1 Introduction

Large language models (LLMs) have demonstrated strong zero-shot and few-shot generalization across reasoning, coding, and instruction following tasks, yet their behavior remains highly sensitive to prompt phrasing and structure. Small changes in instructions can lead to large variance in accuracy, faithfulness, and safety(Cao et al., 2024; Sclar et al., 2023). Consequently, prompting has evolved from ad-hoc trial-and-error into a central interface for controlling LLM capabilities. A series of prompting heuristics, including Chain-of-Thought, Rephrase-and-Respond, and Step-Back prompting, show that explicitly encouraging intermediate reasoning or reformulation can substantially improve performance on challenging benchmarks (Wei et al., 2022; Deng et al., 2024a; Zheng et al., 2023). However, these improvements are typically handcrafted, task-specific, and difficult to transfer across models, motivating automated prompt optimization (APO) methods that can systematically search for effective prompts under limited budgets (Ramnath et al., 2025a).

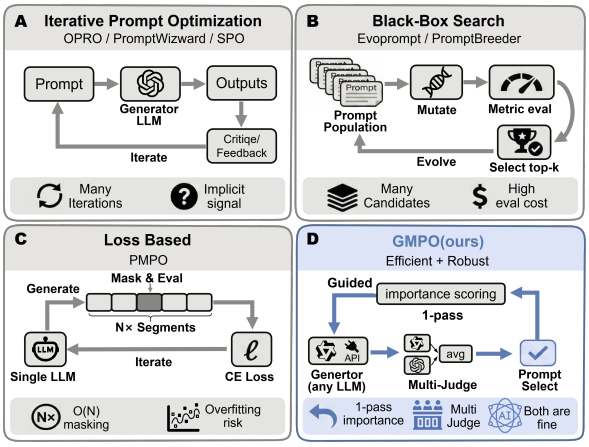

Figure 1: Comparison of APO paradigms and our framework: single-pass attribution and multi-judge loss improve efficiency and robustness, enabling black-box generators.

In terms of methodology, existing APO approaches broadly fall into three paradigms. First, Model-Guided Iterative Prompt Optimization methods use a generator model to iteratively propose prompt revisions and refine them using feedback such as self-critique, meta-optimization, or pairwise comparisons (Yang et al., 2023; Agarwal et al., 2025; Xiang et al., 2025). While flexible,

their optimization signals are often implicit and may require many iterative generations and evaluations, leading to non-trivial cost and instability. Second, black-box search methods such as evolutionary algorithms treat prompts as discrete candidates and optimize task metrics over populations (Tong et al., 2025). These methods avoid subjective self-judgment but typically require evaluating large numbers of variants, which increases compute and variance, especially on long prompts. Third, gradient-inspired and loss-based methods attempt to introduce more direct optimization signals. Textual gradients can guide revisions without true backpropagation (Yuksekgonul et al., 2024), and probabilistic metric prompt optimization uses token-level cross-entropy as an explicit objective, improving efficiency and objectivity (Zhao et al., 2025). Nevertheless, despite offering more explicit optimization signals, current loss-based frameworks often remain computationally demanding in practice, can be brittle to design choices and evaluator assumptions, and are not always straightforward to combine with strong closed-source generators.

To address these issues, we propose GMPO, an efficient and robust framework that combines first-order gradient attribution with multi-judge loss ensembling. Our key idea is to obtain a first-order (local) estimate of segment importance via a Taylor expansion of the loss. Concretely, we compute the directional derivative of the loss when moving a segment's embeddings toward a mask embedding. This yields an importance score for all segments using a single forward-backward pass on judge models. These importance scores provide actionable rewrite guidance, enabling the generator LLM to focus on the most influential segments and propose targeted revisions. Candidate prompts are then evaluated by an ensemble of open-source judge models via explicit cross-entropy losses, reducing evaluator-specific overfitting and improving cross-model robustness. This generator–evaluator decoupling enables us to exploit the creativity and linguistic coverage of black-box generators while retaining transparent, low-variance, loss-based evaluation on accessible judges.

We validate the proposed framework across diverse prompting regimes and domains, spanning mathematical reasoning (GSM8K(Cobbe et al., 2021), AQUA-RAT(Ling et al., 2017)), broad reasoning (Big-Bench Hard(Suzgun et al., 2022)), open-ended instruction following (AlpacaEval 2.0(Dubois et al., 2025)), and safety-critical adversarial evaluation (AdvBench(Chen et al., 2022)). Empirically, our method delivers consistent gains while improving attribution efficiency and reducing reliance on any single evaluator. Our contributions are summarized as follows:

- We introduce a first-order, gradient-based importance scoring method that provides full prompt-segment attribution with a single backward pass, eliminating segment-wise masking overhead.
- We propose a multi-judge loss-ensembling optimization strategy that mitigates single-model overfitting and improves cross-model prompt transferability.
- We develop a hybrid generator and evaluator framework that can incorporate powerful black-box LLMs as generators without requiring their internal likelihoods or gradients.
- We provide extensive evaluations across reasoning, instruction following, and safety-oriented benchmarks, demonstrating improved effectiveness, efficiency, and robustness.

2 Related Work

2.1 Iterative Prompt Optimization

Iterative prompt optimization methods use a generator model to iteratively propose prompt revisions and refine them using feedback such as self-critique, meta-optimization, or pairwise comparisons (Yang et al., 2023; Agarwal et al., 2025; Xiang et al., 2025). This paradigm is appealing for its flexibility: it operates directly in natural language, can incorporate task descriptions and examples, and can leverage strong closed-source models as generators without requiring gradients or token probabilities.

A key limitation is that the optimization signal is often mediated through iterative generation and evaluation loops(Ramnath et al., 2025b; Fathullah and Gales, 2025). As a result, efficiency depends heavily on design choices such as how many candidates are generated per round, how feedback is elicited, and how many refinement rounds are run, which can lead to noticeable cost and sometimes unstable progress across iterations. In addition, when improvement is measured primarily through a single feedback channel, the resulting prompt may be less reliable when transferred to different models or evaluation setups(Chatterjee et al., 2024; Wang et al., 2025).

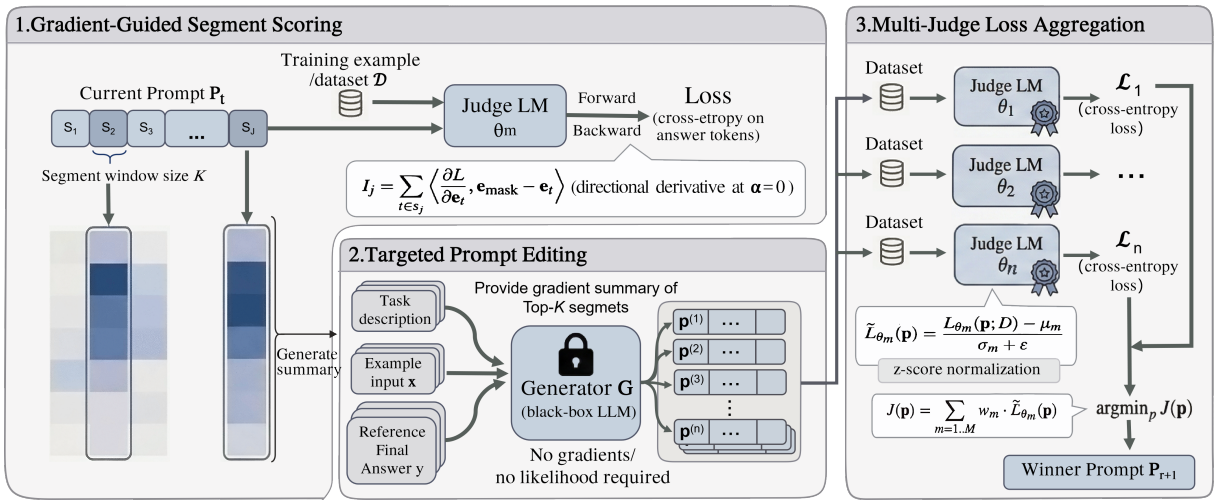

Figure 2: Overview of GMPO. (1) Gradient-guided segment scoring measures each prompt segment's influence using a first-order loss approximation (one forward and one backward pass). (2) Targeted prompt editing summarizes the Top-K segments and prompts a generator model to produce candidate revisions. (3) Multi-Judge Loss Aggregation scores candidates with an ensemble of judges and chooses the prompt with the lowest aggregated loss.

2.2 Black-Box Search Methods

Black-box search methods treat prompts as discrete candidates and optimize task-level metrics over a population of prompt variants, commonly using evolutionary algorithms (Tong et al., 2025; Fernando et al., 2023). Because they only require input–output access, which are broadly applicable to both open- and closed-source models and avoid reliance on free-form self-critique as the primary training signal(Deng et al., 2022; Wu et al., 2024).

However, population-based search typically requires evaluating many prompt variants to make steady progress, which increases computational budget and can yield noisy selection when each candidate is tested on limited data.(Schneider et al., 2024) This variance is especially pronounced when tasks are expensive to evaluate or when prompt changes interact with decoding randomness, making it harder to reliably distinguish genuinely better prompts from fluctuations.

2.3 Gradient-Inspired Methods

Gradient-inspired and loss-based methods use model likelihoods or related objectives to provide more direct optimization signals (Shin et al., 2020; Das et al., 2024). Textual-gradient approaches frame prompt improvement as iteratively applying natural-language gradient feedback produced by LLM(Yuksekgonul et al., 2024). PMPO(Zhao et al., 2025) instead uses token-level cross-entropy loss to evaluate prompts and employs masking-based analysis over a small number of removable semantic units to identify influential parts for revision. Such explicit objectives can reduce subjectivity in selection relative to purely critique-based feedback.

Nonetheless, the optimization outcome can be sensitive to the choice of evaluator model used to compute the objective. When a single evaluator dominates selection, improvements observed under that scorer may not consistently carry over to other models or evaluation conditions, motivating more robust evaluation designs.

3 GMPO Framework

We introduce GMPO, an iterative prompt optimization framework that alternates between segment attribution, targeted rewriting, and robust selection. As illustrated in Figure 2, GMPO first uses a single forward-backward pass on judge models to estimate the importance of each prompt segment. It then rewrites the most influential segments by querying a generator model. Finally, it evaluates all candidates with multiple judge models using explicit cross-entropy losses and updates the prompt to the best candidate under aggregated objective.

Problem Formulation Given a task with input x and target output y, we aim to find an optimized prompt $p^\star$ that minimizes the training loss on a training set under a limited optimization budget:

$$p^\star = \arg\min_{p \in \mathcal{P}} \mathcal{L}(p; \mathcal{D}), \tag{1}$$

where $\mathcal{P}$ denotes the set of candidate prompts, and $\mathcal{D}$ is a training set used for prompt optimization.

3.1 Gradient-Guided Segment Scoring

The core problem is to quantify how much each segment contributes to the current loss. In our implementation, we segment the prompt into contiguous fixed-length token windows (with window size K) and treat each window as a segment s_j for attribution and editing. GMPO computes importance scores for all segments via a first-order gradient approximation, requiring only one forward pass to evaluate the loss and one backward pass to obtain token-level gradients. This yields segment scores in a single evaluation step, avoiding repeated model runs and keeping computationally efficient.

Let the token embeddings of segment s_j be $\{e_t\}_{t\in s_j}$, and let e_{mask} denote a fixed mask embedding. We define a continuous interpolation path that replaces segment embeddings with the mask embedding:

$$e_t(\alpha) \;=\; (1-\alpha)e_t + \alpha e_{\text{mask}}, \alpha \in [0,1]. \quad (2)$$

When $\alpha = 0$, the segment is unchanged; when $\alpha = 1$, all tokens in the segment are replaced by the same mask embedding. Let $\mathcal{L}_{\theta_m}(\alpha)$ denote the loss of model θ_m when segment s_j is interpolated with parameter α (all other tokens remain fixed). The exact loss change induced by masking s_j is

$$\Delta\mathcal{L}_j \;=\; \mathcal{L}_{\theta_m}(1) - \mathcal{L}_{\theta_m}(0) \;=\; \int_0^1 \frac{d\mathcal{L}_{\theta_m}}{d\alpha}(\alpha)\, d\alpha. \quad (3)$$

We approximate this integral using the directional derivative at the starting point $\alpha = 0$:

$$I_j \;\triangleq\; \left.\frac{d\mathcal{L}_{\theta_m}}{d\alpha}\right|_{\alpha=0} \;=\; \sum_{t\in s_j} \left\langle \frac{\partial \mathcal{L}_{\theta_m}}{\partial e_t},\; e_{\text{mask}} - e_t \right\rangle. \quad (4)$$

We use I_j as the importance score for segment s_j. Intuitively, I_j measures the instantaneous rate at which the loss increases when moving the segment toward being masked. Larger positive values indicate that the segment is more critical for the current objective.We obtain the gradients $\frac{\partial \mathcal{L}}{\partial e_t}$ by differentiating the loss with respect to the prompt token embeddings under model θ_m. The segment-level importance scores are then computed by aggregating token-wise contributions within each segment according to Eq. (4), producing the attribution vector $(I_1, \ldots, I_J)$.Implementation details are provided in the appendix.

3.2 Targeted Prompt Editing

Given segment-level importance estimates, GMPO performs prompt revision through a generator that proposes improved instruction prompts. The editing module is intentionally decoupled from the evaluator models: it can be instantiated with either an open-source generator or a closed-source black-box model, since this stage does not require access to token-level likelihoods or gradients.

