Don't Generate, Classify! Low-Latency Prompt Optimization with Structured Complementary Prompt

Hee-Soo Kim[1], **Jun-Young Kim**[2], **Jeong-Hwan Lee**[1],
Seong-Jin Park[1], **Kang-Min Kim**[1,3]*
[1]Department of Artificial Intelligence
[2]Department of Computer Science and Information Engineering
[3]Department of Data Science
The Catholic University of Korea, Bucheon, Republic of Korea
{heesu1231,youngkim1006,jeonghwanlee,sjpark,kangmin89}@catholic.ac.kr

Abstract

Large language models (LLMs) have demonstrated strong performance across diverse natural language processing tasks. However, their performance varies significantly across different prompts, requiring careful engineering for consistent results. Manual prompt engineering requires substantial human effort and suffers from limited reproducibility. In contrast, automatic prompt optimization methods reduce manual effort but often depend on costly autoregressive generation, resulting in substantial latency overheads. To address these limitations, we present low-latency prompt optimization (LLPO), a novel framework that reframes prompt engineering as a classification problem. LLPO classifies structured prompt fields from user input through multi-task classification and populates a predefined template to generate an optimized system prompt with minimal latency. In LLM-based automatic evaluations across four question-answering benchmarks, LLPO improves answer quality by up to 26.5% in Δwin rate compared to prior automatic prompt optimization methods, while reducing latency by up to 1,956 times. Human evaluation shows that LLPO receives the highest proportion of top-ranked responses. Furthermore, we analyze the contribution of each structured prompt field to performance, highlighting the robustness of our framework. GitHub Repository

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities, achieving state-of-the-art performance across diverse benchmarks (Brown et al., 2020; Grattafiori et al., 2024; Zhang et al., 2022; Team et al., 2025; Yang et al., 2025). To leverage this potential, researchers have explored manual prompt engineering techniques such as in-context learning (ICL) (Brown et al., 2020) and chain-of-thought (CoT) prompting (Kojima et al., 2022; Wei et al., 2022). While these approaches improve LLM performance on specific tasks, they often lack explicit mechanisms for aligning outputs with user preferences. Recent studies have emphasized the importance of explicit prompt components (Boonstra, 2025; Bsharat et al., 2023), such as specifying the model's persona, tone, and response style. These structured elements enable LLMs to generate more context-aware and user-adapted responses. However, crafting such prompts requires considerable human expertise and involves extensive trial and error.

Automatic prompt engineering has emerged to reduce manual effort and improve robustness across tasks. Early works explored reinforcement learning-based approaches (Zhang et al., 2023; Deng et al., 2022) and gradient-free optimization methods (Pryzant et al., 2023; Chen et al., 2024; Yang et al., 2024; Guo et al., 2024). While these approaches can be effective, they often require extensive exploration during optimization, leading to high computational costs. To mitigate this overhead, recent methods such as black-box prompt optimization (BPO) (Cheng et al., 2024), plug-and-play augmentation system (PAS) (Zheng et al., 2024), and free-form instruction-oriented prompt optimization (FIPO) (Lu et al., 2025) leverage offline-trained prompt optimizers that can be deployed at inference time, reducing the need for online exploration. However, these methods still rely on autoregressive LLM generation to optimize prompts, introducing substantial latency overhead at inference time.

To address the limitations of existing prompt optimization methods, we introduce low-latency prompt optimization (LLPO), which employs a transformer encoder-based pre-trained language model (PLM) (Liu et al., 2019) for efficient prompt optimization. LLPO predicts structured prompt fields through multi-task classification and populates a predefined template to construct an opti-

*Corresponding author.

Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics
Volume 1: Long Papers, pages 4364–4383

mized system prompt. Since the PLM handles the entire optimization process, LLMs are only required at inference time for final response generation, significantly reducing latency and computational overhead.

Central to LLPO is the structured complementary prompt (SCP), a schema that decomposes prompts into eight key fields: role, audience, user intent, tone type, constraints, reasoning guidance, output format, and interactive mode. To construct SCP annotations at scale, we leverage examples from the BPO dataset[1], which includes original prompts, preferred and dispreferred responses, and optimized prompts. Using these rich supervision signals, we employ an LLM to extract SCP fields from the original BPO prompts. We then select a subset of the resulting annotations as few-shot exemplars to infer SCP for raw user instructions in the PAS dataset[2]. This two-stage approach ensures high-quality SCP construction while minimizing manual effort and maximizing dataset coverage.

We evaluate LLPO using automatic LLM-based comparisons and find that it consistently achieves strong performance with significantly lower latency. Across five LLMs, LLPO improves win rates by 15.1% on average compared to original prompts and outperforms BPO by 8.6%. Given that recent studies have highlighted potential biases in LLM-based automatic evaluation methods (Zheng et al., 2023; Liu et al., 2024; Doostmohammadi et al., 2024), we further validate our findings through human preference evaluations. LLPO receives the highest proportion of top-ranked outputs compared to PAS and FIPO, confirming its effectiveness from both automated and human evaluation perspectives.

In summary, our contributions are threefold:

- We propose LLPO, a novel framework that reframes prompt optimization as a classification problem, enabling low-latency optimization for real-time applications.
- We introduce the SCP formulation and employ a multi-task classification approach with PLMs to predict SCP fields efficiently.
- We demonstrate through comprehensive evaluation that LLPO matches the prompt quality compared to existing methods while reducing optimization latency by up to 1,956 times, with effectiveness confirmed by both automatic and human evaluations.

[1] https://huggingface.co/datasets/zai-org/BPO

[2] https://github.com/PKU-Baichuan-MLSystemLab/PAS/tree/main

2 Related Works

2.1 Manual Prompt Engineering

Manual prompt engineering has become an effective approach for adapting LLMs to new tasks without parameter updates. ICL (Brown et al., 2020) demonstrated that providing a few examples can guide LLMs toward novel task behaviors. Building on this foundation, various prompting techniques have been proposed to enhance reasoning and generalization, including pattern-exploiting training (Schick and Schütze, 2022), prompt programming (Reynolds and McDonell, 2021), and CoT prompting (Wei et al., 2022). Notably, even simple cues such as "Let's think step by step" can improve performance on reasoning tasks (Kojima et al., 2022).

The success of these techniques has motivated efforts to systematize prompt design. For instance, recent guidelines emphasize structured components such as role definition, contextual information, instruction clarity, and output format specification (Boonstra, 2025). Similarly, Bsharat et al. (2023) identified 26 design principles that capture common structural and interaction patterns. These developments reflect a shift from ad hoc experimentation toward more principled and reproducible prompt engineering practices.

Motivated by these insights, we propose the SCP, an eight-field schema that formalizes and integrates established best practices from manual prompt engineering. Unlike prior work that relies on human expertise for prompt design, SCP enables scalable and automatic prompt generation through structured field prediction, retaining the quality benefits of manual design while significantly reducing manual effort.

2.2 Automatic Prompt Engineering

Automatic prompt engineering has explored various gradient-free and black-box approaches to optimize prompts for LLMs. Early discrete prompt optimization methods include AutoPrompt (Shin et al., 2020), which leverages gradient-guided trigger token selection, and APE (Zhou et al., 2023), which frames instruction generation as a black-box search problem guided by LLM proposals. Reinforcement learning-based methods have been proposed for discrete prompt optimization using model feedback. RLPrompt (Deng et al., 2022) learns a

Field	What it controls	Example values
Role	Speaker persona / expertise	'software engineering consultant', 'sports industry analyst', 'film studies expert', 'psychological consultant'
Audience	Target reader	'general readers seeking understanding', 'novice programmers and students', 'prospective pet owners'
User Intent	User's underlying goal	'information-seeking', 'problem-solving and analysis', 'creative writing, analysis and understanding'
Tone Type	Stylistic register	'clear and concise', 'inspirational and thoughtful', 'dramatic and introspective', 'immersive and descriptive'
Constraints	Content / formatting restrictions	'maintain factual accuracy, and avoid misinformation', 'avoid assumptions about intent, promote safe practices'
Reasoning Guidance	Expected reasoning style	'step-by-step explanation and analysis', 'chain-of-thought with detailed reasoning', 'evidence-citation style'
Output Format	Surfaces structure of output	'summary with examples and suggestions', 'structured sections with summary paragraphs and detailed analysis'
Interactive Mode	Clarification-question policy	'not allowed', 'allowed at the end of response for clarification'

Table 1: Description and examples of the eight SCP fields used for prompt optimization.

prompt editing policy through reinforcement learning, while TEMPERA (Zhang et al., 2023) refines prompts via token-level actions during inference. ProTeGi (Pryzant et al., 2023) takes a different approach by guiding discrete prompt edits using textual gradient approximations and beam search without requiring gradient access.

Another line of work treats LLMs themselves as optimizers. InstructZero (Chen et al., 2024), OPRO (Yang et al., 2024), and EvoPrompt (Guo et al., 2024) leverage LLMs to generate, mutate, and evaluate prompt candidates in iterative optimization loops, demonstrating strong performance in black-box settings. More recently, offline-trained prompt optimizers have emerged, including BPO, FIPO, and PAS, which learn from human preference data or LLM-generated responses. BPO and FIPO learn to rewrite prompts to better align with human-preferred responses using LLM-generated feedback from preference datasets, while PAS enhances prompts through a plug-and-play system trained on automatically generated complementary prompts. These approaches represent a shift toward more interpretable, data-efficient, and transferable prompt engineering techniques applicable to both open- and closed-source LLMs.

Despite their effectiveness, these methods rely heavily on autoregressive LLM generation and require substantial inference time and computational resources, making real-time deployment challenging. In contrast, our approach introduces a lightweight PLM-based classifier that predicts structured prompt components through multi-task classification. By eliminating the need for autoregressive generation during optimization, LLPO enables fast and efficient prompt optimization suitable for latency-sensitive applications.

3 Method

Our method generates an optimized system prompt for each user prompt using a lightweight PLM, achieving comparable alignment benefits with significantly lower computational cost. At the core of our approach is the SCP, a set of eight categorical fields shown to effectively guide LLM behavior.