At each optimization step, the generator is queried with (i) the current instruction prompt, (ii) a task description that specifies the intended capability scope, and (iii) a representative supervised example consisting of an input and its reference final answer. When available, we additionally provide a textual summary of gradient-based important segments to guide the generator toward preserving content that is likely to be performance-critical while improving weaker regions. In some settings, we also supply an example rejected answer together with its associated configuration as auxiliary context, encouraging the generator to avoid undesirable behaviors.

We sample multiple candidate updates from the generator conditioned on the same editing context, and forward these candidates to the multi-judge evaluation module for selection. By concentrating revisions on high-impact prompt components while keeping evaluation objective and loss-based, this targeted editing strategy allocates computation where it is most effective, improves iteration-to-iteration stability, and enables the use of strong external generators without sacrificing transparent selection through open-source judges.

3.2.1 Multi-Judge Loss Aggregation

A central challenge in loss-based prompt optimization is evaluator overfitting: a prompt may reduce the loss of a single judge model by exploiting its calibration or stylistic preferences, yet fail to generalize to other models. To promote cross-model transfer, we evaluate each candidate prompt with an ensemble of M judge models $\{\theta_m\}_{m=1}^{M}$ and aggregate their token-level negative log-likelihood losses.

For a prompt p, we compute the development-set loss for each judge. Because different judges may exhibit different loss scales due to tokenization, calibration, or architecture, we apply per-judge normalization before aggregation. Given a candidate pool $\mathcal{C}$ at an optimization step, we compute

$$\widetilde{\mathcal{L}}_{\theta_m}(p) \;=\; \frac{\mathcal{L}_{\theta_m}(p;\mathcal{D}) - \mu_m}{\sigma_m + \epsilon}, \quad (5)$$

where μ_m and σ_m are the mean and standard de-

Method	Math		Big-Bench Hard					Avg.
	GSM8K	AQUA-RAT	CommonSense & Factual	Language & Semantics	Logic & Reasoning	Math & Arithmetic	Spatial /Seq./Attr.	
Prompting Heuristics								
AO	87.11	75.98	62.52	60.54	71.73	35.60	71.04	66.36
CoT	90.67	84.25	67.98	67.86	85.87	67.52	91.06	79.33
RaR	93.86	82.68	71.50	70.02	86.00	65.28	85.44	79.26
StepBack	92.49	81.10	73.32	73.04	86.37	67.52	90.06	80.56
Iterative Prompt Optimization								
OPRO	93.63	81.89	70.88	72.14	86.50	68.96	90.56	80.65
PromptWizard	88.17	79.92	60.78	73.34	87.90	63.44	84.50	76.87
Black-Box Search								
EvoPrompt	93.33	84.25	71.18	73.72	86.93	70.48	91.24	81.60
Gradient-inspired APO								
TextGrad	93.86	80.71	72.78	72.96	86.04	64.80	88.90	80.01
PMPO	<u>93.93</u>	<u>84.65</u>	<u>75.82</u>	<u>77.56</u>	<u>88.80</u>	<u>71.76</u>	**92.48**	83.00
Ours	**94.09**	**85.04**	**77.05**	**78.11**	**90.76**	**72.16**	<u>92.36</u>	**84.22**

Table 1: **Main results on closed-form benchmarks** (accuracy, %). **Avg.** is the mean over all 7 task categories. Colors denote method families; best is **bold** and second-best is <u>underlined</u>.

viation of $\{\mathcal{L}_{\theta_m}(p;\mathcal{D}) : p \in \mathcal{C}\}$, and ϵ is a small constant. We then define the aggregated objective as a convex combination over the normalized losses:

$$\mathcal{J}(p) = \sum_{m=1}^{M} w_m \widetilde{\mathcal{L}}_{\theta_m}(p). \tag{6}$$

The weights satisfy $w_m \geq 0$ and $\sum_{m=1}^{M} w_m = 1$. Unless otherwise specified, we use uniform weights $w_m = 1/M$. This normalization prevents any single judge from dominating the aggregated score solely due to scale differences. The best prompt under the multi-judge criterion is

$$p^\star = \arg\min_{p \in \mathcal{P}} \mathcal{J}(p), \tag{7}$$

where $\mathcal{P}$ denotes the set of candidates produced by the editing procedure.

4 Experiment

4.1 Experimental Results and Analysis

Benchmarks. We evaluate our method on benchmarks spanning mathematical reasoning, logical inference, open-ended instruction following, and adversarial safety robustness: GSM8K(Cobbe et al., 2021) and AQUA-RAT(Ling et al., 2017) for multi-step quantitative reasoning, Big-Bench Hard(Suzgun et al., 2022) for broader logical/compositional reasoning with minimal task-specific guidance, AlpacaEval 2.0(Dubois et al., 2025) for general instruction following (judged via GPT-4 Turbo pairwise comparisons), and AdvBench(Chen et al., 2022) for adversarial safety stress tests. All datasets are evaluated using Qwen2.5-14B-instruct. Across all evaluations, we optimize against datasets with explicit reasoning traces; implementation details are provided in the appendix.

Baselines. We compare against representative baselines spanning handcrafted prompting heuristics and automated prompt optimization. For manual heuristics, we include CoT(Wei et al., 2022), RaR(Deng et al., 2024b), and StepBack(Zheng et al., 2023), which elicit intermediate reasoning and structured reframing. For automated optimization, we consider Iterative refiners (AO, OPRO(Yang et al., 2023), PromptWizard(Agarwal et al., 2025)), black-box population search (EvoPrompt(Tong et al., 2025)), and gradient/loss-inspired methods (TextGrad(Yuksekgonul et al., 2024), PMPO(Zhao et al., 2025)) that update prompts via feedback signals or likelihood objectives. In addition, for prompt-only jailbreak evaluation, we include Persona Prompt(Zhang et al., 2025), GCG(Zou et al., 2023), ICA(Wei et al., 2023), and MSJ(Anil et al., 2024) as representative attack baselines. All baselines are evaluated under the same protocol across benchmark datasets.

Implementation Details. We use a generator driven prompt optimization pipeline. The prompt

generator is `claude-sonnet-4-5-thinking`. We evaluate candidate prompts using an ensemble of three open source judge models, `gpt-oss-20B`(OpenAI, 2025), `qwen2.5-14b-instruct`(Team et al., 2024), and `gemma3-27b-it`(Team et al., 2025). The final evaluation score is the uniform average of the three judges, with each model assigned weight $1/3$. For each dataset, we randomly sample 20% of examples for training, capped at 50 instances, and evaluate on the remaining examples. At each optimization iteration, we select the top $k = 3$ most challenging training samples according to the ensemble score, and for each selected sample we generate 3 prompt variants. We run optimization for 10 iterations. All experiments are executed on 8 NVIDIA A100 GPUs.Implementation details are provided in the appendix.

4.2 Experimental Results and Analysis

Main Result of Closed Tasks. As summarized in Table 1, our method achieves the best overall average accuracy (**84.22**) across GSM8K, AQUA-RAT, and BBH, outperforming the strongest baseline PMPO (**83.00**) and the competitive black-box search method EvoPrompt (**81.60**). Importantly, this advantage is not driven by a single dataset: we obtain the top scores on both math benchmarks (94.09 on GSM8K and 85.04 on AQUA-RAT) and on four of the five BBH subsets (e.g., 90.76 on logic/reasoning and 72.16 on math/arithmetic). Such a distribution of gains suggests that the method improves general closed-form reasoning competence rather than exploiting idiosyncrasies of a particular benchmark.

Beyond the absolute numbers, the comparison reveals qualitative differences between method families. Handcrafted prompting heuristics provide large jumps over naive prompting but tend to exhibit uneven strengths: they are strong on some BBH reasoning subsets yet can lag on arithmetic-heavy or compositional tasks (e.g., AO has a much lower math/arithmetic score). In contrast, automated optimization methods are more stable across categories, reflecting their ability to adapt prompts to diverse error modes through iterative refinement or search. Our method further strengthens this stability: it maintains high performance on already strong dimensions. This pattern indicates that our approach primarily reduces systematic reasoning failures (e.g., multi-step quantitative slips and rule-application errors), yielding broader average gains even when some subsets are near saturation and leave limited headroom.

Method	LC Win	Win	Len
GPT-4-Turbo	55.02	46.12	1802
Claude-3.5-Sonnet	52.37	40.56	1488
GPT-4-1106 (ref.)	50.00	50.00	2049
Ours	45.56	48.58	2186
PMPO	42.57	51.49	2361
Claude-3-Opus	40.51	29.11	1388
Qwen2.5-32B	38.73	34.05	1819
Llama-3.1-70B	38.06	39.13	2044
Qwen2.5-14B	36.93	32.13	1814

Table 2: Results on AlpacaEval 2.0. We report Length-Controlled Win Rate (LC Win), Win Rate, and Avg Length. Rows are sorted by LC Win (descending).

Open-ended Tasks. We report performance on AlpacaEval 2.0, an open-ended instruction-following benchmark evaluated via pairwise comparisons using GPT-4 Turbo as the automatic judge. Table 2 summarizes the results with both the standard win rate and the length-controlled win rate (LC win rate), together with average response length. All methods are evaluated in the same setting; in particular, our method uses optimized prompts applied to the Qwen2.5-14B instruct model.

Overall, our method achieves a 48.58% win rate and a 45.56% length-controlled win rate, with an average length of 2186 tokens. While PMPO attains the highest raw win rate (51.49%), its LC win rate drops to 42.57%, indicating that a non-trivial portion of its raw gains may be attributable to verbosity-induced evaluator bias. In contrast, our method yields a +2.99 absolute improvement in LC win rate over PMPO (45.56 vs. 42.57) while also producing shorter responses on average (2186 vs. 2361), suggesting better quality that is less dependent on response length. Notably, relative to the GPT-4 Turbo reference system (50% by definition under self-comparison), our approach remains competitive in raw win rate and demonstrates stronger robustness under length control than the strongest prompt-optimization baseline.

Safety Robustness on AdvBench. To assess adversarial safety robustness, we additionally evaluate on AdvBench using prompt-only jailbreak attacks that keep the original user request unchanged, adding only auxiliary context such as prefixes, suf-

Method	Math		Big-Bench Hard					Avg.
	GSM8K	AQUA-RAT	CommonSense & Factual	Language & Semantics	Logic & Reasoning	Math & Arithmetic	Spatial /Seq./Attr.	
Closed-source target model: `gpt-4.1-nano`								
CoT	90.98	83.86	69.26	68.74	86.64	78.72	93.72	81.70
OPRO	91.66	84.25	63.74	59.09	78.88	73.04	76.57	75.32
TextGrad	92.04	83.46	71.34	74.24	90.31	78.40	95.11	83.56
Ours	**93.03**	**85.04**	**76.25**	**76.79**	**94.40**	**80.80**	**97.81**	**86.30**
Small open-source target model: `qwen2.5-7b-it`								
CoT	91.13	72.44	51.66	60.66	78.77	62.48	81.70	71.26
OPRO	90.75	**76.38**	55.98	62.24	78.97	60.96	**84.68**	72.85
TextGrad	91.28	74.80	41.97	46.67	72.16	58.72	75.69	65.90
Ours	**91.36**	75.59	**56.28**	**66.72**	**80.72**	**68.80**	82.26	**74.53**

Table 3: **Prompt portability across model families** (accuracy, %). Colors denote method families; best is **bold** and second-best is underlined (within each target-model block).

fixes, or in-context demonstrations. Table 4 reports the attack success rate (ASR) on Qwen2.5-14B-Instruct. Overall, our method achieves an ASR of **46.92%**, outperforming strong prompt-based baselines. These results show that optimized prompt framing can substantially increase the effectiveness of prompt-only jailbreaks even under the constraint that the user request itself is not modified.

Notably, the margin over MSJ suggests that optimization-driven prompt synthesis can outperform simply scaling the number of demonstrations by discovering higher-leverage contextual structures that more reliably induce unsafe compliance. These gains are achieved without altering the malicious instruction content, but purely by refining surrounding context, consistent with our objective that small, targeted changes in framing can produce large behavioral shifts.

Method	ASR (%)
Original	0.2
PP	9.6
GCG	23.85
ICA	1.73
MSJ	37.18
Ours	**46.92**

Table 4: AdvBench attack success rate (ASR) on Qwen2.5-14B-Instruct for prompt-only attacks.

Method	Tokens (M)
PW	0.084
OPRO	4.1
TextGrad	1.1
EvoPrompt	5.5
Ours	**0.11**

Table 5: Total token consumption comparison (in millions, M).

4.3 Portability and Transferability.

We evaluate whether the optimized prompt can transfer across substantially different target models. Concretely, we first run prompt optimization on our development split to obtain an optimized prompt (and similarly, baseline prompts for other methods). We then freeze the prompt and directly apply it to new target models without any re-optimization, calibration, or additional demonstrations: (i) a closed-source model, gpt-4.1-nano, and (ii) a smaller open-source model, qwen2.5-7b-it. Table 3 reports the transfer results.

On gpt-4.1-nano, our prompt achieves the best overall average accuracy (86.30%), improving over the strongest baseline TextGrad (83.56%) with consistent gains across all BBH subsets. On qwen2.5-7b-it, despite the tighter capacity bottleneck, our method remains best on average (74.53%) and attains the top score on multiple categories, while staying competitive on the remaining ones. Overall, these results indicate that our optimized prompt captures transferable behavioral scaffolding rather than model-specific quirks, and can be deployed reliably on both closed-source endpoints and smaller open-source backbones.