3.1 Constructing SCP

Building on the 26 principles for instruction design (Bsharat et al., 2023) and Google's prompt-engineering guide (Boonstra, 2025), we distill prompt design into eight fields that capture who is speaking, to whom, for what purpose, and how the answer should be delivered. Table 1 lists the eight SCP fields and illustrates typical values. We denote these structured prompt elements as X_{SCP}, and the original user query as X_{user}. A detailed explanation of the eight SCP fields is provided in Appendix A.

3.1.1 Deriving SCP Labels in Two Data Types

BPO constructs optimized prompts using preference-labeled data, while PAS generates prompts directly from raw user inputs without human annotations. To leverage both paradigms, we generate $(X_{\text{user}}, X_{\text{SCP}})$ pairs from two data sources: the BPO dataset, which includes human preference signals, and the PAS dataset, which contains only raw instructions. We devise a two-stage labeling pipeline to construct SCP annotations for both datasets using GPT-4o[3]. For the BPO dataset, we employ a preference-aware procedure that incorporates preference signals. For the PAS dataset, we use a 10-shot inference approach.

Dataset with Human-Preference Each instance of BPO dataset supplies $(X_{\text{user}}, Y_{\text{good}}, Y_{\text{bad}}, X_{\text{opt}})$, where X_{opt} is optimized prompt automatically derived from X_{user} by contrasting preferred output Y_{good} with dispreferred output Y_{bad}. As illustrated in the left panel of Step 1 (a) in Figure 1, we feed these items to the LLM and instruct it to:

1. **Preference response comparison** – qualitatively contrast Y_{good} and Y_{bad} with respect to

[3] https://platform.openai.com/docs/models/gpt-4o

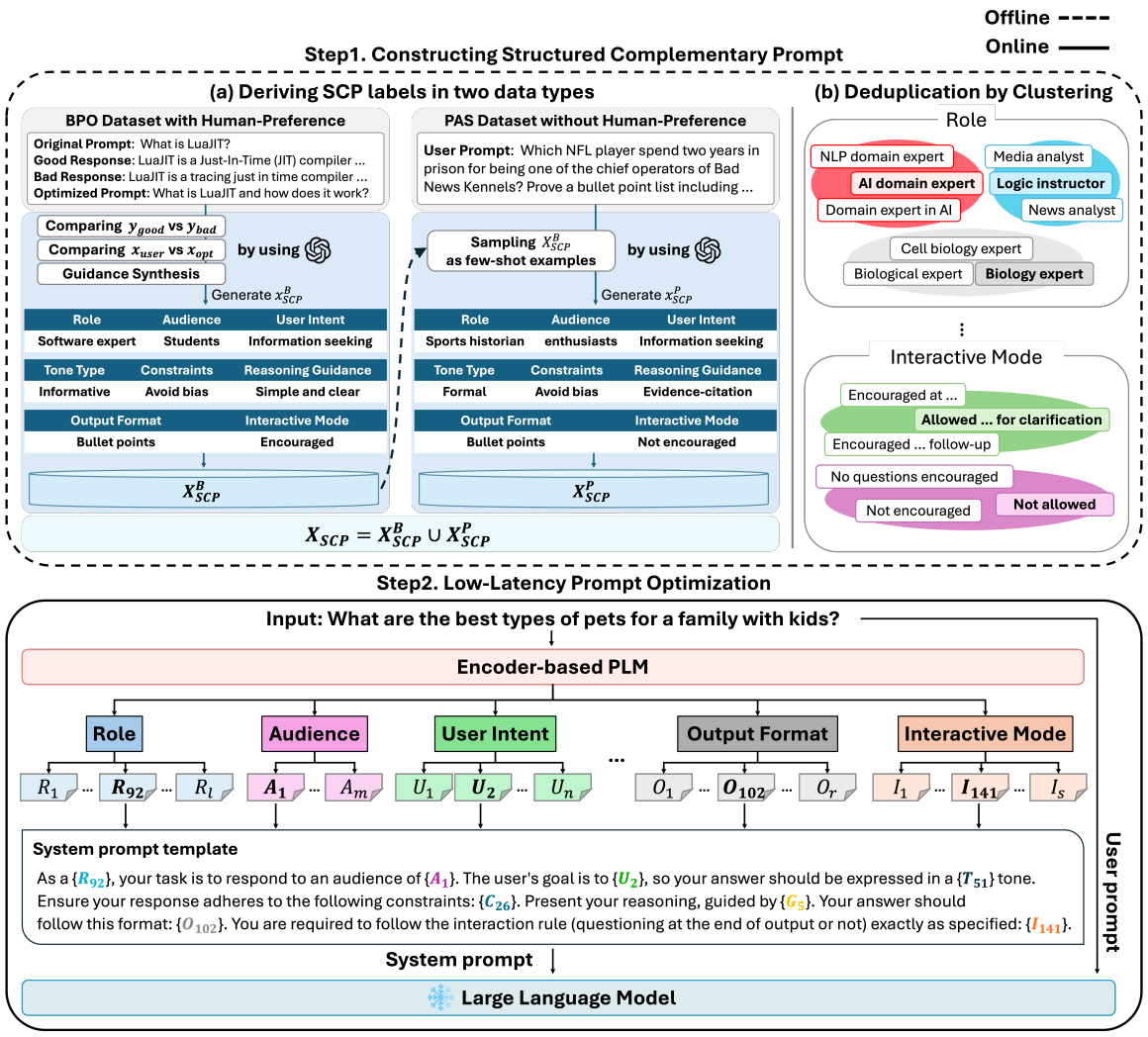

Figure 1: Overview of the LLPO architecture and prompt optimization pipeline.

correctness, helpfulness, and harmlessness.

2. **Prompt comparison** – analyze the differences between X_{user} and X_{opt}.

3. **Guideline synthesis** – from steps 1–2, distill the complementary instructions that would steer the model from Y_{bad} to Y_{good}.

LLM then converts these analyses into concrete values for the eight SCP attributes (i.e., *Role, Audience, ..., Interactive Mode*), and we store the resulting pair $(X_{\text{user}}, X^B_{\text{SCP}})$ as a gold example.

Dataset without Human-Preference PAS highlights that prompt optimization is feasible without human-preference supervision. To incorporate this into our framework, we sample a few gold pairs as exemplars and prompt the LLM with only X_{user} to generate X^P_{SCP}, enabling broader coverage without additional human effort (Step 1 (a), right section in Figure 1). We then combine X^B_{SCP} and X^P_{SCP} to construct the final SCP label set X_{SCP}, leveraging both high-quality annotations and scalable few-shot expansions.

3.1.2 Pruning Long-Tailed Labels via Clustering

A side-effect of generating field values with LLM is an open-ended label space: even semantically identical values (e.g., "formal tone", "formal style") emerge as distinct texts (Step 1 (b), in Figure 1). This long-tailed label distribution inflates the number of classes per field and reduces data efficiency. To standardize the label space, we first embed every raw SCP value with MiNiLM-L6 (384d)[4] to obtain a sentence embedding, then cluster the embeddings using both K-means clustering (MacQueen, 1967) and agglomerative clustering (Murtagh and Legen-

[4] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

dre, 2014). After the SCP clustering process, we assign a representative label (e.g., "formal tone") by selecting the value whose embedding lies closest to the centroid, and all other labels in that cluster are replaced with this representative label. Details on how the number of clusters was chosen and how the representative labels were determined are provided in the Appendix B.

3.2 Low-Latency Prompt Optimization

3.2.1 Multi-task Classification of SCP Fields

Based on the constructed SCP dataset, we train a transformer encoder-based PLM to predict all eight SCP fields via multi-task classification. Each X_{user} is tokenized and passed through the PLM to obtain a pooled representation r. This representation is fed into eight independent classification heads, one for each SCP field k, each producing a probability distribution p_k over its label space:

$$p_k = softmax(W_k r + b_k), \tag{1}$$

where W_k and b_k are the parameters of the k-th classification head.

To handle class imbalance across fields, we adopt the focal loss (Lin et al., 2017). Let y_k be the ground-truth label for field k and $p_k(y_k)$ the model's predicted probability for that label. We define the loss function as:

$$\mathcal{L} = \sum_{k=1}^{|SCP|} (1 - p_k(y_k))^{\gamma}(-\log p_k(y_k)), \tag{2}$$

where $|SCP|$ is the number of the SCP fields (e.g., eight in this method). This loss function emphasizes harder examples by down-weighting those that are confidently classified.

3.2.2 Integrating SCP into LLM Prompts for Inference

At inference time, as illustrated in Figure 1 (Step 2), the trained PLM takes a raw user instruction and predicts a SCP instance, consisting of eight structured attributes. These predicted values are then inserted into a fixed template to construct the optimized system prompt. The template is designed to guide the downstream LLM by explicitly specifying all eight structured fields in the SCP. The resulting system prompt is provided as the system message to the LLM, while the original instruction is used directly as the user message. The LLM then generates a response conditioned on both inputs. Because the system prompt explicitly communicates the intended reasoning process, persona, stylistic tone, and interaction policy, it enables more aligned and controllable generation without any parameter updates to the LLM.

4 Experiments

4.1 Experimental Details

Test Datasets To obtain a balanced view of our model's instruction-following alignment, we evaluate on four public benchmarks that vary in topic breadth and difficulty: Koala Eval (Geng et al., 2023), Self-Instruct Eval (Wang et al., 2023), Dolly Eval (Conover et al., 2023), and Arena-Hard Eval (Li et al., 2024). Detailed descriptions are provided in Appendix C.1.

Metrics Building on prior work demonstrating that strong LLMs, such as GPT-4[5], can reliably approximate human judgment in response evaluation tasks (Zheng et al., 2023; Wang et al., 2024), we adopt a pairwise LLM-as-a-judge protocol using GPT-4o. We follow the MT-Bench prompt format (Zheng et al., 2023) (Figure 5 in Appendix C.2) and set the temperature to 0 for consistency. To reduce positional bias, the two candidate responses are presented in randomized order for each evaluation.

Baselines We compare LLPO with five representative baselines from prior work: original prompt (ori.), zero-shot CoT (CoT), BPO[6], PAS[7], and FIPO[8]. All methods are evaluated using the same set of LLMs for inference. See Appendix C.3 for the full list and naming conventions of the base LLMs used in our experiments.

Experimental Setup We experiment six LLPO configurations by combining three PLMs (RoBERTa-large[9], DeBERTa-v3-large[10], and ModernBERT-large[11]) with two clustering methods (K-means and agglomerative). Unless otherwise noted, all results are based on the RoBERTa-large with K-means setup, which offered the best trade-off between accuracy and latency. Additional settings and results are provided in Appendices D and E, respectively.

[5] https://platform.openai.com/docs/models/gpt-4
[6] https://huggingface.co/zai-org/BPO
[7] https://huggingface.co/PKU-Baichuan-MLSystemLab/PAS-7B
[8] https://huggingface.co/Junrulu/FIPO-IPL-IPO-Tulu2-70B
[9] https://huggingface.co/FacebookAI/roberta-large
[10] https://huggingface.co/microsoft/deberta-v3-large
[11] https://huggingface.co/answerdotai/ModernBERT-large

Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR	*Opt.Latency*		Δ*Tot.Latency*
	A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win		A	B	
Claude-Haiku-3.5	LLPO	ori.	**54.5**	3.2	42.3	**66.3**	4.7	29.0	**75.0**	3.0	22.0	**49.3**	4.8	45.9	+26.5	0.01s	0.0s	+0.87s
GPT-3.5-turbo	LLPO	ori.	**52.6**	5.1	42.3	**50.8**	9.1	40.1	**64.0**	7.5	28.5	**58.8**	2.8	38.4	+19.2			+0.22s
Qwen2-0.5B	LLPO	ori.	**46.2**	10.9	42.9	44.0	9.2	**46.8**	**54.0**	7.5	38.5	**51.9**	9.9	38.2	+7.4			+1.05s
Qwen2-1.5B	LLPO	ori.	**62.2**	3.8	34.0	**54.4**	5.1	40.5	**60.5**	4.0	35.5	**59.4**	4.4	36.2	+22.6			+1.75s
Qwen2-7B	LLPO	ori.	**49.4**	2.5	48.1	**46.8**	8.8	44.4	46.0	3.0	**51.0**	48.9	1.8	**49.3**	-0.4			+0.13s
Claude-Haiku-3.5	LLPO	CoT	**48.8**	4.3	46.9	**58.3**	6.8	34.9	**59.0**	9.0	32.0	**56.6**	4.8	38.6	+17.6	0.01s	0.0s	+0.36
GPT-3.5-turbo	LLPO	CoT	41.7	3.8	**54.5**	**48.0**	6.0	46.0	**48.0**	9.0	43.0	**54.7**	2.3	43.0	+1.5			+0.03
Qwen2-0.5B	LLPO	CoT	**53.2**	7.1	39.7	**50.8**	9.1	40.1	**49.5**	10.0	40.5	**49.9**	9.3	40.8	+10.6			+0.04s
Qwen2-1.5B	LLPO	CoT	**60.9**	5.1	34.0	**53.6**	3.1	43.3	**62.0**	3.0	35.0	**63.2**	5.1	31.7	+23.9			+0.59s
Qwen2-7B	LLPO	CoT	37.8	1.9	**60.3**	34.5	6.8	**58.7**	30.0	5.5	**64.5**	**48.9**	2.8	48.3	-20.2			-1.35s
Claude-Haiku-3.5	LLPO	BPO	**52.6**	3.8	43.6	**59.1**	4.4	36.5	**68.0**	3.0	29.0	**58.6**	3.2	38.2	+22.8	0.01s	2.14s (↑ 198×)	-1.61s
GPT-3.5-turbo	LLPO	BPO	41.7	3.8	**54.5**	**48.8**	5.2	46.0	**51.5**	9.0	39.5	**58.8**	2.4	38.8	+5.5			-2.33s
Qwen2-0.5B	LLPO	BPO	37.8	7.7	**54.5**	**47.2**	6.8	46.0	**49.5**	8.0	42.5	**50.9**	9.1	40.0	+0.6			-2.37s
Qwen2-1.5B	LLPO	BPO	**57.1**	3.8	39.1	**49.6**	6.4	44.0	**60.0**	5.0	35.0	**61.0**	5.1	33.9	+18.9			-1.68s
Qwen2-7B	LLPO	BPO	46.8	2.6	**50.6**	39.3	5.1	**55.6**	41.0	3.0	**56.0**	**57.0**	2.2	40.8	-4.7			-3.74s
Claude-Haiku-3.5	LLPO	PAS	32.1	1.9	**66.0**	**50.8**	5.9	43.3	44.5	7.0	**48.5**	44.8	3.7	**51.5**	-9.3	0.01s	1.23s (↑ 114×)	-1.59s
GPT-3.5-turbo	LLPO	PAS	28.2	1.3	**70.5**	39.3	7.1	**53.6**	31.5	6.0	**62.5**	48.1	1.6	**50.3**	-22.5			-1.78s
Qwen2-0.5B	LLPO	PAS	35.3	4.4	**60.3**	40.5	6.3	**53.2**	44.0	11.5	**44.5**	44.0	8.1	**47.9**	-10.5			-2.17s
Qwen2-1.5B	LLPO	PAS	**46.8**	6.4	**46.8**	45.2	2.8	**52.0**	**50.0**	3.5	46.5	**51.7**	3.5	44.8	+0.9			-1.71s
Qwen2-7B	LLPO	PAS	39.1	2.6	**58.3**	35.3	4.8	**59.9**	33.0	2.5	**64.5**	44.4	2.9	**52.7**	-20.9			-2.93s
Claude-Haiku-3.5	LLPO	FIPO	**54.5**	1.9	43.6	**55.6**	0.7	43.7	**65.5**	3.5	31.0	**62.2**	2.4	35.4	+21.0	0.01s	21.12s (↑ 1956×)	-20.90s
GPT-3.5-turbo	LLPO	FIPO	48.7	1.9	**49.4**	48.0	3.2	**48.8**	**51.5**	4.0	44.5	**68.5**	0.8	30.7	+10.8			-21.14s
Qwen2-0.5B	LLPO	FIPO	40.4	4.5	**55.1**	38.1	5.2	**56.7**	40.5	6.5	**53.0**	41.8	6.5	**51.7**	-13.9			-21.04s
Qwen2-1.5B	LLPO	FIPO	45.5	4.5	**50.0**	48.8	0.8	**50.4**	**49.5**	5.0	45.5	**54.7**	4.9	40.4	+3.1			-20.24s
Qwen2-7B	LLPO	FIPO	44.2	2.6	**53.2**	46.0	3.2	**50.8**	41.0	2.5	**56.5**	**53.7**	2.5	43.8	-4.9			-21.85s

Table 2: Pairwise comparison of LLPO (A) against baseline methods (B: Original, CoT, BPO, PAS, and FIPO) on four instruction-following benchmarks (Koala Eval, Self-Instruct Eval, Dolly Eval, Arena-Hard Eval) across five base LLMs. Evaluations are conducted by GPT-4o. Boldface indicates the preferred method between A and B in each Win/Tie/Lose row. Red font in ΔWR highlights cases where LLPO outperforms the baseline in win rate. Underlined values in ΔTotal Latency indicate scenarios where LLPO incurs lower overall latency than the baseline.

Human Evaluation We conduct a human evaluation with five English-speaking annotators recruited via Positly[12]. Annotators are asked to rank outputs from LLPO, PAS, and FIPO based on four criteria: clarity, accuracy, completeness, and fluency. The evaluation set consists of 32 instances, created by randomly sampling approximately 5% from each of Koala Eval, Dolly Eval, and Self-Instruct Eval. Detailed procedures for the human evaluation, including sampling strategy, annotator instructions are provided in Appendix F.

4.2 Experimental Results

Table 2 presents a comprehensive comparison between LLPO and five baselines evaluated across four public instruction-following benchmarks and five base LLMs. Each cell reports the pairwise preference rates, while the rightmost columns summarize the aggregated win-rate gap (ΔWR), the average optimization latency (*Opt.Latency*), and the total latency difference (Δ*Tot.Latency*) relative to the baseline.

LLPO consistently outperforms the original prompt across most models and datasets. On four of five base models, it improves win rates by an average of 18.9%, with the highest gains on Dolly Eval up to 75.0%. These quality gains come with minimal cost: LLPO adds only 0.01 seconds of optimization latency on average and under 1.75 seconds total latency even on large models. Compared to prior prompt optimizers, LLPO offers a favorable quality-latency trade-off. Against BPO, LLPO achieves higher win rates on four of five models (+12.0% on average) while reducing latency by 2.3 seconds. It also outperforms or matches PAS and FIPO with significantly lower latency up to 21 seconds faster than FIPO. Finally, LLPO surpasses CoT in win rates on most models while maintaining comparable latency. These results highlight LLPO's ability to deliver high-quality outputs with low latency suitable for real-time use.

Human Evaluation While LLPO shows competitive performance compared to BPO in automatic evaluations, it underperforms relative to PAS and FIPO in several settings. However, recent studies have shown that automatic evaluations can be susceptible to various biases, such as verbosity or positional preference, motivating the need for human-centric assessment (Zheng et al., 2023; Liu et al., 2024; Doostmohammadi et al., 2024). To address this, we conduct a human evaluation to assess whether the observed performance gaps align with actual human preferences.

Annotators are shown the same original prompt along with three responses, each generated using

[12] https://www.positly.com/

	Koala Eval				Self-Instruct Eval				Dolly Eval				Arena-Hard Eval			
	LLPO	BPO	PAS	FIPO	LLPO	BPO	PAS	FIPO	LLPO	BPO	PAS	FIPO	LLPO	BPO	PAS	FIPO
avg	0.012s	1.384s	1.240s	23.660s	0.012s	1.714s	1.145s	17.783s	0.009s	2.277s	1.380s	16.802s	0.010s	3.188s	1.145s	26.226s
min	0.011s	0.163s	0.463s	3.164s	0.009s	0.255s	0.420s	2.228s	0.009s	0.207s	0.537s	2.134s	0.009s	0.279s	0.420s	2.368s
max	0.016s	12.533s	6.904s	291.246s	0.018s	12.848s	10.673s	290.641s	0.013s	37.814s	8.646s	290.623s	0.015s	40.619s	10.673s	291.165s

Table 3: Prompt optimization latency (avg/min/max) of LLPO, BPO, PAS, and FIPO across four benchmarks.