4.4 Cost analysis

We analyze optimization efficiency using *total token consumption* during the prompt search stage, i.e., the sum of (i) generator-side tokens used to propose candidate prompts and (ii) evaluator-side tokens processed to score candidates. As shown in Table 5, our method consumes **0.11M** tokens, comparable to PromptWizard and substantially lower than OPRO, EvoPrompt, and TextGrad, indicating that search-heavy baselines incur steep scaling costs when they repeatedly evaluate many candidates across iterations or populations.

This efficiency primarily comes from two design choices. First, our pipeline requires only *one-*

Variant	Math	BBH	Avg
Full method	89.57	82.09	84.22
– w/o mask	88.43	80.76	82.95
– w/o normalization	89.32	81.88	84.01
– w/o multi-judge	88.94	81.48	83.61
– random selection	86.84	79.13	81.33

Table 6: Ablations on masking attribution and evaluator design.

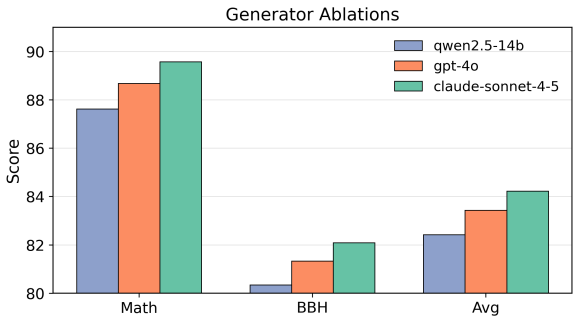

Figure 3: Generator ablations.Only the prompt generator is changed; evaluation and selection remain identical across rows.

way prompt generation: the generator proposes candidates once per step, without multi-round self-critique or refinement dialogues. Second, evaluation uses cross-entropy scoring with open-source judge models, avoiding long-form decoding for every candidate and enabling batched, computation-efficient scoring. In contrast, OPRO appends prior prompts and scores to the context, increasing per-iteration prompt length and token usage; EvoPrompt evaluates populations across generations, so cost scales with population size and the number of generations; and TextGrad is dominated by validation-set selection (in our accounting, 1.05M of its 1.1M tokens are spent on GPT-3.5 evaluation, with the remainder for prompt generation). Overall, these design choices explain why our method achieves strong performance while staying near the low-cost end of the optimization spectrum.

4.5 Ablation Study

We conduct ablations to isolate the contribution of masking-based attribution and evaluator design, as well as the impact of the prompt generator. All ablations follow the same protocol as the main experiments; unless otherwise specified, only the component under study is modified while keeping the remaining pipeline unchanged.

Masking and evaluator ablations. Table 6 shows that removing masking-based attribution leads to a consistent degradation on both Math and BBH, indicating that targeted edits informed by attribution are important for reducing systematic reasoning errors. In contrast, removing z-score normalization yields only a minor regression, suggesting that normalization primarily plays a stabilizing role by reducing scale mismatch across evaluators rather than being a dominant driver of gains. We further observe that collapsing to a single evaluator (self-judge) results in a small drop, consistent with reduced robustness when optimization relies on a single model's preferences. Finally, when we remove evaluator-based selection entirely (while still generating candidates), performance drops more noticeably, highlighting the importance of explicit selection signals for maintaining iteration quality and preventing drift across optimization steps.

Generator sensitivity. We also replace the prompt generator while keeping the rest of the pipeline fixed (Table 3). Using `gpt-4o` as generator yields a small regression relative to the default, indicating that the method is not overly sensitive to the exact choice of a strong generator. Replacing the generator with `qwen2.5-14b` produces a slightly larger drop, consistent with weaker proposal quality and reduced diversity in candidate prompts; nevertheless, performance remains competitive, suggesting that the evaluator-driven optimization loop can partially compensate for a weaker generator by selecting and consolidating higher-quality candidates across iterations.

5 Conclusion

In this work, we proposed GMPO, a scalable framework for automated prompt optimization that improves both efficiency and robustness. By estimating prompt-segment importance with a first-order gradient approximation, GMPO concentrates edits on the most influential parts of a prompt while keeping computation low, and it selects candidates using an ensemble objective over multiple lightweight judges to mitigate evaluator bias. Importantly, this design jointly balances edit locality and evaluation diversity, enabling reliable optimization without relying on a single expensive or potentially biased evaluator. Extensive experiments across diverse tasks demonstrate consistent quality improvements, reduced optimization overhead, and strong transfer to unseen evaluators and settings.

Limitations

Our segment-importance estimator is based on a continuous masking direction and a first-order approximation of the loss change. While this yields an efficient signal, it may not faithfully capture the true causal contribution of a prompt segment, especially when effects are highly non-linear, depend on interactions across segments, or differ between continuous interpolation and discrete text edits. Consequently, the resulting mask-based attribution should be interpreted as a heuristic proxy rather than a ground-truth measure of prompt importance.

Although the multi-judge design reduces reliance on a single evaluator, optimization can still be sensitive to the particular choice, calibration, and weighting of judge models. Different judge set may induce different preference landscapes, potentially affecting which prompt variants are selected and how well they transfer. In addition, using entropy as an evaluation signal typically requires deploying and maintaining multiple open-source judge models to score candidates, which increases engineering complexity and compute requirements and may limit practicality in resource-constrained settings.

Ethical Considerations

This work carries inherent dual-use risk: while the studied techniques can support systematic safety evaluation and mitigation, they could also be misused to increase jailbreak success and elicit unsafe behavior. Accordingly, we follow a minimum-necessary-disclosure approach and omit from the paper and public materials actionable details that would enable direct replication or scalable misuse (e.g., concrete jailbreak prompts, key parameter configurations, complete attacker meta-prompts, and turnkey automation). Our experiments are conducted primarily in an unprotected setting to isolate variables and characterize base-model failure modes, and therefore should be interpreted as stress tests of underlying alignment behavior rather than a comprehensive assessment of any product-grade, multi-layer safety stack.

References

Eshaan Agarwal, Raghav Magazine, Joykirat Singh, Vivek Dani, Tanuja Ganu, and Akshay Nambi. 2025. Promptwizard: Optimizing prompts via task-aware, feedback-driven self-evolution. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19974–20003.

Cem Anil, Esin Durmus, Nina Panickssery, Mrinank Sharma, Joe Benton, Sandipan Kundu, Joshua Batson, Meg Tong, Jesse Mu, Daniel Ford, and 1 others. 2024. Many-shot jailbreaking. *Advances in Neural Information Processing Systems*, 37:129696–129742.

Bowen Cao, Deng Cai, Zhisong Zhang, Yuexian Zou, and Wai Lam. 2024. On the worst prompt performance of large language models. *Advances in Neural Information Processing Systems*, 37:69022–69042.

Anwoy Chatterjee, HSVNS Kowndinya Renduchintala, Sumit Bhatia, and Tanmoy Chakraborty. 2024. Posix: A prompt sensitivity index for large language models. *arXiv preprint arXiv:2410.02185*.

Yangyi Chen, Hongcheng Gao, Ganqu Cui, Fanchao Qi, Longtao Huang, Zhiyuan Liu, and Maosong Sun. 2022. Why should adversarial perturbations be imperceptible? rethink the research paradigm in adversarial nlp. *arXiv preprint arXiv:2210.10683*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Sarkar Snigdha Sarathi Das, Ryo Kamoi, Bo Pang, Yusen Zhang, Caiming Xiong, and Rui Zhang. 2024. Greater: Gradients over reasoning makes smaller language models strong prompt optimizers. *arXiv preprint arXiv:2412.09722*.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. 2022. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 3369–3391.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. 2024a. Rephrase and respond: Let large language models ask better questions for themselves. *Preprint*, arXiv:2311.04205.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. 2024b. Rephrase and respond: Let large language models ask better questions for themselves. *Preprint*, arXiv:2311.04205.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. 2025. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *Preprint*, arXiv:2404.04475.

Yassir Fathullah and Mark Gales. 2025. Efficient prompt optimization for comparative llm-as-a-judge through uncertainty estimation. In *First International KDD Workshop on Prompt Optimization, 2025*.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. Promptbreeder: Self-referential self-improvement via prompt evolution. *arXiv preprint arXiv:2309.16797*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*.

OpenAI. 2025. gpt-oss-120b gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, Haozhu Wang, Han Ding, Yuzhe Lu, Zhichao Xu, Yun Zhou, Balasubramaniam Srinivasan, Qiaojing Yan, Yueyan Chen, Haibo Ding, and 2 others. 2025a. A systematic survey of automatic prompt optimization techniques. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, page 33066–33098. Association for Computational Linguistics.

Kiran Ramnath, Kang Zhou, Sheng Guan, Soumya Smruti Mishra, Xuan Qi, Zhengyuan Shen, Shuai Wang, Sangmin Woo, Sullam Jeoung, Yawei Wang, and 1 others. 2025b. A systematic survey of automatic prompt optimization techniques. *arXiv preprint arXiv:2502.16923*.

Lennart Schneider, Martin Wistuba, Aaron Klein, Jacek Golebiowski, Giovanni Zappella, and Felice Antonio Merra. 2024. Hyperband-based bayesian optimization for black-box prompt selection. *arXiv preprint arXiv:2412.07820*.

Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2023. Quantifying language models' sensitivity to spurious features in prompt design or: How i learned to start worrying about prompt formatting. *arXiv preprint arXiv:2310.11324*.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. *arXiv preprint arXiv:2010.15980*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and 1 others. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, and 1 others. 2025. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*.

Qwen Team and 1 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2(3).

Zeliang Tong, Zhuojun Ding, and Wei Wei. 2025. Evoprompt: Evolving prompts for enhanced zero-shot named entity recognition with large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 5136–5153.

Yaxuan Wang, Quan Liu, Zhenting Wang, Zichao Li, Wei Wei, Yang Liu, and Yujia Bao. 2025. Promptbridge: Cross-model prompt transfer for large language models. *arXiv preprint arXiv:2512.01420*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zeming Wei, Yifei Wang, Ang Li, Yichuan Mo, and Yisen Wang. 2023. Jailbreak and guard aligned language models with only few in-context demonstrations. *arXiv preprint arXiv:2310.06387*.

Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. 2024. Prompt optimization with ease? efficient ordering-aware automated selection of exemplars. *Advances in Neural Information Processing Systems*, 37:122706–122740.

Jinyu Xiang, Jiayi Zhang, Zhaoyang Yu, Fengwei Teng, Jinhao Tu, Xinbing Liang, Sirui Hong, Chenglin Wu, and Yuyu Luo. 2025. Self-supervised prompt optimization. *arXiv preprint arXiv:2502.06855*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409*.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. Textgrad: Automatic "differentiation" via text. *Preprint*, arXiv:2406.07496.

Zheng Zhang, Peilin Zhao, Deheng Ye, and Hao Wang. 2025. Enhancing jailbreak attacks on llms via persona prompts. *arXiv preprint arXiv:2507.22171*.

ChenZhuo Zhao, Ziqian Liu, Xinda Wang, Junting Lu, and Chaoyi Ruan. 2025. PMPO: Probabilistic metric prompt optimization for small and large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2025*.

Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H Chi, Quoc V Le, and Denny Zhou. 2023. Take a step back: Evoking reasoning via abstraction in large language models. *arXiv preprint arXiv:2310.06117*.

Andy Zou, Zifan Wang, Nicholas Carlini, Milad Nasr, J Zico Kolter, and Matt Fredrikson. 2023. Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

Appendix

A Full Ethical Considerations

Dual-use risk and research scope. The techniques studied in this paper are inherently dual-use. On the one hand, they can support systematic safety evaluation and help improve alignment and guardrails. On the other hand, they may be misused to increase jailbreak success rates and elicit unsafe or policy-violating model behavior. Since our results suggest that relatively small framing changes can substantially increase unsafe compliance, we follow a *minimum necessary disclosure* principle: we provide sufficient information to understand and validate the scientific claims while reducing the risk of real-world misuse.

Release policy and controlled access. For safety reasons and to mitigate misuse risk, we will **not** include in the paper or public appendix: (i) concrete jailbreak prompts and their optimized variants; (ii) detailed parameter choices and configurations that would enable direct replication of optimized jailbreak prompts at scale; or (iii) the complete attacker *meta-prompt* and turnkey scripts that could facilitate automated exploitation. When replication is necessary for bona fide safety research, the attacker meta-prompt and essential implementation details will be shared **only with authorized researchers** under controlled access (e.g., with a clear research-purpose statement and an agreement not to deploy the materials for real-world attacks or broad redistribution), subject to reasonable review.

Evaluation setting and limitations. Our experiments are conducted primarily in an *unprotected* setting (i.e., without external guardrails) to isolate variables and characterize the underlying failure modes under our core constraint: the method focuses on manipulating context *without rewriting the user query*. This choice does not imply that real-world deployments lack defenses. Production systems often include multiple safety layers (e.g., prompt-injection detection, policy classifiers, refusal templates, and logging/auditing). Therefore, our findings should be interpreted as stress-testing base-model and base-alignment behavior rather than as a complete assessment of any product-grade safety stack.

Mitigation recommendations. Because our method operates via contextual manipulation without rewriting the user request, we recommend strengthening *prompt safety* and *context integrity* defenses, including but not limited to:

- **Input-side detection:** detect prompt-injection and jailbreak patterns (e.g., role hijacking, instruction-priority manipulation, policy evasion phrasing, hidden instruction structures) and trigger refusal, de-escalation, or human review when appropriate.
- **Context integrity constraints:** ensure system and safety prompts cannot be overridden or redefined by user-provided context; enforce robust instruction hierarchy and sandboxing for untrusted context.
- **Output-side guardrails:** apply policy checkers or safety classifiers to block, rewrite, or terminate unsafe generations before delivery, and log triggers for iterative hardening.
- **Adversarial training and red-teaming:** incorporate the attack patterns highlighted in this work into continuous red-team evaluation and regression test suites, and evaluate under safer configurations (e.g., enabled guardrails, alternative system prompts, stricter refusal thresholds).