	LLPO	BPO	PAS	FIPO
Claude-Haiku-3.5	0.87s	2.48s (↑ 2.8×)	2.47s (↑ 2.8×)	21.78s (↑ 24.9×)
GPT-3.5-turbo	0.22s	2.55s (↑ 11.4×)	2.00s (↑ 9.0×)	21.37s (↑ 95.8×)
Qwen2-0.5B	1.05s	3.43s (↑ 3.3×)	3.22s (↑ 3.1×)	22.09s (↑ 21.0×)
Qwen2-1.5B	1.75s	3.43s (↑ 2.0×)	3.45s (↑ 2.0×)	21.98s (↑ 12.6×)
Qwen2-7B	0.13s	3.87s (↑ 29.0×)	3.07s (↑ 23.0×)	21.98s (↑ 164.9×)

Table 4: Total latency overhead (ΔTotal Latency) of LLPO, BPO, PAS, and FIPO compared to original prompts across different LLMs.

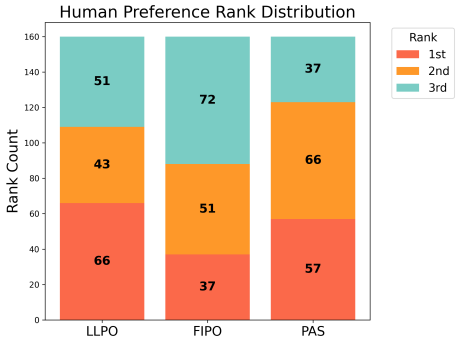

Figure 2: Human preference rank distribution for LLPO, FIPO, and PAS. The stacked bars show the number of times each system was ranked 1st, 2nd, or 3rd by annotators.

LLPO, PAS, or FIPO, and are asked to rank the responses from most to least preferred based on clarity, accuracy, completeness, and fluency. As illustrated in Figure 2, LLPO receives the highest number of 1st-place rankings, outperforming both PAS and FIPO in human preference.

Comparing Latency Table 3 highlights LLPO's latency advantage across all benchmarks. LLPO's optimization step averages just 0.01 seconds, roughly 198 times faster than BPO, 114 times faster than PAS and 1,956 times faster than FIPO, enabling real-time prompt optimization that other approaches cannot match. Unlike prior prompt optimization approaches' reliance on LLMs, LLPO's PLM-based classification achieves this speedup while preserving, or even surpassing output quality.

Table 4 further shows that LLPO adds minimal latency to total response time. LLPO adds at most 1.75 seconds across LLMs, whereas BPO and PAS increase latency by up to 29 times and 23 times, respectively. Notably, FIPO exhibits the largest

Base LLM	Method		ComplexBench			ΔWR
	A	B	A win	tie	B win	
Claude-Haiku-3.5	LLPO	ori.	**53.7**	9.3	37.0	+16.7
GPT-3.5-turbo	LLPO	ori.	**54.3**	7.4	38.3	+16.0
Qwen2-0.5B	LLPO	ori.	**46.7**	10.5	42.8	+3.9
Qwen2-1.5B	LLPO	ori.	**52.8**	5.9	41.3	+11.5
Qwen2-7B	LLPO	ori.	**48.0**	5.0	47.0	+1.0
Claude-Haiku-3.5	LLPO	CoT	**52.0**	8.3	39.7	+12.3
GPT-3.5-turbo	LLPO	CoT	**51.3**	7.2	41.5	+9.8
Qwen2-0.5B	LLPO	CoT	**53.4**	7.9	38.7	+14.7
Qwen2-1.5B	LLPO	CoT	**52.3**	4.6	43.1	+9.2
Qwen2-7B	LLPO	CoT	43.4	5.0	**51.6**	-8.2
Claude-Haiku-3.5	LLPO	BPO	**57.6**	6.1	36.3	+21.3
GPT-3.5-turbo	LLPO	BPO	**58.3**	5.6	36.1	+22.2
Qwen2-0.5B	LLPO	BPO	**51.0**	9.3	39.7	+11.3
Qwen2-1.5B	LLPO	BPO	**54.3**	5.5	40.2	+14.1
Qwen2-7B	LLPO	BPO	**54.2**	4.1	41.7	+12.5
Claude-Haiku-3.5	LLPO	PAS	42.0	8.8	**49.2**	-7.2
GPT-3.5-turbo	LLPO	PAS	**46.7**	6.6	**46.7**	0.0
Qwen2-0.5B	LLPO	PAS	42.7	9.1	**48.2**	-5.5
Qwen2-1.5B	LLPO	PAS	**47.8**	5.1	47.1	+0.7
Qwen2-7B	LLPO	PAS	44.5	4.5	**51.0**	-6.5
Claude-Haiku-3.5	LLPO	FIPO	**80.8**	1.1	18.1	+62.7
GPT-3.5-turbo	LLPO	FIPO	**78.9**	1.6	19.5	+59.4
Qwen2-0.5B	LLPO	FIPO	**62.0**	7.2	30.8	+31.2
Qwen2-1.5B	LLPO	FIPO	**72.7**	2.1	25.2	+47.5
Qwen2-7B	LLPO	FIPO	**76.3**	1.7	22.0	+54.3

Table 5: Pairwise comparison of LLPO (A) against baseline methods (B) on ComplexBench dataset, evaluated by GPT-4o.

	LLPO	BPO	PAS	FIPO
avg	0.009s	7.243s	1.570s	27.338s
min	0.009s	0.483s	0.469s	3.091s
max	0.016s	78.624s	25.873s	297.663s

Table 6: Prompt optimization latency statistics of LLPO, BPO, PAS, and FIPO on ComplexBench.

latency increase, reaching up to 165 times. These results indicate that LLPO maintains low latency overhead, making it well-suited for interactive or real-time deployments. Additional analyses on network-aware latency and token consumption are provided in Appendices G and H, respectively.

5 Analysis

5.1 Instruction-Following under Complex Structural Constraints

To evaluate whether LLPO generalizes to more complex instruction-following scenarios, we conduct experiments on ComplexBench (Wen et al.,

Scored by	Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR
		A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win	
GPT-4o	GPT-3.5-turbo	clustered	unclustered	**54.5**	2.6	42.9	**52.8**	9.5	37.7	**45.5**	14.5	40.0	**54.9**	3.7	41.4	**+11.4**
	Qwen2-7B	clustered	unclustered	**49.4**	5.1	45.5	**46.8**	7.2	46.0	**48.0**	8.0	44.0	**54.7**	3.7	41.6	**+5.5**

Table 7: Win-rate comparison between LLPO models trained with clustered and unclustered SCP labels across four instruction-following benchmarks, scored by GPT-4o. Boldface indicates the preferred method between A and B in each Win/Tie/Lose row.

2024), which emphasizes compositional structural constraints such as sequential and conditional requirements. As shown in Table 5, LLPO consistently outperforms or matches the baselines across most models and configurations, with especially pronounced gains over FIPO, achieving up to 62.7% higher aggregated win rates across both closed- and open-source LLMs. These results indicate that LLPO is also effective in settings where multiple constraints must be satisfied jointly.

Importantly, these performance improvements come with minimal cost. As summarized in Table 6, LLPO incurs minimal optimization latency, averaging 0.009 seconds, while FIPO introduces substantially higher overhead with an average latency of 27.3 seconds and a worst-case latency reaching 297.7 seconds. These results demonstrate that LLPO preserves its favorable quality–latency trade-off even under structurally complex instruction-following scenarios. Additional analysis of LLPO on reasoning-heavy and domain-specific tasks is provided in the Appendix I.

5.2 Effect of Field Clustering on Prompt Optimization and Output Alignment

We investigate how label clustering affects output quality by comparing models trained on raw versus clustered SCP labels. Some fields, like *Constraints* and *Role*, have over 10,000 unique values, creating sparsity and hindering generalization. Clustering reduce this to a manageable number (e.g., *Role*: 11,860 → 92), turning it into a semantically grounded classification task.

Clustering semantic field labels yield substantial gains in both intermediate classification accuracy and downstream output quality. The average F1-score of LLPO's intermediate classification improves from 0.01 (unclustered) to 0.46 (clustered). This enhancement in prompt alignment translated directly into better LLM responses. As shown in Table 7, clustering consistently improves win rates across all benchmarks and models. For instance, when scored by GPT-4o, clustering leads to win-rate improvements of +11.4% for GPT-3.5-turbo and +5.5% for Qwen2-7B.

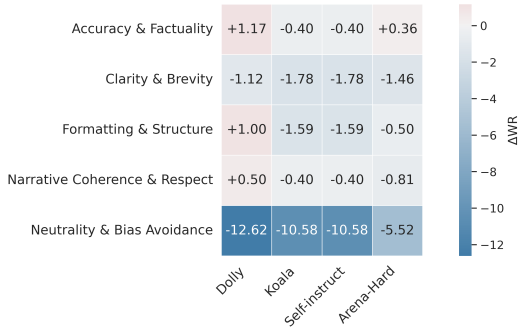

Figure 3: Impact of different constraint categories on performance across four benchmarks. Each row represents a semantic constraint category, and each cell shows the win-rate gap.

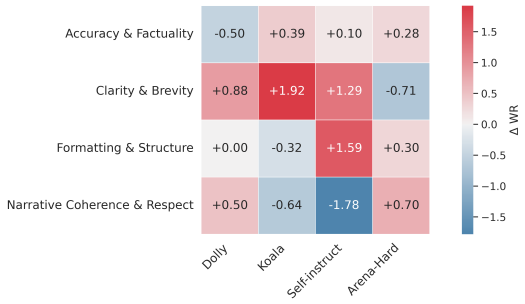

Figure 4: Performance change when replacing the predicted *Neutrality & Bias Avoidance* constraint with the next most probable category.

5.3 Understanding the Impact of SCP Field

To evaluate the impact of each SCP field, we conduct ablation studies. Removing fields like *Role*, *Audience*, *User Intent*, *Tone Type*, *Reasoning Guidance*, and *Output Format* consistently degrades performance, indicating their importance. Interestingly, removing the *Interactive Mode* and *Constraints* actually improves performance. Rather than suggesting that these fields are ineffective, we interpret this outcome as reflecting a mismatch between the SCP's general-purpose design and certain dataset-specific characteristics. For example, *Interactive Mode* assumes multi-turn contexts, but

most of our evaluation data is single-turn, causing it to misalign with user expectations.

To better understand why the *Constraints* field leads to the most pronounced degradation, we analyze the performance of five semantically grouped constraint categories across four datasets. As illustrated in Figure 3, the sharpest performance drop is associated with *Neutrality & Bias Avoidance* constraints. This trend aligns with qualitative findings reported in (OpenAI et al., 2024; Models, 2023), where interventions aimed at mitigating bias, such as refusal training, effectively reduce output bias but can also lead to decreased informativeness or reduced specificity in certain scenarios.

To test whether the performance drop was due to the presence of bias-related constraints, we conduct a follow-up experiment. Specifically, when the SCP classification model initially predicts a *Neutrality & Bias Avoidance* label, we instead replace it with the next most probable constraint category. As shown in Figure 4, this leads to noticeable improvements across most metrics. While these findings highlight the challenges of enforcing bias-related constraints, we argue that such constraints remain essential for building safe and socially responsible LLMs. Their presence should not be discarded but rather handled with greater contextual awareness.

Furthermore, under the bias-removed setup, we observe that the effectiveness of individual constraint types varied across datasets, improving performance in some while degrading it in others. These findings indicate that structured fields like *Constraints* are not universally helpful or harmful, but depend on task format and user intent. This highlights the need for adaptive prompt construction that aligns field values with interaction context.

6 Conclusion

In this paper, we introduce LLPO, a low-latency prompt optimization framework. By leveraging PLMs and a structured prompt schema, LLPO generates high-quality prompts with minimal latency. Experiments show that LLPO matches or surpasses prior methods, with up to 1,956 times faster prompt optimization. Human evaluations further validate its effectiveness. Ablation studies reveal that most SCP fields contribute to performance, though *Interactive Mode* and *Constraints* can reduce it. Nevertheless, we believe these fields remain important for building reliable systems and can be applied with awareness of task-specific needs.

Limitations

Despite its strengths, LLPO has several limitations. The effectiveness of structured fields varies across datasets and suboptimal combinations may introduce noise rather than useful guidance. Although clustering mitigates label sparsity, it relies on manually defined taxonomies and may miss subtle distinctions in user intent. In addition, follow-up experiments show that LLPO is less effective in reasoning-heavy domains, indicating limitations in handling explicit multi-step reasoning. Our evaluation also does not include large-scale models like GPT-4 due to resource constraints. Moreover, the classification module is evaluated only with lightweight models, leaving the latency–quality trade-off of larger classifiers unexplored. Finally, we do not study how LLPO's optimized prompts interact with built-in system prompts in commercial LLMs, which may affect outputs in unforeseen ways.

Ethical Considerations

Generative AI models were used to generate dataset used in our study. In addition, ChatGPT was employed solely for ancillary purposes during manuscript preparation, including grammar correction and translation. Any generative AI models were not involved in the conception or development of any scientific contributions in this work.

Acknowledgements

We thank the anonymous reviewers for their helpful comments. This work was supported by the Basic Research Program through a National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT) (No.2022R1C1C1010317).

References

Lee Boonstra. 2025. Prompt engineering whitepaper.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Sondos Mahmoud Bsharat, Aidar Myrzakhan, and Zhiqiang Shen. 2023. Principled instructions are all you need for questioning llama-1/2, gpt-3.5/4. *arXiv preprint arXiv:2312.16171*.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2024. InstructZero: Efficient instruction optimization for black-box large language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 6503–6518. PMLR.

Jiale Cheng, Xiao Liu, Kehan Zheng, Pei Ke, Hongning Wang, Yuxiao Dong, Jie Tang, and Minlie Huang. 2024. Black-box prompt optimization: Aligning large language models without model training. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3201–3219, Bangkok, Thailand. Association for Computational Linguistics.

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*.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. 2023. Free dolly: Introducing the world's first truly open instruction-tuned llm.

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, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ehsan Doostmohammadi, Oskar Holmström, and Marco Kuhlmann. 2024. How reliable are automatic evaluation methods for instruction-tuned LLMs? In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6321–6336, Miami, Florida, USA. Association for Computational Linguistics.

Xinyang Geng, Arnav Gudibande, Hao Liu, Eric Wallace, Pieter Abbeel, Sergey Levine, and Dawn Song. 2023. Koala: A dialogue model for academic research. Blog post.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. 2024. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *The Twelfth International Conference on Learning Representations*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E Gonzalez, and Ion Stoica. 2024. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. *arXiv preprint arXiv:2406.11939*.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Yiqi Liu, Nafise Moosavi, and Chenghua Lin. 2024. LLMs as narcissistic evaluators: When ego inflates evaluation scores. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 12688–12701, Bangkok, Thailand. Association for Computational Linguistics.

Junru Lu, Siyu An, Min Zhang, Yulan He, Di Yin, and Xing Sun. 2025. FIPO: Free-form instruction-oriented prompt optimization with preference dataset and modular fine-tuning schema. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 11029–11047, Abu Dhabi, UAE. Association for Computational Linguistics.

J MacQueen. 1967. Multivariate observations. In *Proceedings ofthe 5th Berkeley Symposium on Mathematical Statisticsand Probability*, volume 1, pages 281–297.

Claude Models. 2023. Model card and evaluations for claude models.

Fionn Murtagh and Pierre Legendre. 2014. Ward's hierarchical agglomerative clustering method: which algorithms implement ward's criterion? *Journal of classification*, 31(3):274–295.

OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander Mądry, Alex Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, and 401 others. 2024. Gpt-4o system card. *Preprint*, arXiv:2410.21276.

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968, Singapore. Association for Computational Linguistics.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended abstracts of the 2021 CHI conference on human factors in computing systems*, pages 1–7.

Timo Schick and Hinrich Schütze. 2022. True few-shot learning with prompts—a real-world perspective. *Transactions of the Association for Computational Linguistics*, 10:716–731.

Taylor Shin, Yasaman Razeghi, Robert L. Logan IV, Eric Wallace, and Sameer Singh. 2020. AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and 1 others. 2023. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy Lillicrap, Angeliki Lazaridou, and 1332 others. 2025. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Yidong Wang, Zhuohao Yu, Wenjin Yao, Zhengran Zeng, Linyi Yang, Cunxiang Wang, Hao Chen, Chaoya Jiang, Rui Xie, Jindong Wang, Xing Xie, Wei Ye, Shikun Zhang, and Yue Zhang. 2024. PandaLM: An automatic evaluation benchmark for LLM instruction tuning optimization. In *The Twelfth International Conference on Learning Representations*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

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

Bosi Wen, Pei Ke, Xiaotao Gu, Lindong Wu, Hao Huang, Jinfeng Zhou, Wenchuang Li, Binxin Hu, Wendy Gao, Jiaxing Xu, and 1 others. 2024. Benchmarking complex instruction-following with multiple constraints composition. *Advances in Neural Information Processing Systems*, 37:137610–137645.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. *Preprint*, arXiv:2309.03409.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. Opt: Open pre-trained transformer language models. *Preprint*, arXiv:2205.01068.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. 2023. TEMPERA: Test-time prompt editing via reinforcement learning. In *The Eleventh International Conference on Learning Representations*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Miao Zheng, Hao Liang, Fan Yang, Haoze Sun, Tianpeng Li, Lingchu Xiong, Yan Zhang, Yozhen Wu, Kun Li, Yanjun Sheng, and 1 others. 2024. Pas: Data-efficient plug-and-play prompt augmentation system. *arXiv preprint arXiv:2407.06027*.

Yongchao Zhou, Andrei Ioan Muresanu, Ziwen Han, Keiran Paster, Silviu Pitis, Harris Chan, and Jimmy Ba. 2023. Large language models are human-level prompt engineers. In *The Eleventh International Conference on Learning Representations*.

A Detailed Explanation of the Eight SCP Fields

Our design of eight prompt fields is grounded in Bsharat et al. (2023) on principled instructions and best practices for prompt engineering provided by Boonstra (2025). Below, we describe the rationale for each field, highlighting how it reflects theoretical principles and empirical findings from these studies.

Role Explicitly assigning a role to the language model helps simulate domain expertise and constrain its responses to a specific persona. This encourages consistent tone and perspective, improving interpretability and user trust. It also boosts alignment between instructions and model behavior, especially in professional or context-sensitive scenarios.

Audience Indicating the target audience (e.g., novice, expert, child) calibrates the model's register, vocabulary, and explanatory depth. Including audience information enables the model to adapt its outputs in terms of complexity and tone, resulting in more contextually appropriate and accessible responses.

User Intent Clarifying the underlying purpose of the prompt, such as explanation, problem-solving, or creative generation, ensures task alignment. Making the intent explicit guides the model's reasoning process and reduces the likelihood of irrelevant or off-topic outputs.

Tone Type Specifying the desired tone (e.g., formal, humorous, descriptive) helps control the style of the output. This supports use cases requiring tone consistency such as educational content, persuasive writing, or professional communication, and prevents stylistic ambiguity in model responses.

Constraints Including requirements such as factual accuracy, neutrality, or safety constraints helps operationalize ethical and normative considerations. By making such boundaries explicit, the model is more likely to produce outputs that are aligned with user expectations and mitigate unintended harms.

Reasoning Guidance Providing instructions for structured reasoning (e.g., "step-by-step", "cite evidence") encourages transparent and interpretable outputs. This is particularly beneficial for complex tasks where intermediate reasoning steps improve accuracy and allow users to evaluate the logic behind conclusions.

Output Format Specifying the expected output structure (e.g., bullet points, JSON, table) enhances consistency and readability, and enables easier integration into downstream applications. Format cues reduce ambiguity and help the model organize content in predictable, user-friendly ways.

Interactive Mode Defining interaction policies (e.g., whether the model should ask clarifying questions) facilitates smoother user interaction. This supports dynamic use cases like tutoring or co-creation, where iterative refinement and user engagement are critical for success.