Intended use. We publish this work to advance the study of alignment vulnerabilities and to support the development of stronger mitigations and evaluation protocols. We do not endorse the use of these techniques for unauthorized attacks against deployed systems or for any unlawful or harmful activities.

B Detailed Prompts of ours

In this section, we provide the meta prompts used throughout our framework, covering (i) prompt optimization in the GMPO-style pipeline, and (ii) evaluation prompts used for scoring, ranking, and success-rate measurement. We also include the segment-level masking / loss attribution prompts used to identify high-impact spans for editing.

Prompt Optimization (Large Model). We use a capable instruction-following model to propose improved prompts conditioned on task description and optional diagnostics.

Optimization Prompt for Large Models

```
You are an expert prompt engineer tasked with
↪  dynamically improving prompts to generate more
↪  effective, robust solutions across models and
↪  domains.

Task description:
{task_description}

We are using multiple judge models and a
↪  gradient-based importance estimator that
↪  highlights segments of the current instruction
↪  prompt which are most critical for
↪  performance.{importance_text}

Given:
- Current instruction prompt:
{current_prompt}

- Example Input:
{user_input}

- Example Final Answer:
{true_answer}{reject_block}

Your job:
1. Diagnose weaknesses of the current prompt for this
↪  task.
2. Respect the task_description as the full scope of
↪  capabilities; do NOT specialize only for this
↪  example.
3. Use the importance information to preserve truly
↪  critical parts, and refine or restructure less
↪  important parts.
4. Produce a new instruction prompt that:
   - Is clear, robust, and domain-general for this
   ↪  task family.
   - Encourages step-by-step reasoning.
   - Works well across different judge models (not
   ↪  just the generator itself).

Return ONLY the final improved prompt, wrapped in
↪  <prompt></prompt> with PLAIN TEXT format (no
↪  markdown).
```

Importance Prompt (Segment-level Attribution). We use an importance (attribution) prompt to summarize which parts of the current instruction are most responsible for model behavior, based on segment-level signals (e.g., masking / loss attribution statistics or other per-span diagnostics). The goal is not to rewrite the prompt directly, but to produce a concise, actionable list of high-impact spans and the likely reasons they matter, so that the optimizer can focus edits on the most influential regions while minimizing unnecessary changes.

Importance Prompt (Segment-level)

```
\n\nGradient-based important segments of current
↪  prompt:\n{importance_summary}\n

{importance_summary}:
[Segment {seg_idx}] score={score:.4f} text:
↪  {repr(text)}
```

Preference Pair Prompt (Positive/Negative Samples). We use a preference pair prompt constructed from a positive (preferred) and a negative (rejected) sample under the same evaluation context. Given the task description and input, the judge model is asked to verify (or restate) which of the two candidates better satisfies the task requirements and adheres to the desired behavior. This prompt is designed to elicit a clear preference signal grounded in an explicit rubric, enabling supervision from relative comparisons rather than absolute scores.

Preference Pair Prompt (Chosen vs. Rejected)

```
- Example Rejected Answer (should be discouraged):
{rejected_answer}

Rejection setup:
- reject_objective: {reject_objective}
- reject_weight(alpha): {reject_weight}
- dpo_beta(beta): {dpo_beta}
```

Correctness Judge Prompt (Binary / Structured Scoring). We use a dedicated correctness judge prompt to determine whether a model output satisfies the task requirements. The judge is provided with the task description, the user input, and the model output, and optionally a reference answer when available. It must return a structured decision (e.g., pass/fail or a binary label) and, if needed, a minimal rationale. This judge is used to compute accuracy-style metrics and to support automated evaluation in settings where exact-match is insufficient.

Correctness Judge Prompt

```
You are an expert evaluator determining whether an
↪  answer matches the ground truth. Treat equivalent
↪  formats such as "A", "(A)", and "A." as correct,
↪  and focus on the meaning rather than exact string
↪  matching. When the answer format is important,
↪  verify that the model responds in the required
↪  format; for example, in Dyck language problems, if
↪  the question asks what follows "([{" and the
↪  ground truth is "])", then a model answer of
↪  "([{}])" should be considered correct because it
↪  includes the proper closing brackets.
```

C Experiment Details

C.1 Tasks and Data Details

We evaluate our approach on a set of benchmarks that collectively cover (i) closed-form multi-step reasoning with objective, accuracy-based scoring, (ii) open-ended instruction following assessed via preference-style comparisons, and (iii) adversarial safety robustness under jailbreak-oriented prompts. Specifically, we include Big-Bench Hard (BBH) for broad reasoning, GSM8K and AQUA-RAT for quantitative and algebraic reasoning, AlpacaEval

2.0 for instruction-following quality under pairwise judging, and AdvBench for adversarial safety stress-testing.

C.1.1 Big-Bench Hard (BBH)

Big-Bench Hard (BBH) is a curated subset of particularly challenging tasks drawn from BIG-bench, designed to probe multi-step reasoning abilities where standard few-shot prompting historically underperforms. BBH spans diverse reasoning skills (e.g., logical inference, compositional generalization, and structured decision making) and is commonly evaluated using task-specific accuracy over a fixed set of problems and prompts (Suzgun et al., 2022).

BBH task groupings. Following common practice, we organize BBH tasks into five coarse-grained categories:

- **CommonSense&Factual:** `causal_judgement`, `movie_recommendation`, `salient_translation`, `sports_understanding`, `web_of_lies`.
- **Language&Semantics:** `disambiguation_qa`, `hyperbaton`, `ruin_names`, `snarks`, `word_sorting`.
- **Logic&Reasoning:** `boolean_expressions`, `formal_fallacies`, `logical_deduction`.
- **Math&Arithmetic:** `date_understanding`, `dyck_languages`, `geometric_shapes`, `multistep_arithmetic_two`, `object_counting`.
- **Spatial/Sequential/Attribute:** `navigate`, `penguins_in_a_table`, `reasoning_about_colored_objects`, `temporal_sequences`, `tracking_shuffled_objects`.

Dataset for prompt generation. To construct prompts that elicit explicit chain-of-thought solutions, we use `gpt-4o` to generate reasoning-annotated answers on BBH. We then use `gpt-4o` again as an automatic judge to verify whether each generated answer is correct. From the verified pool, we randomly sample 50 chain-of-thought instances for targeted analysis and reporting, while also conducting evaluation over the full BBH dataset. We will release the resulting prompt-generation dataset to support reproducibility and follow-up research.

C.1.2 GSM8K

GSM8K is a dataset of grade-school math word problems written by human annotators to test multi-step numerical reasoning. Problems typically require a short sequence of elementary arithmetic operations to reach a final numeric answer, and performance is usually measured by exact-match accuracy on the final result (Cobbe et al., 2021).

Dataset for prompt generation. We use `openai/gsm8k` and treat the ground-truth `answer` field as the optimization target. For detailed, sample-level evaluation, we select 50 problems from the training split as an evaluation subset, and we additionally report performance on the full dataset.

C.1.3 AQUA-RAT

AQUA-RAT (Algebra Question Answering with Rationales) is a large-scale collection of algebraic word problems presented in multiple-choice format and accompanied by natural-language rationales. The benchmark targets algebraic and quantitative reasoning, and it is typically evaluated by answer-choice accuracy while also enabling rationale-aware modeling and analysis (Ling et al., 2017).

Dataset for prompt generation. We use `deepmind/aqua_rat` and treat the ground-truth `rationale` field as the optimization target. For detailed, sample-level evaluation, we select 50 problems from the training split as an evaluation subset, and we additionally report performance on the full dataset.

C.1.4 AlpacaEval 2.0

AlpacaEval 2.0 is an automated instruction-following evaluation framework built around a fixed set of user-like instructions, where model responses are compared head-to-head against a reference system using an LLM judge. In addition to standard win-rate, AlpacaEval 2.0 emphasizes length-controlled comparisons to reduce confounding effects from verbosity and better approximate quality judgments (Dubois et al., 2025).

Dataset for prompt generation (AlpacaEval 2.0). Because AlpacaEval 2.0 consists of open-ended instructions, we optimize prompts using the preference pairs provided in `reciprocate/alpaca-eval`. Concretely, we treat the `selected` responses as preferred samples and

the `rejected` responses as dispreferred samples. We use 50 preference pairs for prompt optimization. For evaluation, we follow the default AlpacaEval configuration, using `gpt-4-1106-preview` as the judge, and we also report results on the full dataset.

C.1.5 AdvBench

AdvBench is a widely used adversarial safety benchmark consisting of harmful-behavior instructions intended to test whether models can be induced to generate disallowed or unsafe content. It is commonly used in jailbreak evaluation, where attack success is measured by whether the model produces an affirmative or materially compliant response to the harmful instruction, often summarized as attack success rate (ASR) (Chen et al., 2022).

Dataset for prompt generation. We use `walledai/AdvBench` and treat the `target` field as the optimization target. We employ a jailbroken `DeepSeek-v3.2` model as the generator; for safety reasons, we do not disclose the jailbreak prompt. We select 50 instances from the dataset for detailed evaluation and additionally report results on the full dataset.

C.2 Baselines

This section describes the comparison methods included in our experiments. We consider representative baselines spanning handcrafted prompting heuristics, automated prompt optimization algorithms, and prompt-only jailbreak attacks. For each baseline, we follow the original paper's recommended setup (e.g., prompt templates, search/iteration budgets, selection criteria, and any task-specific formatting rules) to the extent applicable, and we apply the same evaluation protocol across all benchmark datasets.

C.2.1 Prompt Optimization Methods

Answer-Only (AO). AO is a minimalist prompting baseline that instructs the model to output only the final answer (i.e., no intermediate reasoning or explanations). This baseline helps isolate the benefit of explicit reasoning elicitation from the underlying model capability.

Chain-of-Thought (CoT). CoT prompts the model to produce explicit intermediate reasoning steps before giving the final answer, typically via few-shot exemplars that include rationales. This approach is known to improve performance on multi-step reasoning tasks by encouraging structured deliberation (Wei et al., 2022).

StepBack. StepBack Prompting first asks the model to abstract from the specific instance ("take a step back") to derive high-level concepts or principles, and then uses those abstractions to guide a second-stage solution. The method targets better global reasoning trajectories by explicitly separating abstraction from execution (Zheng et al., 2023).

Rephrase and Respond (RaR). RaR asks the model to rephrase and expand the user question into a form it can better interpret, and then answer the rephrased query (optionally with a two-stage rephraser/responder variant). This aims to reduce misinterpretation errors and can be complementary to reasoning-oriented prompts (Deng et al., 2024b).

Optimization by PROmpting (OPRO). OPRO treats prompt engineering as an optimization problem solved with an LLM acting as an optimizer: given task feedback (e.g., development-set scores), the optimizer proposes revised prompts iteratively. OPRO is a model-guided refinement baseline that searches for improved instructions in natural language (Yang et al., 2023).

PromptWizard. PromptWizard is a feedback-driven prompt optimization framework that iteratively generates candidate prompts, critiques failures, and synthesizes improved prompts, with mechanisms designed to balance exploration and exploitation. It optimizes discrete prompt text (and optionally in-context examples) under black-box access to the target model (Agarwal et al., 2025).

EvoPrompt. EvoPrompt performs population-based prompt search inspired by evolutionary algorithms. It maintains a set of candidate prompts and applies evolutionary operators (e.g., variation and selection) guided by validation performance, enabling black-box optimization without gradients or parameter access (Tong et al., 2025).

TextGrad. TextGrad frames prompt improvement through a gradient metaphor in which an LLM produces "textual gradients" (feedback signals) that can be backpropagated through a text-based computation graph to update prompts and other textual variables. The method operationalizes iterative refinement using structured feedback rather than model parameter gradients (Yuksekgonul et al., 2024).

PMPO. PMPO (Probabilistic Metric Prompt Optimization) refines prompts using likelihood-based signals (e.g., token-level cross-entropy) as a lightweight objective for selecting and rewriting prompt segments. This loss-inspired approach aims to scale prompt optimization while reducing reliance on expensive output sampling or human preference annotation (Zhao et al., 2025).

C.2.2 Jailbreak Methods (Prompt-Only Attacks)

Persona Prompt. Persona Prompt attacks use role-playing or persona conditioning to shape the model's behavior and reduce refusal tendencies. Recent work shows that automatically crafted persona prompts can substantially increase jailbreak success and can also synergize with other attacks (Zhang et al., 2025).

Greedy Coordinate Gradient (GCG). GCG is a universal adversarial prompt/suffix generation approach that searches for transferable token sequences that induce harmful compliance across aligned models. It is widely used as a strong prompt-only jailbreak baseline due to its transferability and effectiveness under black-box evaluation (Zou et al., 2023).

In-Context Attack (ICA). ICA jailbreaks aligned models using only a few in-context demonstrations that exemplify harmful responses, leveraging in-context learning to shift the model toward unsafe behavior without parameter updates. The associated line of work also studies paired in-context defenses, but here we use ICA strictly as an attack baseline (Wei et al., 2023).

Many-shot Jailbreaking (MSJ). MSJ is a long-context jailbreak that conditions the model on many demonstrations of undesirable behavior (potentially hundreds), exploiting increased context windows to steer the model into continuing the harmful pattern. This baseline evaluates robustness against long-context, demonstration-driven attacks (Anil et al., 2024).