Collectively, these eight fields synthesize the theoretical and practical insights from principled instruction design and industrial prompt-engineering practices. They provide a structured yet flexible framework that captures "who is speaking, to whom, for what purpose, and how the answer should be delivered," ensuring prompts are both interpretable for humans and actionable for models.

B Implementation Details of Clustering

B.1 Determining the Number of Clusters

Implementing clustering-based label reduction necessitates specifying the number of clusters, k for K-means and *n_clusters* for agglomerative clustering. To identify appropriate values, we conduct a silhouette score–driven search. Initially, we perform a coarse-grained exploration in the range of 50 to 500, increasing the cluster count in increments of 50. The configuration yielding the highest silhouette score for each field is selected and used to cluster the labels prior to training the pretrained language models (PLMs) on the pruned dataset. However, this coarse configuration results in suboptimal classification performance. To address this issue, we refine the search space by exploring narrower ranges with finer granularity: (50–150), (50–100), and (10–50), all using a step size of 1. The cluster counts and corresponding classification performance for each configuration are reported in Table 8 and Table 9.

We observe that reducing the maximum number of clusters from 500 to 150, 100, and finally 50 leads to consistent improvements in downstream model performance. Notably, the best results for both clustering methods are obtained within the

method	$(k_{min}, k_{max}, step\_size)$	field	k	silhouette score	f1score	Avg_f1-score
K-means	(50, 500, 50)	*Role*	500	0.083	0.231	0.114
		Audience	350	0.071	0.235	
		User Intent	500	0.090	0.067	
		Tone Type	500	0.139	0.046	
		Constraints	500	0.048	0.072	
		Reasoning Guidance	500	0.048	0.042	
		Output Format	400	0.069	0.108	
	(50, 150, 1)	*Role*	142	0.076	0.467	0.271
		Audience	129	0.067	0.401	
		User Intent	150	0.071	0.177	
		Tone Type	51	0.112	0.179	
		Constraints	128	0.039	0.165	
		Reasoning Guidance	54	0.047	0.196	
		Output Format	75	0.063	0.311	
	(50, 100, 1)	*Role*	92	0.072	0.533	0.297
		Audience	99	0.060	0.418	
		User Intent	100	0.063	0.221	
		Tone Type	51	0.112	0.179	
		Constraints	67	0.038	0.210	
		Reasoning Guidance	54	0.047	0.192	
		Output Format	75	0.063	0.327	
	(10, 50, 1)	*Role*	41	0.063	0.598	0.446
		Audience	47	0.056	0.510	
		User Intent	15	0.122	0.489	
		Tone Type	26	0.118	0.290	
		Constraints	17	0.065	0.334	
		Reasoning Guidance	10	0.143	0.406	
		Output Format	15	0.066	0.492	

Table 8: Silhouette scores and classification performance across search ranges under K-means clustering.

method	$(n_{min}, n_{max}, step\_size)$	field	n_clusters	silhouette score	f1score	Avg_f1-score
Agglomerative	(50, 500, 50)	*Role*	500	0.076	0.274	0.114
		Audience	500	0.063	0.174	
		User Intent	500	0.101	0.083	
		Tone Type	500	0.167	0.051	
		Constraints	500	0.031	0.070	
		Reasoning Guidance	500	0.054	0.049	
		Output Format	500	0.060	0.098	
	(50, 150, 1)	*Role*	147	0.054	0.477	0.245
		Audience	150	0.038	0.386	
		User Intent	148	0.058	0.170	
		Tone Type	137	0.093	0.099	
		Constraints	150	0.010	0.139	
		Reasoning Guidance	60	0.036	0.208	
		Output Format	150	0.043	0.233	
	(50, 100, 1)	*Role*	75	0.050	0.570	0.309
		Audience	82	0.034	0.482	
		User Intent	54	0.057	0.287	
		Tone Type	75	0.093	0.151	
		Constraints	85	0.003	0.186	
		Reasoning Guidance	60	0.036	0.202	
		Output Format	100	0.035	0.288	
	(10, 50, 1)	*Role*	49	0.044	0.599	0.427
		Audience	50	0.026	0.508	
		User Intent	27	0.095	0.386	
		Tone Type	50	0.081	0.189	
		Constraints	17	0.028	0.304	
		Reasoning Guidance	10	0.123	0.463	
		Output Format	14	0.041	0.541	

Table 9: Silhouette scores and classification performance across search ranges under agglomerative clustering.

range of 10 to 50. Consequently, we adopt the cluster configuration that maximizes the silhouette score within this range. Nonetheless, this reduction introduces trade-offs: smaller cluster counts may collapse semantically distinct labels into a single group, leading to information loss. To mitigate

Field	Raw unique values	K-means clusters	Agglomerative clusters
Role	12019	92	75
Audience	11439	47	50
User Intent	4501	15	27
Tone Type	2239	26	50
Constraints	13751	17	17
Reasoning Guidance	6319	10	10
Output Format	8388	15	14
Interactive Mode	145	2	2

Table 10: Number of raw unique values and final clustered labels for each SCP field.

this degradation, we fix the number of clusters for the *Role* field at the configuration that yields the highest silhouette score when the maximum value is set to 100, for both K-means ($k_{\max} = 100$) and agglomerative clustering ($n_{\max} = 100$). This decision reflects the critical importance of preserving semantic integrity in role assignment.

Interactive Mode requires a tailored approach because its raw labels naturally fall into two categories, allowed and not allowed. We observe that semantically identical strings such as "allowed" and "allowed at the end of the response" are placed in separate clusters. To improve classification accuracy, we manually define two canonical labels, "not allowed", and "allowed at the end of the response for clarification". We then map each original variant to its corresponding canonical label.

By consolidating semantically equivalent labels into coherent categories, the clustering process reduces ambiguity and enhances generalization. This, in turn, improves classifier robustness and ensures that the system can adapt effectively to the diversity of real-world usage scenarios. Table 10 summarizes the final number of clusters used for each SCP field, as determined by the procedures described above. It reports both the raw unique values and the number of clustered labels produced by K-means and agglomerative clustering.

B.2 Representative Label Selection

For each cluster, we assign a representative label that serves as the canonical value to which all other labels in the same cluster are mapped. The representative label is chosen based on its proximity to the cluster center.

- **K-means clustering:** The cluster centroid is directly obtained from the algorithm. Among all values assigned to a given cluster, we select the label whose embedding lies closest to the centroid, and designate it as the representative. All other labels in that cluster are then replaced by this representative label.
- **Agglomerative clustering:** Since agglomerative clustering does not return explicit centroids, we compute the mean of all embeddings in a cluster and treat it as the cluster center. The representative label is then selected as the one whose embedding is nearest to this mean vector. This approximation ensures that the representative label is the most central value in its cluster.

This procedure guarantees that each cluster is represented by the label most semantically aligned with the cluster center, thereby minimizing semantic drift when replacing other labels with their representative.

C Details of Evaluation

C.1 Evaluation Datasets

To obtain a balanced view of our model's instruction-following alignment, we evaluate on four public benchmarks that vary in topic breadth and difficulty. All datasets we used are free to use for research.

- **Koala Eval**: A 156-prompt subset of AlpacaEval covering creative, factual, coding, and planning tasks.
- **Self-Instruct Eval**: Consists of 252 expert-written instructions held out from the Self-Instruct training pipeline.
- **Dolly Eval**: Includes 200 prompts uniformly sampled from the eight task categories of the Dolly 15K corpus, replicating the evaluation subset used in BPO for direct comparison.
- **Arena-Hard Eval**: Contains 500 high-difficulty prompts distilled from Chatbot Arena logs. To prevent data leakage from the Arena subset used in BPO's preference tuning, we identified and removed five overlapping prompts, resulting in a final evaluation set of 495 prompts.

C.2 LLM-as-a-Judge Prompt Template

Figure 5 presents the full prompt template used in our evaluation setting. The template consists of a system message instructing the LLM to compare the quality of two candidate responses to a

System Prompt

Please act as an impartial judge and evaluate the quality of the responses provided by two AI assistants to the user question displayed below. You should choose the assistant that follows the user's instructions and answers the user's question better. Your evaluation should consider factors such as the helpfulness, relevance, accuracy, depth, creativity, and level of detail of their responses. Begin your evaluation by comparing the two responses and provide a short explanation. Avoid any position biases and ensure that the order in which the responses were presented does not influence your decision. Do not allow the length of the responses to influence your evaluation. Do not favor certain names of the assistants. Be as objective as possible. After providing your explanation, output your final verdict by strictly following this format: "[[A]]" if assistant A is better, "[[B]]" if assistant B is better, and "[[C]]" for a tie.

User Prompt

[User Question]
{input}

[The Start of Assistant A's Answer]
{output_a}[The End of Assistant A's Answer]

[The Start of Assistant B's Answer]
{output_b}
[The End of Assistant B's Answer]

Figure 5: Pairwise scoring prompt.

given user query, and a user message containing the question and responses. To ensure fairness, we randomize the order of candidate responses during each evaluation.

C.3 Details of Base LLMs

Throughout this paper, we refer to these base LLMs using the following standardized identifiers: Claude-Haiku-3.5 refers to Anthropic's Claude 3.5 Haiku model[13] (snapshot as of claude-3-5-haiku-latest); GPT-3.5-turbo denotes OpenAI's gpt-3.5-turbo model[14]. For open-source models, we use Qwen2-0.5B[15], Qwen2-1.5B[16], and Qwen2-7B[17] to indicate the instruction-tuned Qwen2 variants with 0.5, 1.5, and 7 billion parameters. These naming conventions are used consistently across all experimental tables and analyses.

D Implementation Settings and Classification Performance

All experiments were conducted using one NVIDIA A100 80GB GPU. Training the PLM took approximately 10 hours in total (about 15 minutes per epoch), and inference using the LLM required around 20 hours.

D.1 Train Data

The BPO dataset is a human preference–driven instruction optimization corpus constructed from four publicly available instruction-tuning datasets, while the PAS dataset spans 14 diverse categories including coding, medical QA, logical reasoning, and system instruction tasks. To construct a reliable SCP dataset from BPO and PAS dataset, we first preprocessed the data by removing duplicates and filtering out non-English samples. After this preprocessing, we used 14,331 out of 14,395 instances from the BPO dataset and 6,362 out of 7,729 instances from the PAS dataset, resulting in a total of 20,693 instances. We randomly split the dataset into 80% for training, 10% for validation, and 10% for test. This resulted in 16,554 instances for training, 2,069 for validation, and 2,070 for test, respectively. Both datasets are publicly available and free to use for research purposes.