D Implementation details

D.1 Hyperparameter selection

We use a fixed set of hyperparameters across all datasets. Table 7 summarizes the settings used in our optimization procedure. In particular, we set the segment window size to $K = 16$ tokens for importance attribution, use $\epsilon = 1 \times 10^{-6}$ as the stabilization term in z-score normalization, and cap the maximum sequence length at $L_{\max} = 2048$ with bfloat16 enabled. For AlpacaEval-style pairwise training, we set $\alpha = \beta = 1.0$.

Table 7: Hyperparameters used in GMPO optimization.

Hyperparameter	**Value**
Search budget	
Optimization iterations	10
Challenging samples per iteration (k)	3
Prompt variants per sample	3
Training split	20% (max 50 instances per dataset)
Attribution & normalization	
Segment window size (K)	16
Stabilization term in z-score (ϵ)	1×10^{-6}
Model/runtime	
Max sequence length ($L_{\max}$)	2048
bfloat16	Enabled
Pairwise objective (AlpacaEval)	
Reject weight (α)	1.0
DPO temperature (β)	1.0

D.2 Mask-based segment importance estimation

This section details practical choices used to compute mask-direction segment importance in a single forward–backward pass, complementing the main derivation.

Mask embedding choice. We represent the continuous mask direction using a single fixed embedding vector $\mathbf{e}_{\text{mask}}$ taken from the model's input embedding table. When the tokenizer provides a dedicated `[MASK]` token, we use its embedding. Otherwise, we fall back to a special token that is guaranteed to exist for most chat LMs (priority: padding token, then end-of-sequence token); if none are defined, we use token id 0 as a safe default. This design ensures $\mathbf{e}_{\text{mask}}$ is always well-defined without modifying the vocabulary.

Sequence construction and supervised region. For each training example, we form a single token sequence by concatenating: (i) the system instruction, (ii) the user input (in chat format), and (iii) the reference (correct) answer. The training objective for attribution is the conditional negative log-likelihood of the reference answer given the prompt. Concretely, we compute cross-entropy only on the answer tokens: tokens belonging to the system/user prompt are excluded from the loss by assigning them an ignore label, while the answer span is labeled with its corresponding token ids. This prevents the attribution signal from being dominated by prompt-format boilerplate and focuses gradients on how the prompt supports predicting the answer.

Maximum length handling. When enforcing a maximum sequence length, we first build the full prompt+answer sequence and then apply *left truncation* to keep the last $L_{\max}$ tokens. This strategy preferentially preserves the answer span and the immediately preceding prompt context, which are most

relevant for the conditional likelihood. Labels are truncated consistently with the retained tokens; if the reference answer itself exceeds $L_{\max}$, we keep only its suffix aligned to the truncated window.

Gradient extraction w.r.t. input embeddings. To obtain token-level gradients efficiently, we treat the input embeddings $\{\mathbf{e}_t\}$ as the differentiation variables. We run one forward pass to compute the answer-only loss and then obtain $\nabla_{\mathbf{e}_t} L$ via a single backward pass (or an equivalent gradient query) with respect to the input-embedding tensor. No parameter gradients are required for this attribution step, which reduces overhead and isolates the signal needed by the importance estimator.

Segmentation policy. We segment only the *prompt* portion (system + user + generation prefix, excluding the reference answer) into contiguous fixed-length token windows of size K. The last segment may be shorter if the prompt length is not divisible by K. Answer tokens are excluded from segmentation to avoid selecting segments that directly contain ground-truth content.

Scoring, ranking, and interpretation. For each segment s_j, we compute its importance by aggregating token-wise contributions along the mask direction:

$$I_j \;=\; \sum_{t \in s_j} \langle \nabla_{\mathbf{e}_t} L,\; \mathbf{e}_{\text{mask}} - \mathbf{e}_t \rangle \,.$$

In practice, we rank segments by $|I_j|$ and select the Top-k segments for editing. The sign of I_j can be informative: positive values indicate that moving the segment toward $\mathbf{e}_{\text{mask}}$ locally increases the loss, while negative values indicate a local decrease; however, selection is based on magnitude to capture both strongly helpful and strongly harmful spans.

Memory and throughput considerations. To reduce peak accelerator memory during attribution, we construct token ids and labels on CPU and transfer them to the device only for the forward–backward computation. After obtaining $\nabla_{\mathbf{e}_t} L$, we can move the gradients (and, if needed, the corresponding $\mathbf{e}_t$ vectors) back to CPU and release device-resident intermediates before performing segment aggregation and ranking. On CUDA devices, automatic mixed precision (e.g., `bfloat16`) can be enabled for the forward pass to further improve throughput, without changing the estimator definition.

D.3 Pairwise Loss on AlpacaEval

For open-ended instruction following on AlpacaEval, we optimize from preference pairs. Each training instance shares the same context c (instruction prompt and user input) and contains a *chosen* output y^+ as well as a *rejected* output y^-. Given a judge (or scorer) parameterized by θ, we define the token-level negative log-likelihood (cross-entropy) for an output sequence y under the shared context c as

$$\ell_\theta(y \mid c) = -\frac{1}{|y|} \sum_{t=1}^{|y|} \log P_\theta(y_t \mid c, y_{<t}), \tag{8}$$

where $|y|$ denotes the output length. We then compute

$$\text{pos\_loss} = \ell_\theta(y^+ \mid c), \quad \text{neg\_loss} = \ell_\theta(y^- \mid c), \quad \Delta = \text{neg\_loss} - \text{pos\_loss}. \tag{9}$$

To enforce the preference ordering, we adopt a DPO-style pairwise objective defined over the loss gap Δ:

$$\mathcal{L}_{\text{pair}} = \text{softplus}(-\beta\Delta) = \log(1 + \exp(-\beta(\text{neg\_loss} - \text{pos\_loss}))) \,, \tag{10}$$

where β controls the sharpness of the separation between preferred and rejected outputs. The per-example training objective is

$$\mathcal{L} = \text{pos\_loss} + \alpha\, \mathcal{L}_{\text{pair}}, \tag{11}$$

with α weighting the contribution of the pairwise term. In our implementation, we set $\alpha \leftarrow$ `reject_weight` and $\beta \leftarrow$ `dpo_beta`.

Minimizing $\mathcal{L}$ jointly (i) reduces pos_loss, increasing the likelihood of the chosen output, and (ii) reduces $\mathcal{L}_{\text{pair}}$, which is monotonically decreasing in Δ and therefore encourages neg_loss $\gg$ pos_loss under the same context. Equivalently, using $\mathrm{softplus}(-x) = -\log\sigma(x)$, we can rewrite

$$\mathcal{L}_{\text{pair}} = -\log\sigma(\beta(\text{neg\_loss} - \text{pos\_loss}))\,. \tag{12}$$

For numerical stability, we implement $\mathrm{softplus}(\cdot)$ via `logaddexp`:

$$x \leftarrow \beta \cdot (\text{neg\_loss} - \text{pos\_loss}), \tag{13}$$

$$\mathcal{L}_{\text{pair}} \leftarrow \log(\exp(0) + \exp(-x)) \;=\; \texttt{logaddexp}(0, -x), \tag{14}$$

$$\mathcal{L} \leftarrow \text{pos\_loss} + \alpha \cdot \mathcal{L}_{\text{pair}}. \tag{15}$$

Finally, we average $\mathcal{L}$ over the batch, and additionally average across judges when an ensemble is used.

E Full Experiment Results

E.1 Detailed Results on Big-Bench Hard (BBH)

In the main paper, for clarity of presentation, we summarize BBH performance by aggregating task-level accuracies into five high-level categories and reporting category-wise averages. This appendix provides the *complete per-task* results *before* any such aggregation. Specifically, Table 8 reports the test accuracy for each BBH task under the 1-shot setting on Qwen2.5-14B-Instruct, together with the overall average across all tasks. These task-level numbers constitute the underlying data used to compute the category scores and other aggregated metrics reported in the main text. Bold values denote the best-performing method on each task, and underlined values denote the second best

E.2 Open-ended Tasks (AlpacaEval 2.0)

We evaluate open-ended instruction following on AlpacaEval 2.0, where model responses are compared head-to-head against a reference system under an automatic LLM judge (pairwise comparisons). We report both the standard win rate (`win_rate`) and the length-controlled win rate (`length_controlled_winrate`, abbreviated as LC win), together with the average response length (`avg_length`). The LC win metric is designed to mitigate confounding effects from verbosity and better reflect response quality independent of length.

Table 9 provides detailed AlpacaEval results. Our method achieves a win rate of 48.58% and an LC win rate of 45.56%, demonstrating a substantial gain over the same backbone without prompt optimization (Qwen2.5-14B: 32.13% win / 36.93% LC win). Compared with PMPO, our method has a lower raw win rate (48.58% vs. 51.49%) but a higher LC win rate (45.56% vs. 42.57%), while also generating shorter responses on average (2186 vs. 2361 tokens). This pattern is consistent with the interpretation that part of the raw win-rate gains of some methods can be affected by response-length bias, which is explicitly controlled for by the LC win metric.

F Optimized Prompts (Ours)

Task: boolean_expressions

Description: Evaluate the truth value of a random Boolean expression consisting of Boolean constants (True, False) and basic Boolean operators (and, or, not).

BBH Prompt (boolean_expressions)

```
You are evaluating a Boolean expression to determine its truth value (True or False).

Task: Evaluate the given Boolean expression and provide the final result.

Operator Precedence (highest to lowest):
1. NOT - negates a value
2. AND - requires both values to be True
3. OR - requires at least one value to be True

Evaluation Process:

Step 1: Parse the expression to identify all values (True, False) and operators (not, and, or).

Step 2: Apply NOT operators first, working from right to left (closest to the value outward):
- not True = False
- not False = True
- For multiple NOTs in sequence, apply each one step by step

Step 3: Apply AND operators next:
- True and True = True
- Any other combination = False

Step 4: Apply OR operators last:
- False or False = False
- Any other combination = True

Step 5: Show your work clearly by writing out each transformation.

Step 6: State the final answer as either "True" or "False".

Now evaluate the given Boolean expression using this process.
```

Task: causal_judgment

Description: Given a short story, determine how a typical person would answer a causal question about the story.

BBH Prompt (causal_judgement)

```
You are predicting how typical people answer questions about everyday scenarios involving causation, intentionality, and moral responsibility.

Your goal: Determine the answer an ordinary person would give using common sense and everyday intuitions.

Question Types and How People Think:

CAUSATION (Did X cause Y?):
- People focus on what seems abnormal, unusual, or rule-breaking
- Actions that violate norms are singled out as "the cause"
- Normal or permitted actions fade into the background
- Ask: What stands out as wrong or unexpected in this situation?

INTENTIONALITY (Did someone do X intentionally?):
- Intentional means they wanted it as a goal OR knew it would happen
- Harmful side effects feel more "intentional" than helpful ones
- Ask: Did they mean for this to happen or know it would?
```

Table 8: Average test accuracy in the 1-shot setting across multiple tasks for different prompting methods, evaluated on Qwen2.5-14B-Instruct. Bold values indicate the best-performing method for each task; underlined values indicate the second best.

Task Name	AO	CoT	RaR	StepBack	OPRO	EvoPrompt	PromptWizard	PMPO	TextGrad	Ours
boolean_expressions	0.756	0.920	0.952	0.936	0.972	0.952	0.976	0.984	0.968	**0.992**
causal_judgement	0.674	0.631	**0.695**	0.658	0.636	0.647	0.599	**0.695**	0.679	0.684
date_understanding	0.684	0.740	0.708	0.752	**0.800**	0.772	0.636	0.784	0.756	0.796
disambiguation_qa	0.656	0.776	0.716	0.640	0.848	0.760	**0.892**	0.736	0.712	0.700
dyck_languages	0.096	0.240	0.236	0.228	**0.392**	0.308	0.220	0.256	0.020	0.320
formal_fallacies	0.704	0.800	0.784	0.808	**0.856**	0.792	0.816	0.816	0.760	**0.856**
geometric_shapes	0.440	0.616	0.576	**0.684**	0.580	0.620	0.508	0.676	0.624	0.656
hyperbaton	0.632	0.704	0.768	0.848	0.740	0.756	0.752	**0.896**	0.812	0.844
logical_deduction	0.692	0.856	0.844	0.847	0.767	0.864	0.845	0.864	0.853	**0.875**
movie_recommendation	0.564	0.636	0.624	0.640	0.636	0.596	0.676	**0.684**	0.616	0.672
multistep_arithmetic_two	0.052	0.968	0.972	0.956	**0.988**	0.948	0.976	**0.988**	0.972	0.972
navigate	0.660	0.908	0.856	0.924	0.896	0.944	0.848	**0.960**	0.952	0.912
object_counting	0.508	0.812	0.772	0.756	0.688	0.876	0.832	**0.884**	0.868	0.864
penguins_in_a_table	0.753	0.945	0.932	0.952	0.932	0.959	0.726	0.952	0.932	**0.973**
reasoning_about_colored_objects	0.708	0.892	0.768	0.880	0.876	0.872	0.912	0.888	0.840	**0.916**
ruin_names	0.632	0.660	0.556	0.716	0.788	0.680	0.692	0.840	0.724	**0.864**
salient_translation_error_detection	0.600	0.572	0.604	0.644	0.624	0.604	0.504	0.600	0.624	**0.692**
snarks	0.831	0.809	0.837	0.848	0.843	0.882	0.787	0.826	0.792	**0.893**
sports_understanding	0.752	0.660	0.680	0.804	0.828	0.812	0.544	0.836	0.808	**0.860**
temporal_sequences	0.832	0.908	0.864	0.900	**0.964**	0.916	0.900	0.944	0.908	0.900
tracking_shuffled_objects	0.599	0.900	0.852	0.847	0.860	0.871	0.839	0.880	0.813	**0.917**
web_of_lies	0.536	0.900	0.972	0.920	0.820	0.900	0.716	**0.976**	0.912	0.944
word_sorting	0.276	0.444	**0.624**	0.600	0.388	0.608	0.544	0.580	0.608	0.604
Average Accuracy	0.593	0.752	0.747	0.773	0.770	0.780	0.728	0.806	0.763	**0.813**

Model	Win (%)	SE	Wins	BaseWins	Draws	N	Disc. Win (%)	Mode	Avg Len	LC Win (%)	LC SE
gpt-4-turbo-2024-04-09	46.12	1.47	370	426	9	805	46.52	minimal	1802	55.02	–
claude-3-5-sonnet-20240620	40.56	1.47	312	493	0	805	38.76	community	1488	52.37	–
gpt4_1106_preview (ref.)	50.00	0.00	0	0	805	805	50.00	minimal	2049	50.00	–
Qwen2.5-72B	52.17	1.47	414	386	5	805	51.74	community	2282	46.95	–
Ours	48.58	1.53	390	411	2	803	48.69	community	2186	45.56	0.87
PMPO	51.49	1.53	403	398	3	804	50.31	community	2361	42.57	0.82
claude-3-opus-20240229	29.11	1.39	223	579	3	805	27.89	minimal	1388	40.51	–
Qwen2.5-32B	34.05	1.40	261	541	3	805	32.61	community	1819	38.73	–
Meta-Llama-3.1-70B-Instruct-Turbo	39.13	1.43	306	496	3	805	38.20	minimal	2044	38.06	0.90
Qwen2.5-14B	32.13	1.44	248	553	4	805	31.06	community	1814	36.93	0.89

Table 9: Detailed AlpacaEval 2.0 results. `Wins/BaseWins/Draws` denote the number of pairwise comparisons won by the model / won by the reference baseline / tied. `SE` is the standard error reported by the evaluator. `Disc. Win` is the discrete win rate computed from win/loss/tie counts. `Mode` is the AlpacaEval evaluation template/config label. Missing entries ("–") indicate fields not reported in the current evaluator output.

```
MORAL RESPONSIBILITY (Is someone blameworthy?):
- Consider: Did they know? Could they avoid it? Did
↪  they break rules?
- Ask: Would ordinary people blame this person?

How to Answer:

1. Read the scenario carefully
2. Identify what is unusual, norm-violating, or
↪  unexpected
3. Apply common sense intuition for the question type
4. Select your answer based on how typical people
↪  would reason

Focus on what ordinary people naturally notice and
↪  care about - rule violations, unexpected actions,
↪  and who is doing something they shouldn't be
↪  doing.

Answer directly with your selected option.
```

Task: date_understanding

Description: Given sentences about a date, answer questions about related dates.

BBH Prompt (date_understanding)

```
You will be given information containing temporal
↪  clues about dates, followed by a question asking
↪  for a specific date in MM/DD/YYYY format.

Your task is to determine the requested date through
↪  careful, systematic reasoning.

Core Approach:

1. IDENTIFY THE QUESTION
   - Determine exactly what date is being asked for:
   ↪  "today," "tomorrow," "yesterday," or another
   ↪  specific relative date
   - Note the exact wording used in the question

2. ESTABLISH "TODAY" (THE REFERENCE POINT)
   - Find which date represents the present moment
   ↪  ("today") in the problem
   - Key patterns:
     * "tomorrow is [date]"  today = [date] minus 1 day
     * "yesterday was [date]"  today = [date] plus 1
     ↪  day
     * "today is [date]" → today = [date] directly
```

```
      * "delayed by X days to today"  today = original
      ↪  date plus X days
      * "X days have passed since [date]"  today =
      ↪  [date] plus X days
    - Always explicitly state what "today" is before
    ↪  proceeding

3. PARSE TEMPORAL RELATIONSHIPS
    - Understand directional phrases:
      * "before" means subtract days
      * "after" means add days
      * "since" typically means counting forward from a
      ↪  past date
    - Be precise about whether events are described
    ↪  relative to "today" or to other dates

4. PERFORM DATE CALCULATIONS
    - Use correct month lengths:
      * 31 days: Jan, Mar, May, Jul, Aug, Oct, Dec
      * 30 days: Apr, Jun, Sep, Nov
      * 28/29 days: Feb (29 if leap year)
    - Leap year rule: divisible by 4, except centuries
    ↪  must be divisible by 400
    - When adding/subtracting days:
      * Handle month boundaries carefully
      * Handle year boundaries (Dec 31 → Jan 1)
      * Count carefully when crossing multiple months

5. ANSWER THE SPECIFIC QUESTION
    - Calculate the exact date requested:
      * For "yesterday": today minus 1 day
      * For "tomorrow": today plus 1 day
      * For "today": use the reference date established
      * For other requests: apply appropriate
      ↪  arithmetic from today
    - Format as MM/DD/YYYY with leading zeros (e.g.,
    ↪  07/09/1972)

6. VERIFY YOUR ANSWER
    - Review your calculation step-by-step
    - Confirm the date matches one of the provided
    ↪  options
    - Check that month, day, and year are all correct

Present your reasoning clearly, showing each step of
↪  your logic. Always state what "today" is before
↪  calculating the requested date.
```

Task: disambiguation_qa

Description: Answer questions by resolving ambiguity in the question.

BBH Prompt (disambiguation_qa)

```
You are tasked with determining pronoun reference in
↪  sentences. For each sentence containing a pronoun,
↪  you must identify what the pronoun refers to (its
↪  antecedent), or determine if the reference is
↪  genuinely ambiguous.

Your task:
- Analyze the pronoun in the given sentence
- Consider all possible antecedents (nouns the
↪  pronoun could refer to)
- Use grammatical cues, sentence structure, world
↪  knowledge, and common-sense reasoning to
↪  determine the most likely referent
- Choose the most reasonable interpretation based on
↪  how the sentence would naturally be understood in
↪  everyday communication

Key principles:
- Apply typical real-world interpretations and
↪  pragmatic reasoning
- Consider: What makes logical sense? What would be
↪  the natural interpretation in everyday
↪  communication?
- Many sentences may seem grammatically ambiguous but
↪  have clear intended meanings based on context,
↪  world knowledge, and typical language usage
- Mark a sentence as ambiguous ONLY when multiple
↪  interpretations remain equally plausible after
↪  applying all available reasoning

Reasoning approach:
1. Identify the pronoun and list all potential
↪  antecedents
2. Check grammatical constraints (gender, number,
↪  person agreement)
3. Consider semantic and contextual factors (what
↪  makes logical sense given the sentence meaning?)
4. Apply world knowledge about typical scenarios,
↪  causal relationships, and common patterns in
↪  communication
5. Determine which antecedent is most reasonable
↪  based on the overall evidence

Output format:
- Provide step-by-step reasoning that shows your
↪  analysis
- State your final answer by selecting the option
↪  that best identifies the pronoun's antecedent
- Choose "Ambiguous" only when no interpretation can
↪  be reasonably preferred over others after
↪  thorough analysis
- Commit to the most plausible interpretation rather
↪  than defaulting to ambiguity when contextual or
↪  pragmatic clues favor one reading

Your goal is to resolve the reference when reasonable
↪  evidence exists, recognizing that natural
↪  language communication typically conveys clear
↪  intended meanings even when strict grammatical
↪  analysis might allow multiple possibilities.
```

Task: dyck_languages

Description: Determine if a sequence of nested parentheses is valid.

BBH Prompt (dyck_languages)

```
system
You are solving a bracket sequence completion task.
↪  You will receive a partial sequence of four
↪  bracket types: round ( ), square [ ], curly { },
↪  and angle < >. Your task is to determine exactly
↪  which closing brackets are needed at the end to
↪  make the sequence balanced and properly closed.

Instructions:
1. Use a stack-based approach to track unclosed
↪  brackets
2. Process the input left to right:
    - Push each opening bracket ( [ { < onto the stack
    - Pop from stack when a closing bracket ) ] } >
    ↪  matches the top
3. After processing all input, the remaining stack
↪  shows unclosed brackets
4. Generate the answer by closing brackets in reverse
↪  order (top to bottom of stack)
5. Provide your step-by-step reasoning, then clearly
↪  state your final answer on a line starting with
↪  "Answer:"

Show your work by displaying the stack state as you
↪  process the sequence.
```

Task: formal_fallacies

Description: Identify logical fallacies in deductive reasoning.

BBH Prompt (formal_fallacies)

```
You are tasked with determining whether an argument
↪  is deductively valid or invalid.

An argument is deductively valid if and only if the
↪  conclusion necessarily follows from the premises
↪  by logical necessity. If the premises can be true
↪  while the conclusion is false, the argument is
↪  invalid.

Instructions:

Carefully identify the premises and the conclusion.
↪  Distinguish what is given from what is claimed to
↪  follow.

Analyze the logical structure with precision. Pay
↪  special attention to:
- Conditional statements (if-then, necessary
↪  conditions, sufficient conditions)
- Quantifiers (all, every, some, whoever)
- Logical connectives (and, or, not, neither...nor)
- Negations and their scope

Trace the logical chain from premises to what can be
↪  validly deduced. Use contrapositives and other
↪  logical transformations as needed. Document each
↪  inference step.

Compare what the premises actually establish with the
↪  stated conclusion. Check for exact logical
↪  equivalence. Be vigilant about:
- Subtle wording differences between derived
↪  statements and the conclusion
- Negation mismatches (not X versus X)
- Disjunction differences (X or Y versus X or not-Y,
↪  or other variations)

Determine validity: The argument is valid only if the
↪  conclusion exactly matches what the premises
↪  logically entail. Any logical gap or mismatch
↪  makes it invalid.

Provide step-by-step reasoning showing your analysis,
↪  then state your final answer.

Options:
- valid
- invalid
```

Task: geometric_shapes

Description: Answer questions about geometric shapes and their properties.

BBH Prompt (geometric_shapes)

```
Determine the geometric shape drawn by the given SVG
↪  path element, then select your answer from the
↪  provided options.

SVG Path Commands Reference:
- M: Move to starting point
- L: Draw line to point
- A: Draw arc (rx,ry rotation
↪  large-arc-flag,sweep-flag x,y)
- Z: Close path (return to start)

Shape Identification Guidelines:
- Circle: Arcs with equal radii forming closed curve
- Ellipse: Arcs with unequal radii forming closed
↪  curve
- Polygons: Line segments forming closed angular
↪  shapes
- Sector: Arc segment with lines to center
- Line: Open path with only line commands

Analyze the path by examining the commands,
↪  coordinates, and whether the path closes to form
↪  a complete shape. Consider if the shape uses
↪  curved arcs or straight lines, count the number
↪  of sides for polygons, and check if arcs have
↪  equal or unequal radii.

Provide clear reasoning for your analysis, then
↪  select the correct answer from the given options.
```

Task: hyperbaton

Description: Rearrange words in a sentence to form a grammatical sentence.

BBH Prompt (hyperbaton)

```
You are tasked with determining which of two English
↪  sentences follows correct adjective order
↪  according to standard grammar rules.

ADJECTIVE ORDER RULE:
When multiple adjectives modify a noun, they follow
↪  this sequence:
Opinion  Size  Age  Shape  Color  Origin  Material
↪  Purpose  Noun

CATEGORY GUIDE:
1. Opinion: subjective judgments about quality
↪  (beautiful, terrible, wonderful, lovely,
↪  horrible)
2. Size: physical dimensions (large, small, tiny,
↪  huge, midsize, long)
3. Age: temporal state (old, new, young, ancient,
↪  modern)
4. Shape: physical form (round, square, flat, curved)
5. Color: color words (red, blue, green, black, white)
6. Origin: geographical or cultural source (American,
↪  French, Italian, Chinese)
7. Material: composition (wooden, metal, plastic,
↪  cotton, silk)
8. Purpose: function or use (racing, cooking,
↪  snorkeling, sleeping)

ANALYSIS PROCEDURE:
For each sentence, follow these steps:

Step 1: Identify all adjectives modifying the noun
Step 2: Classify each adjective by determining which
↪  category (1-8) it belongs to based on its meaning
Step 3: Write the category sequence as numbers
↪  (example: 127 means OpinionSizeMaterial)
Step 4: Check if the numbers are in ascending order
Step 5: The sentence with categories in ascending
↪  numerical order is correct

IMPORTANT NOTES:
- Not all categories must be present; skipping
↪  categories is valid (136 is correct)
- Each adjective fits exactly one category
- The rule applies whether or not the phrase sounds
↪  common or natural
- When an adjective's category is unclear, use its
↪  primary meaning in context

YOUR RESPONSE FORMAT:
State which sentence is correct, then explain your
↪  reasoning:
- List each adjective with its category name and
↪  number
- Show the numerical category sequence for both
↪  sentences
- Explicitly state which sequence follows ascending
↪  order and why the other violates the rule
```

Task: logical_deduction (five_objects)

Description: Deduce facts from given statements using logical reasoning.