D.2 Training Settings

The training hyperparameter configurations for all backbone PLMs, including RoBERTa-large, DeBERTa-v3-large, and ModernBERT-large are summarized in Table 11. For each backbone, we report the settings used with both K-means and agglomerative clustering, covering batch size, learning rate, dropout, number of epochs, and focal loss parameters.

D.3 Classification Performance

The classification performance of the SCP field prediction models across all backbone PLMs is summarized in Table 12. We report F1-scores for each SCP field under different combinations of backbone models and clustering methods, along with the overall average performance across fields.

E Additional Results with Alternative LLPO Configurations

In addition to the default LLPO configuration using RoBERTa-large with K-means clustering, we evaluate five other model-clustering combinations to ex-

[13] https://www.anthropic.com/claude/haiku
[14] https://platform.openai.com/docs/models/gpt-3.5-turbo
[15] https://huggingface.co/Qwen/Qwen2-0.5B-Instruct
[16] https://huggingface.co/Qwen/Qwen2-1.5B-Instruct
[17] https://huggingface.co/Qwen/Qwen2-7B-Instruct

Backbone PLM	Clustering	Batch	LR	Dropout	Epochs	Focal γ
RoBERTa-large	K-means	64	1e-4	0.3	15	2.0
	Agglomerative	64	1e-4	0.3	15	1.0
DeBERTa-v3-large	K-means	32	1e-5	0.3	15	2.0
	Agglomerative	32	1e-5	0.0	15	2.0
ModernBERT-large	K-means	64	1e-4	0.5	15	2.0
	Agglomerative	32	1e-4	0.3	15	1.0

Table 11: Training hyperparameter configurations for different backbone PLMs and clustering methods.

Backbone PLM	Clustering	Role	Audience	Intent	Tone	Constraints	Reasoning	Format	Interactive	Overall Avg.
RoBERTa-large	K-means	0.52	0.50	0.50	0.30	0.34	0.42	0.50	0.64	0.46
	Agglomerative	0.55	0.48	0.39	0.27	0.28	0.28	0.53	0.65	0.43
DeBERTa-v3-large	K-means	0.47	0.49	0.50	0.26	0.34	0.44	0.47	0.64	0.45
	Agglomerative	0.49	0.49	0.35	0.24	0.27	0.24	0.50	0.64	0.40
ModernBERT-large	K-means	0.55	0.53	0.51	0.30	0.36	0.45	0.50	0.65	0.48
	Agglomerative	0.57	0.51	0.40	0.28	0.27	0.28	0.56	0.65	0.44

Table 12: F1-score performance for SCP field classification across backbone PLMs and clustering methods. Overall Avg. denotes the average F1-score across all SCP fields.

amine the generalizability and robustness of LLPO across different setups. Detailed performance comparisons for each configuration are provided in Table 13 through Table 15.

F Human Evaluation Details

Methodology To evaluate whether the automatic performance differences between LLPO, PAS, and FIPO align with human preferences, we conducted a human study involving five annotators from English-speaking countries via Positly[18]. Annotators were between 31 and 48 years old and required to have completed at least a bachelor's degree. The annotators were compensated at $25.00 per activity completion, with an estimated hourly rate of $30.00/hour. Each participant completed a single evaluation session, taking on average 32.2 minutes to finish all assigned tasks. Each annotator assessed the model outputs independently, without collaboration.

Evaluation Data Sampling We constructed an evaluation dataset by randomly sampling approximately 5% of prompts from each of three instruction tuning datasets: Koala Eval, Dolly Eval, and Self-Instruct Eval. This yielded a total of 32 distinct prompts. We excluded prompts from the Arena-Hard benchmark because many of them involve coding-related tasks that require domain-specific expertise, making them less suitable for general-purpose human evaluation.

[18] https://www.positly.com/

Annotator Instruction Participants were instructed to rank the three responses based on their subjective preference while considering four evaluation criteria: clarity (ease of understanding), accuracy (factual correctness), completeness (coverage of the prompt), and style/fluency (coherence and readability). The instructions emphasized that there were no correct or incorrect answers and that rankings should reflect the annotator's own judgment. For each prompt, three responses, generated using LLPO, PAS, and FIPO, were shown in random order to prevent positional bias. Annotators were blind to the source of each response and were tasked with reading the original prompt and the three model-generated responses and assigning a unique rank (1st, 2nd, or 3rd) to each response based on their subjective preference. These instructions were clearly presented at the beginning of the task to ensure consistent interpretation across annotators. Examples of the task description and the annotation interface presented to participants are shown in Figure 6.

G Extended Latency Analysis with Network Overhead

When the online LLM APIs, such as GPT-4 accessed via OpenAI, are invoked remotely, network latency can become a dominant component of end-to-end response time. To more realistically evaluate end-to-end latency in such scenarios, we additionally measure network round-trip

Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR	*Opt.Latency*		Δ*Tot.Latency*
	A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win		A	B	
Claude-Haiku-3.5	LLPO	ori.	**50.0**	5.8	44.2	**67.5**	3.9	28.6	**75.0**	3.5	21.5	**55.4**	5.0	39.6	+28.5	0.01s	0.0s	+0.99s
GPT-3.5-turbo	LLPO	ori.	46.8	3.2	**50.0**	**50.8**	6.7	42.5	**54.5**	6.0	39.5	**58.2**	2.8	39.0	+9.8			+0.59s
Qwen2-0.5B	LLPO	ori.	43.6	9.6	**46.8**	42.1	9.9	**48.0**	**48.0**	10.5	41.5	**50.1**	10.3	39.6	+2.0			+0.58s
Qwen2-1.5B	LLPO	ori.	**63.5**	4.4	32.1	**50.0**	5.2	44.8	**50.5**	7.0	42.5	**58.4**	3.0	38.6	+16.1			+1.14s
Qwen2-7B	LLPO	ori.	45.5	5.8	**48.7**	**48.0**	5.6	46.4	42.0	4.0	**54.0**	43.0	3.1	**53.9**	-6.1			+0.17s
Claude-Haiku-3.5	LLPO	CoT	**51.3**	4.5	44.2	**54.4**	5.9	39.7	**54.5**	4.5	41.0	**62.2**	4.1	33.7	+16.0	0.01s	0.0s	+0.48
GPT-3.5-turbo	LLPO	CoT	37.8	3.2	**59.0**	**46.4**	7.2	**46.4**	**48.0**	6.5	45.5	**58.0**	3.2	38.8	+0.1			+0.40
Qwen2-0.5B	LLPO	CoT	**53.2**	7.1	39.7	46.0	5.6	**48.4**	45.0	9.0	**46.0**	**50.3**	7.1	42.6	+4.5			-0.43
Qwen2-1.5B	LLPO	CoT	**60.3**	5.1	34.6	48.0	3.6	**48.4**	**47.5**	6.0	46.5	**54.1**	4.9	41.0	+9.9			-0.01
Qwen2-7B	LLPO	CoT	38.5	4.4	**57.1**	39.7	4.3	**56.0**	34.5	3.5	**62.0**	43.0	3.7	**53.3**	-18.2			-1.31s
Claude-Haiku-3.5	LLPO	BPO	**52.6**	2.5	44.9	**63.9**	5.1	31.0	**68.0**	3.0	29.0	**63.0**	3.5	33.5	+27.3	0.01s	2.14s (↑ 163×)	-1.50s
GPT-3.5-turbo	LLPO	BPO	41.0	3.9	**55.1**	46.4	4.0	**49.6**	**50.5**	7.0	42.5	**59.6**	3.4	37.0	+3.3			-1.96s
Qwen2-0.5B	LLPO	BPO	**47.4**	7.1	45.5	40.1	9.9	**50.0**	**48.0**	10.0	42.0	**51.5**	9.3	39.2	+2.6			-2.84s
Qwen2-1.5B	LLPO	BPO	**54.5**	4.5	41.0	**48.4**	3.6	48.0	41.0	8.5	**50.5**	**59.0**	5.8	35.2	+7.1			-2.29s
Qwen2-7B	LLPO	BPO	**50.0**	3.8	46.2	47.2	3.2	**49.6**	38.5	3.5	**58.0**	**62.0**	1.6	36.4	+1.9			-3.70s
Claude-Haiku-3.5	LLPO	PAS	35.9	2.6	**61.5**	42.5	5.1	**52.4**	40.5	5.5	**54.0**	46.5	3.6	**49.9**	-13.1	0.01s	1.23s (↑ 93×)	-1.48s
GPT-3.5-turbo	LLPO	PAS	28.2	3.9	**67.9**	33.3	5.6	**61.1**	31.5	6.0	**62.5**	46.7	2.6	**50.7**	-25.6			-1.41s
Qwen2-0.5B	LLPO	PAS	37.2	3.8	**59.0**	38.9	6.3	**54.8**	43.0	8.5	**48.5**	43.6	8.5	**47.9**	-11.9			-2.64s
Qwen2-1.5B	LLPO	PAS	46.2	3.8	**50.0**	41.7	4.7	**53.6**	44.5	8.0	**47.5**	**49.5**	4.6	45.9	-3.8			-2.31s
Qwen2-7B	LLPO	PAS	36.5	0.0	**63.5**	40.5	3.9	**55.6**	33.5	3.5	**63.0**	45.1	4.0	**50.9**	-19.4			-2.90s
Claude-Haiku-3.5	LLPO	FIPO	**54.5**	1.9	43.6	**55.2**	3.1	41.7	**66.0**	3.0	31.0	**65.5**	2.8	31.7	+23.3	0.01s	21.12s (↑ 1605×)	-20.79s
GPT-3.5-turbo	LLPO	FIPO	47.4	0.7	**51.9**	46.4	4.0	**49.6**	**50.0**	3.0	47.0	**67.1**	2.2	30.7	+7.9			-20.77s
Qwen2-0.5B	LLPO	FIPO	37.2	5.1	**57.7**	38.1	6.3	**55.6**	37.5	6.0	**56.5**	41.8	6.3	**51.9**	-16.8			-21.51s
Qwen2-1.5B	LLPO	FIPO	**48.7**	3.2	48.1	44.8	4.4	**50.8**	43.5	5.5	**51.0**	**54.5**	2.9	42.6	-0.3			-20.84s
Qwen2-7B	LLPO	FIPO	44.2	2.0	**53.8**	42.5	3.5	**54.0**	41.0	3.0	**56.0**	**53.7**	1.9	44.4	-6.7			-21.82s

Table 13: Performance comparison of LLPO with RoBERTa-large using agglomerative clustering

time (RTT) by approximating the TCP connection setup time and analyze how RTT contributes to inference latency through a breakdown of the time to first token (TTFT). TTFT measures the elapsed time from request initiation to the receipt of the first generated token and is defined as the sum of RTT and backend latency, where backend latency corresponds to server-side processing time until first-token generation. Formally, $TTFT = RTT + backend\ latency$.