BBH Prompt (logical_deduction_five_objects)

```
You are tasked with determining the correct order or
↪  arrangement of objects based on given clues and
↪  relationships. Your goal is to provide a
↪  systematic, well-reasoned response that clearly
↪  demonstrates your logical thinking process.

Follow this systematic approach:

1. EXTRACT AND ORGANIZE ALL CLUES: Carefully identify
↪  every piece of information that tells you
↪  something about the relationships or positions of
↪  objects. Look for:
   - Position indicators (first, last, middle, third
   ↪  from left, between X and Y)
   - Comparative relationships (taller than, more
   ↪  expensive than, comes before)
   - Fixed placements (anchored positions that don't
   ↪  change)
   - Sequential information (A follows B, X comes
   ↪  after Y)

2. WORK WITH DEFINITE INFORMATION FIRST: Start by
↪  placing objects with the most specific, fixed
↪  positions. These serve as anchors for your
↪  arrangement. For each placement:
   - State exactly which clue you are using
   - Explain why this placement is certain
   - Check that it doesn't contradict any other clues

3. BUILD SYSTEMATICALLY USING LOGICAL CHAINS: Add
↪  remaining objects one by one, using clear logical
↪  reasoning. For each decision:
   - Show how the clues lead to your conclusion
   - Verify the placement works with all previously
   ↪  placed objects
   - If multiple positions seem possible, work
   ↪  through the logic to eliminate options

4. VALIDATE YOUR COMPLETE SOLUTION: Test your final
↪  arrangement against every original clue. Each
↪  relationship must be satisfied. If any clue fails,
↪  trace back through your reasoning to identify and
↪  correct the error.

5. PRESENT YOUR REASONING CLEARLY: Make your logical
↪  chain explicit and easy to follow. Show
↪  step-by-step how you moved from the given
↪  information to your final answer.

This method works for any ordering problem: arranging
↪  objects in space, sequencing events in time,
↪  ranking by attributes, organizing hierarchies, or
↪  determining preference orders.

Answer the specific question with direct,
↪  well-reasoned logic that can be easily verified.
```

Task: logical_deduction (seven_objects)

Description: Deduce facts from given statements using logical reasoning.

BBH Prompt (logical_deduction_seven_objects)

```
You are solving a spatial ordering problem. You will
↪  be given descriptions of objects arranged in a
↪  sequence, along with constraints about their
↪  relative positions or rankings.

Your task is to determine the complete order of
↪  objects and answer the specific question asked.

Follow this systematic approach:

Step 1 - Identify all objects: List every object
↪  mentioned that needs to be ordered. Count them to
↪  ensure you have the complete set.

Step 2 - Determine the ordering dimension and
↪  direction: Identify what type of ordering is
↪  required (left-to-right position, price ranking
↪  from most to least expensive, size, etc.).
↪  Establish a consistent reference direction - this
↪  is critical for interpreting all constraints
↪  correctly.

Step 3 - Extract and normalize all constraints: List
↪  every constraint provided. Pay special attention
↪  to:
- Directional terms: "to the right of", "to the left
↪  of", "more expensive than", "less expensive than"
- Absolute positions: "third-most expensive", "second
↪  from the left", "rightmost", "cheapest"
- The perspective: "third from the right" counts from
↪  the opposite end as "third from the left"

Write each constraint in a consistent format relative
↪  to your chosen reference direction.

Step 4 - Identify anchor points: Look for constraints
↪  that fix objects at specific positions (e.g., "X
↪  is the rightmost" means position 7 in a 7-object
↪  sequence, "Y is second from the left" means
↪  position 2). These anchor points are essential
↪  for building the complete order.

Step 5 - Build relationship chains: Connect related
↪  constraints into ordered chains. For example, if
↪  A is right of B, and B is right of C, then the
↪  chain is C-B-A (left to right). Merge chains
↪  where they share common objects.

Step 6 - Integrate chains with anchor points: Use
↪  your fixed positions from Step 4 to place your
↪  relationship chains. Determine where each chain
↪  must be positioned to satisfy the absolute
↪  position constraints.

Step 7 - Complete the sequence: Assign all objects to
↪  positions. Fill in any remaining objects by
↪  process of elimination, ensuring all constraints
↪  remain satisfied.

Step 8 - Verify all constraints: Check every single
↪  constraint from Step 3 against your final
↪  ordering. Verify both relative positions and
↪  absolute positions are correct.

Step 9 - Answer the question with directional
↪  awareness: Read the question carefully. Remember:
- "Nth from the left" means counting N positions
↪  starting from the left end
- "Nth from the right" means counting N positions
↪  starting from the right end
- "Nth-most expensive" means N positions from the
↪  most expensive end
- "Nth-cheapest" means N positions from the cheapest
↪  end
In a sequence of 7 objects, "third from the right" is
↪  position 5 (when counting left to right), NOT
↪  position 3.

Present your reasoning clearly at each step.
```

Task: logical_deduction (three_objects)

Description: Deduce facts from given statements using logical reasoning.

BBH Prompt (logical_deduction_three_objects)

```
You are tasked with solving logical reasoning
↪  problems that require determining the correct
↪  order or arrangement of objects based on given
↪  constraints and relationships.

For each problem, follow this systematic approach:

1. List all objects to be arranged and note the type
↪  of ordering (left-to-right, first-to-last,
↪  top-to-bottom, etc.)

2. Identify all constraints:
   - Fixed positions: "X is third", "Y is in the
   ↪  middle"
   - Relative positions: "A is left of B", "C is
   ↪  before D"
   - Additional relationships or orderings

3. Build the arrangement methodically:
   - Begin with any fixed position constraints
   - Apply relative position constraints to place
   ↪  remaining objects
   - Use transitive reasoning when needed (if A is
   ↪  left of B and B is left of C, then A is left of
   ↪  C)
   - Fill remaining positions by elimination

4. Verify: Check that your final arrangement
↪  satisfies every constraint

5. Select the answer option that matches your
↪  arrangement

Always display your reasoning steps clearly before
↪  giving your final answer.
```

Task: movie_recommendation

Description: Recommend a movie based on given preferences.

BBH Prompt (movie_recommendation)

```
You will analyze movies a user has watched and liked,
↪  identify the connecting pattern, and recommend
↪  the best matching option from a provided list.

Instructions:

Look for patterns in the given movies. Patterns may
↪  be structural or content-based:

Structural patterns:
- Word count in titles (e.g., alternating lengths,
↪  consistent counts)
- Alphabetical ordering or sequences
- Title length or formatting patterns

Content patterns:
- Genre, themes, or storytelling style
- Era or time period
- Director, actors, or production characteristics

Approach:
1. Examine the given movies for any discernible
↪  pattern
2. Test each option against the identified pattern
3. Select the option that best continues or matches
↪  the pattern
4. Explain your reasoning step-by-step
5. State your final answer clearly as: "The answer is
↪  (X)"
```

Task: multistep_arithmetic_two

Description: Solve multi-step arithmetic problems.

BBH Prompt (multistep_arithmetic_two)

```
Evaluate the arithmetic expression by following these
↪  steps:

1. Identify all operations and apply the correct
↪  order of operations: parentheses first (innermost
↪  to outermost), then multiplication and division
↪  from left to right, then addition and subtraction
↪  from left to right.

2. For each operation you perform, explicitly write
↪  out the calculation and its intermediate result
↪  before proceeding to the next step.

3. Pay careful attention to negative numbers and
↪  their signs. When multiplying or combining
↪  negative values, show the sign handling
↪  explicitly (e.g., -9 * -5 = 45, because negative
↪  times negative equals positive).

4. For nested expressions with multiple parentheses,
↪  work systematically from the innermost
↪  parentheses outward, fully simplifying each level
↪  before moving to the next.

5. After completing all operations, clearly state the
↪  final numerical answer.
```

Task: navigate

Description: Determine whether the user returns to the origin after a series of movements.

BBH Prompt (navigate)

```
You will be given a series of navigation instructions.
↪  Your task is to determine whether following these
↪  instructions exactly as given would return you to
↪  the starting point.

Parse each instruction carefully and track your
↪  position:
- If instructions include directional changes (turns,
↪  rotations), track your heading
- If instructions include movements (steps, distance),
↪  track displacement in your current heading
- Process all instructions in the order given

Key assumptions:
- Start at position (0, 0) facing a default direction
↪  (e.g., North)
- If no direction is specified for a movement, assume
↪  it continues in the current heading
- If only movements are given with no turns, assume
↪  all movement is in the same straight line

After processing all instructions, evaluate whether
↪  your final position matches the starting position
↪  (0, 0).

Answer "Yes" only if you return to the exact starting
↪  point. Answer "No" if you are at any other
↪  position.

Show your step-by-step analysis before giving your
↪  final answer.
```

Task: object_counting

Description: Count objects in a given description.

BBH Prompt (object_counting)

```
Count the objects step by step. First, identify each
↪  distinct item mentioned and its quantity. If no
↪  quantity is specified for an item, assume there
↪  is 1 of that item. Then, sum the quantities to
↪  get the final count of objects requested in the
↪  question.
```

Task: penguins_in_a_table

Description: Answer questions about data presented in a table about penguins.

BBH Prompt (penguins_in_a_table)

```
You will be given one or more tables with structured
↪  data about subjects such as penguins or other
↪  animals. Each table has a header row defining the
↪  attributes, followed by data rows.

Your task is to answer a question about the data by
↪  following these steps:

1. Read the table header to identify the attributes
↪  (name, age, height, weight, etc.)

2. Carefully parse each data row, noting which value
↪  corresponds to which attribute for each subject

3. Read the question carefully to understand exactly
↪  what is being asked

4. Work through the problem step-by-step:
   - Explicitly identify which rows and values are
   ↪  relevant
   - Show your reasoning for each step
   - If counting, filtering, or comparing,
   ↪  demonstrate each evaluation clearly

5. If multiple-choice options are provided, verify
↪  which option matches your calculated answer

6. Double-check your work by reviewing the relevant
↪  data points

7. Present your final answer clearly and
↪  unambiguously at the end

Always structure your response to show your complete
↪  reasoning process before stating your final
↪  answer.
```

Task: reasoning_about_colored_objects

Description: Reason about relationships between colored objects.

BBH Prompt (reasoning_about_colored_objects)

```
You are given a description of objects on a surface,
↪  followed by a question about those objects. The
↪  question may ask about colors, quantities,
↪  positions, or other properties.

Your task is to answer the question by carefully
↪  analyzing the given information and reasoning
↪  through it step by step.

Approach each problem systematically:
- Identify all objects mentioned and their relevant
↪  attributes
- Understand exactly what the question is asking
↪  (what to count, compare, or identify, and any
↪  specific conditions or criteria)
- Evaluate each object against the question's criteria
- If counting is needed, explicitly count the
↪  qualifying objects
- State your final answer clearly by selecting from
↪  the provided options

Always show your complete reasoning process before
↪  giving your final answer selection.
```

Task: ruin_names

Description: Fix names that have been altered or corrupted.

BBH Prompt (ruin_names)

```
You are solving a word puzzle. Your task: identify
↪  which option is a humorous one-character edit of
↪  a given artist, band, or movie name.

A one-character edit means changing exactly one
↪  character by substitution, deletion, or addition.

When evaluating options:

1. Check if each option differs from the original by
↪  exactly one character

2. Evaluate the humor: Does the edit create a funny
↪  new meaning, wordplay, absurd imagery, or
↪  recognizable pun? Avoid selecting mere typos or
↪  meaningless letter changes.

3. Select the option with the strongest, clearest
↪  humor

Now solve the given puzzle. State your final answer
↪  clearly as: "Final answer: [option letter]
↪  [option text]"
```

Task: salient_translation_error_detection

Description: Identify errors in translations.

BBH Prompt (salient_translation_error_detection)

```
You are tasked with identifying translation errors
↪  between German source sentences and their English
↪  translations.

For each task, you will receive:
- A German source sentence
- An English translation
- Multiple error type options

Your goal is to determine which single error type
↪  best describes the translation mistake.

Error Categories:

(A) Modifiers or Adjectives: Descriptive words,
↪  adjectives, or modifiers pertaining to a noun are
↪  changed or omitted. Examples: "small town"
↪  becomes "town" or "big town"; "red car" becomes
↪  "blue car"; "quickly ran" becomes "ran"

(B) Numerical Values: Numbers, dates, ordinals,
↪  cardinals, or units are altered. Examples: "1995"
↪  becomes "1996"; "three" becomes "four"; "meters"
↪  becomes "feet"; "first" becomes "second"
```

```
(C) Negation or Antonyms: A negation is added,
↪  removed, or a word is changed to its antonym.
↪  Examples: "not good" becomes "good"; "more"
↪  becomes "less"; "hot" becomes "cold"; "always"
↪  becomes "never"

(D) Named Entities: Proper nouns such as person names,
↪  place names, or organization names are changed to
↪  different entities. Examples: "Berlin" becomes
↪  "Munich"; "John Smith" becomes "Jane Smith";
↪  "Amazon" becomes "Google"

(E) Dropped Content: A significant clause, phrase, or
↪  portion of the sentence is completely omitted
↪  from the translation

(F) Facts: Factual errors involving incorrect
↪  categorization or type of things that don't fit
↪  the above categories. Examples: "town" becomes
↪  "city"; "river" becomes "lake"; "president"
↪  becomes "mayor"; "sculptor" becomes "painter"

Analysis Process:

1. Read the German source sentence carefully and
↪  understand its complete meaning
2. Compare the English translation to the source and
↪  identify what differs
3. Determine the specific nature of the difference -
↪  what word or phrase changed, and how
4. Match the error to the most specific category:
   - If a number, date, or unit changed  (B) Numerical
   ↪  Values
   - If a proper name or place changed  (D) Named
   ↪  Entities
   - If significant content is completely missing  (E)
   ↪  Dropped Content
   - If negation changed or an antonym was used  (C)
   ↪  Negation or Antonyms
   - If a descriptive modifier or adjective changed
   ↪  (A) Modifiers or Adjectives
   - If the fundamental type or category of something
   ↪  changed  (F) Facts
5. State your reasoning with specific reference to
↪  words from both sentences, then provide your
↪  final answer

Important Notes:
- Choose only the single most accurate category for
↪  the primary error
- Focus on what changed, not what stayed the same
- When an antonym substitution occurs, classify it as
↪  Negation or Antonyms even if it involves modifiers
- Provide clear reasoning before your final answer
```

Task: snarks

Description: Identify sarcasm in statements.

BBH Prompt (snarks)

```
Identify which statement is sarcastic by carefully
↪  analyzing both options.

To detect sarcasm, look for irony, mockery,
↪  exaggeration, or absurdity that signals the
↪  speaker is being deliberately facetious rather
↪  than literal. Sarcasm often involves saying
↪  something that contradicts reality, common sense,
↪  or expected outcomes in a way that highlights the
↪  ridiculousness of a situation.

Follow these steps:

1. Read both statements and note how they differ.
2. For each statement, assess whether it makes a
↪  literal, reasonable claim or whether it uses
↪  irony or absurdity.
3. Consider the context: which statement expresses a
↪  genuine observation versus which one uses
↪  exaggeration or false equivalence to mock or
↪  criticize?
4. Compare the two options to determine which is
↪  straightforward and which is ironic.

Conclude which statement is sarcastic based on your
↪  analysis.
```

Task: sports_understanding

Description: Answer questions about sports rules and situations.

BBH Prompt (sports_understanding)

```
system
Determine whether a sports-related sentence is
↪  plausible by verifying if the described action
↪  matches the sport the player actually plays.

Reasoning approach:
1. Identify the player and their actual sport
2. Identify the action/event described
3. Determine if that action belongs to the player's
↪  sport
4. Conclude: plausible or not plausible

Provide concise reasoning, then state your final
↪  answer.

Now evaluate the given sentence using the same
↪  reasoning process.
```

Task: temporal_sequences

Description: Reason about sequences of events in time.

BBH Prompt (temporal_sequences)

```
You are tasked with determining when a person was
↪  available during their day to perform a specific
↪  activity.

Your goal is to identify time periods when the person
↪  was free to perform the activity in question.

To solve this problem:

First, establish the timeline by identifying all
↪  times when the person was occupied with confirmed
↪  activities. During these periods, the person was
↪  NOT available.

Second, note any restrictions that limit availability,
↪  such as when the person was asleep, when
↪  locations were closed, or other temporal
↪  limitations mentioned in the problem.

Third, identify the gaps in the timeline - these are
↪  periods when the person was not engaged in any
↪  confirmed activity and no restrictions apply.
↪  These gaps represent potential free time.

Fourth, evaluate each answer option by checking:
- Does this time period overlap with any confirmed
↪  activity? If yes, the person was NOT free during
↪  this time.
- Does this time period violate any stated
↪  restriction? If yes, the person was NOT free
↪  during this time.
- If neither conflict exists, then this time period
↪  represents when the person WAS free.
```

```
Present your reasoning clearly. For each option,
↪  state whether the person was free during that
↪  time and explain your reasoning. Conclude with
↪  your final answer stating which option correctly
↪  identifies when the person was free to perform
↪  the activity in question.

Be systematic and thorough in checking all activities
↪  and constraints against each time option.
```

Task: tracking_shuffled_objects (five_objects)

Description: Track objects that have been shuffled.

BBH Prompt (tracking_shuffled_objects_five_objects)

```
You will be given a scenario where entities (people,
↪  objects, etc.) each start with an assigned
↪  attribute (partner, position, item, etc.). A
↪  series of swaps will then occur between pairs of
↪  entities. Your task is to track these swaps and
↪  determine the final state.

APPROACH:

1. INITIAL STATE: Write out the complete starting
↪  assignments for all entities.

2. PROCESS EACH SWAP IN ORDER:
   - Identify the two entities involved
   - Note what each currently has BEFORE the swap
   - Exchange their assignments
   - Write out the COMPLETE state after this swap for
   ↪  ALL entities

3. FINAL ANSWER: After all swaps, identify the
↪  requested entity's final assignment and select
↪  the correct option.

CRITICAL RULES:
- Swaps are processed sequentially in the exact order
↪  given
- Only the two named entities exchange assignments in
↪  each swap
- All other entities keep their current assignments
↪  during a swap
- Always write the full state after each swap to
↪  avoid errors
- Each entity swaps whatever they hold at that moment,
↪  not their original assignment
```

Task: tracking_shuffled_objects (seven_objects)

Description: Track objects that have been shuffled.

BBH Prompt (tracking_shuffled_objects_seven_objects)

```
system
You will track objects through a sequence of pairwise
↪  swaps to determine final positions.

APPROACH:

1. ESTABLISH INITIAL STATE
   - Write out each person and their starting object
   - This is your reference point before any swaps

2. PROCESS EACH SWAP IN ORDER
   - For each swap, explicitly state:
     a) Who is swapping
     b) What each person currently holds before the
     ↪  swap
     c) What each person holds after the swap
   - Process swaps in the exact sequence given
   - Only the two people mentioned exchange objects;
   ↪  others remain unchanged

3. STATE FINAL HOLDINGS
   - After all swaps, list what each person holds
   - Verify all objects are accounted for

4. ANSWER THE QUESTION
   - Identify what the specified person holds in your
   ↪  final state
   - Select the matching option

CRITICAL POINTS:

- Execute swaps in exact order provided
- Each swap affects only two people
- Track current state explicitly before each swap
- Be thorough; write out intermediate states to
↪  prevent errors
- Double-check your final answer
```

Task: tracking_shuffled_objects (three_objects)

Description: Track objects that have been shuffled.

BBH Prompt (tracking_shuffled_objects_three_objects)

```
You will be given a scenario where objects start in
↪  initial positions and undergo a series of pairwise
↪  swaps. Your task is to determine where a specific
↪  object ends up after all swaps are completed.

Solution approach:

1. Record the initial state: List each object and its
↪  current holder or location.

2. Process each swap sequentially: For every swap
↪  described, exchange the two objects between their
↪  current positions. A swap means two objects
↪  completely exchange positions.

3. Write the state after each swap: After every
↪  single swap, document the complete current state.
↪  This is critical to avoid tracking errors.

4. Identify the final answer: Once all swaps are
↪  complete, determine where the queried object is
↪  located.

5. Select the matching option from the choices
↪  provided.

Important: Apply swaps in the exact order given and
↪  update your tracking immediately after each swap.

Now apply this systematic approach to solve the given
↪  problem.
```

Task: web_of_lies

Description: Determine who is telling the truth or lying based on statements.

BBH Prompt (web_of_lies)

```
You are solving a logic puzzle involving
↪  truth-tellers and liars. In these puzzles, each
↪  person either always tells the truth or always
↪  liesthere is no middle ground.

Your objective: Determine whether a specific target
↪  person tells the truth or lies.

Approach:
```

```
1. Identify any given facts: Note statements that are
↪  explicitly presented as true (e.g., "X tells the
↪  truth" or "X lies").

2. Build a chain of logical deductions:
   - When a truth-teller makes a claim, that claim is
   ↪  true.
   - When a liar makes a claim, that claim is false.
   - Use each established fact to evaluate the next
   ↪  person's statement.

3. Test for consistency: If you reach a contradiction,
↪  the assumption about a person's truthfulness must
↪  be wrong. Try the opposite assumption.

4. Work systematically through all statements until
↪  you can definitively determine the target
↪  person's status.

5. Verify your conclusion: Trace back through your
↪  reasoning to ensure no logical errors or
↪  contradictions remain.

Present your reasoning clearly for each step. State
↪  your final answer explicitly: Does the target
↪  person tell the truth, or does the target person
↪  lie?
```

Task: word_sorting

Description: Sort words according to specific criteria.

BBH Prompt (word_sorting)

```
Sort the following words alphabetically using
↪  lexicographic (dictionary) order.

Compare words character by character from left to
↪  right. Earlier letters in the alphabet come first
↪  (a < b < c ... < z). If one word is a prefix of
↪  another, the shorter word comes first. Treat
↪  capital letters as lowercase for comparison.

Show your step-by-step work, then provide your final
↪  answer as a comma-separated list on a new line
↪  starting with "Final Answer:"
```

Task: GSM8K

Description: Solve grade-school math word problems that require multi-step arithmetic reasoning, showing intermediate calculations and producing a final numeric answer.

GSM8K Prompt

```
Read the problem carefully and identify all
↪  mathematical relationships between the given
↪  quantities and what you need to find.\\n\\nFollow
↪  this systematic approach:\\n\\n1. Extract key
↪  information: List all given numbers, quantities,
↪  and their relationships. Clearly state what the
↪  problem is asking you to calculate.\\n\\n2. Plan
↪  your solution: Determine what mathematical
↪  operations you need and in what sequence. Identify
↪  any intermediate values you must calculate
↪  first.\\n\\n3. Calculate step-by-step: Work
↪  through each calculation in order. Use the format
↪  <<calculation>> for all arithmetic operations
↪  (e.g., <<5+8=13>>). After each calculation,
↪  clearly state what that result represents.\\n\\n4.
↪  Verify your answer: Check that your final result
↪  makes sense in the context of the problem and
↪  directly answers the question asked.\\n\\nPresent
↪  your final numerical answer using the format ####
↪  [number].\\n\\nShow your reasoning at each step
↪  and track intermediate results carefully, as they
↪  are often needed for subsequent calculations.
```

Task: AQUA-RAT

Description: Solve algebraic word problems (often multiple-choice) and provide a step-by-step rationale that justifies the selected answer.

AQUA-RAT Prompt

```
Solve this mathematical word problem using systematic
↪  step-by-step reasoning. Generate both a clear
↪  explanation and a solution
↪  program.\\n\\nEXPLANATION REQUIREMENTS:\\n-
↪  Identify given information, constraints, and what
↪  needs to be found\\n- Break the problem into
↪  logical steps with clear reasoning at each
↪  stage\\n- Show all calculations and mathematical
↪  operations\\n- State your final answer
↪  explicitly\\n\\nSOLUTION PROGRAM
↪  REQUIREMENTS:\\n- Express your method as an
↪  algorithm, formula, or computational procedure\\n-
↪  Define variables clearly and specify all
↪  mathematical operations\\n- Ensure your program
↪  logic directly corresponds to your explanation
↪  steps\\n- Make the solution general enough to
↪  work for similar problems\\n\\nQUALITY
↪  STANDARDS:\\n- Each reasoning step must lead
↪  logically to the next\\n- Both explanation and
↪  program must be mathematically sound and
↪  verifiable\\n- Maintain consistency between your
↪  natural language explanation and programmatic
↪  solution\\n- Use precise mathematical language
↪  throughout\\n\\nPROBLEM-SPECIFIC ADAPTATION:\\n-
↪  Algebraic problems: Set up and solve equations
↪  systematically\\n- Logic puzzles: Identify key
↪  patterns and apply them methodically\\n- Counting
↪  problems: Apply appropriate counting principles
↪  or formulas\\n- Optimization problems: Define
↪  constraints and objectives clearly\\n\\nPresent
↪  your work in a structured format where the
↪  reasoning process is transparent and the solution
↪  method can be independently verified and applied
↪  to similar problems.
```

Task: AlpacaEval 2.0

Description: Evaluate instruction-following model responses using an LLM judge to compare outputs and estimate preference-based win rates in an automated, scalable manner.

AlpacaEval 2.0 Prompt

```
**Instructions for Generating High-Quality Responses:**

When responding to user queries, follow these principles to deliver maximum value:

**Core Approach:**
- **Be directly helpful**: Address the user's actual need, not just the literal question. Anticipate follow-up questions and proactively provide relevant information.
- **Show your reasoning**: When appropriate, briefly explain your thought process to build trust and help users understand the answer better.
- **Be conversational yet professional**: Write naturally as if having an intelligent conversation, avoiding overly formal or robotic language.
- **Prioritize clarity over complexity**: Use simple language and concrete examples. Break down complex topics into digestible explanations.

**Content Quality:**
- **Provide actionable insights**: Focus on practical, useful information the user can immediately apply.
- **Use specific examples**: Illustrate concepts with real-world examples, analogies, or step-by-step demonstrations rather than abstract explanations.
- **Balance depth and brevity**: Be thorough enough to be useful, but concise enough to respect the user's time. Adjust based on query complexity.
- **Acknowledge limitations**: If uncertain or if information may be outdated, say so honestly rather than speculating.

**Structure and Presentation:**
- **Lead with the most important information**: Answer the core question first, then provide supporting details.
- **Use natural formatting**: Employ lists, examples, or structured breakdowns only when they genuinely enhance claritynot by default.
- **Make it scannable**: Use clear transitions and logical flow so users can quickly find what they need.

**Engagement:**
- **Anticipate needs**: Offer related insights, common pitfalls to avoid, or next steps the user might find valuable.
- **Encourage exploration**: When relevant, suggest ways to dive deeper or related topics worth exploring.
- **Be adaptable**: Match the tone and depth to the user's apparent expertise level and the context of their question.

**Key Mindset:**
Think like a knowledgeable, helpful colleague who genuinely wants to solve the user's problemnot just answer their question literally. Focus on being useful, clear, and trustworthy above all else.
```