Table 16 reports a detailed latency breakdown for different prompt optimization methods on Claude-Haiku-3.5 (shown above) and GPT-3.5-Turbo (shown below), including total latency, optimization latency, inference latency, TTFT, RTT, and backend latency. Across both models, RTT contributes only a very small fraction of the overall inference latency, typically on the order of several to tens of milliseconds, while TTFT and backend latency dominate the end-to-end response time.

These results indicate that even when downstream LLMs are accessed remotely, network overhead does not diminish the relative benefit of LLPO. In particular, the local classification component employed by LLPO is computationally lightweight and introduces minimal overhead compared to the backend processing and token generation costs of the remote LLM. Consequently, the efficiency advantage of LLPO remains meaningful in realistic deployment scenarios involving remote API-based language models.

H Token Consumption Analysis

Token consumption is an important factor for understanding the computational characteristics of different prompt optimization methods. To complement the latency-focused analysis in the main paper, we provide additional statistics on input and output token usage across methods.

Table 17 reports the average number of input and output tokens generated during inference across four evaluation datasets for LLPO and all baseline methods, evaluated on three open-source Qwen2 models (0.5B, 1.5B, and 7B). We observe that LLPO consistently produces longer input prompts compared to baseline methods.

These results highlight a trade-off between prompt expressiveness and input token length. While longer input prompts may marginally increase inference latency due to additional token processing, LLPO consistently achieves lower total latency than baseline methods due to its substantially reduced prompt optimization latency.

I Performance on Reasoning-Heavy and Domain-Specific Tasks

To further examine the generalizability of SCP, we conduct additional experiments on two reasoning-heavy and domain-specific benchmarks: the causal-judgement subset of BBH (Suzgun et al., 2023), which requires multi-step commonsense and causal reasoning, and GSM8K (Cobbe et al., 2021), which focuses on mathematical problem solving.

K-means clustering

Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR	*Opt.Latency*		*ΔTot.Latency*
	A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win		A	B	
Claude-Haiku-3.5	LLPO	ori.	**55.8**	3.8	40.4	**64.3**	6.7	29.0	**75.0**	3.0	22.0	**50.1**	4.0	45.9	+27.0	0.02s	0.0s	+1.04s
GPT-3.5-turbo	LLPO	ori.	**50.0**	6.4	43.6	**50.8**	7.5	41.7	**60.0**	8.5	31.5	**54.9**	2.5	42.6	+14.1			+0.24s
Qwen2-0.5B	LLPO	ori.	**50.6**	5.2	44.2	**47.6**	9.5	42.9	**53.0**	7.0	40.0	**53.1**	10.3	36.6	+10.2			+1.11s
Qwen2-1.5B	LLPO	ori.	**61.5**	2.6	35.9	**56.3**	6.4	37.3	**60.0**	6.5	33.5	**63.4**	3.5	33.1	+25.4			+1.96s
Qwen2-7B	LLPO	ori.	**48.1**	4.5	47.4	**47.2**	6.8	46.0	42.5	4.5	**53.0**	**49.7**	2.0	48.3	-1.8			+0.20s
Claude-Haiku-3.5	LLPO	CoT	**55.1**	3.9	41.0	**58.3**	3.6	38.1	**60.5**	6.0	33.5	**60.2**	3.2	36.6	+21.2	0.02s	0.0s	+0.54s
GPT-3.5-turbo	LLPO	CoT	40.4	6.4	**53.2**	46.8	4.8	**48.4**	**52.5**	8.5	39.0	**54.5**	3.3	42.2	+2.9			+0.04s
Qwen2-0.5B	LLPO	CoT	**50.0**	9.0	41.0	**52.0**	6.7	41.3	42.0	8.5	**49.5**	**50.9**	8.7	40.4	+5.7			+0.10s
Qwen2-1.5B	LLPO	CoT	**63.5**	5.1	31.4	**51.6**	4.0	44.4	**54.5**	6.5	39.0	**58.6**	4.0	37.4	+19.0			+0.80s
Qwen2-7B	LLPO	CoT	35.3	3.8	**60.9**	36.5	4.0	**59.5**	31.5	1.0	**67.5**	46.7	2.2	**51.1**	-22.3			-1.28s
Claude-Haiku-3.5	LLPO	BPO	**56.4**	2.6	41.0	**60.3**	5.2	34.5	**66.5**	4.5	29.0	**57.6**	3.0	39.4	+24.2	0.02s	2.14s (↑ 87×)	-1.44s
GPT-3.5-turbo	LLPO	BPO	42.9	2.0	**55.1**	46.4	6.8	**46.8**	**54.0**	8.0	38.0	**58.8**	2.2	39.0	+5.8			-2.31s
Qwen2-0.5B	LLPO	BPO	**48.1**	7.0	44.9	41.7	6.7	**51.6**	**49.0**	9.5	41.5	**52.9**	8.5	38.6	+3.8			-2.31s
Qwen2-1.5B	LLPO	BPO	**55.1**	3.9	41.0	**49.6**	6.4	44.0	**53.0**	4.5	42.5	**60.6**	4.7	34.7	+14.0			-1.47s
Qwen2-7B	LLPO	BPO	**52.6**	1.9	45.5	44.4	4.0	**51.6**	40.5	3.5	**56.0**	**56.8**	1.8	41.4	-0.05			-3.67s
Claude-Haiku-3.5	LLPO	PAS	41.0	3.9	**55.1**	43.3	5.9	**50.8**	44.0	6.0	**50.0**	40.4	3.6	**56.0**	-10.8	0.02s	1.23s (↑ 50×)	-1.42s
GPT-3.5-turbo	LLPO	PAS	25.0	3.8	**71.2**	36.5	7.2	**56.3**	33.0	6.5	**60.5**	44.8	2.5	**52.7**	-25.4			-1.77s
Qwen2-0.5B	LLPO	PAS	35.9	2.6	**61.5**	38.1	10.7	**51.2**	42.0	7.5	**50.5**	43.6	8.3	**48.1**	-12.9			-2.11s
Qwen2-1.5B	LLPO	PAS	43.6	3.8	**52.6**	46.4	4.8	**48.8**	**50.0**	5.5	44.5	**50.9**	3.6	45.5	-0.1			-1.49s
Qwen2-7B	LLPO	PAS	37.2	1.9	**60.9**	32.1	5.2	**62.7**	34.0	2.5	**63.5**	41.2	2.8	**56.0**	-24.7			-2.87s
Claude-Haiku-3.5	LLPO	FIPO	**63.5**	1.2	35.3	**52.8**	2.8	44.4	**67.0**	2.0	31.0	**62.2**	2.2	35.6	+24.8	0.02s	21.12s (↑ 858×)	-20.73s
GPT-3.5-turbo	LLPO	FIPO	43.6	3.8	**52.6**	46.0	4.8	**49.2**	**55.5**	3.0	41.5	**68.7**	1.6	29.7	+10.2			-21.13s
Qwen2-0.5B	LLPO	FIPO	37.8	8.4	**53.8**	33.3	4.8	**61.9**	44.0	7.0	**49.0**	**47.1**	7.0	45.9	-12.1			-20.98s
Qwen2-1.5B	LLPO	FIPO	**50.6**	3.9	45.5	48.8	1.6	**49.6**	**51.0**	3.0	46.0	**60.2**	2.2	37.6	+8.0			-20.02s
Qwen2-7B	LLPO	FIPO	**50.0**	1.3	48.7	38.1	3.2	**58.7**	35.5	5.0	**59.5**	**49.9**	2.4	47.7	-10.3			-21.79s

Agglomerative clustering

Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR	*Opt.Latency*		*ΔTot.Latency*
	A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win		A	B	
Claude-Haiku-3.5	LLPO	ori.	**55.1**	4.5	40.4	**71.8**	4.0	24.2	**74.5**	4.5	21.0	**56.2**	5.0	38.8	+33.3	0.02s	0.0s	+0.88s
GPT-3.5-turbo	LLPO	ori.	**51.9**	2.6	45.5	**50.8**	6.7	42.5	**54.5**	10.0	35.5	**58.8**	3.0	38.2	+13.6			+0.28s
Qwen2-0.5B	LLPO	ori.	**47.4**	5.8	46.8	**46.8**	8.4	44.8	**45.0**	12.0	43.0	**50.1**	10.1	39.8	+3.7			+0.51s
Qwen2-1.5B	LLPO	ori.	**59.6**	7.1	33.3	**49.2**	3.6	47.2	**53.5**	5.0	41.5	**58.0**	4.4	37.6	+15.2			+1.17s
Qwen2-7B	LLPO	ori.	41.0	1.9	**57.1**	46.4	4.8	**48.8**	38.0	6.5	**55.5**	**52.9**	1.4	45.7	-7.2			+0.00s
Claude-Haiku-3.5	LLPO	CoT	**51.3**	3.2	45.5	**57.9**	6.4	35.7	**61.0**	6.0	33.0	**53.5**	6.3	40.2	+17.3	0.02s	0.0s	+0.37s
GPT-3.5-turbo	LLPO	CoT	40.4	3.8	**55.8**	**47.6**	8.0	44.4	42.5	8.5	**49.0**	**60.0**	3.2	36.8	+1.1			+0.09s
Qwen2-0.5B	LLPO	CoT	**49.4**	8.9	41.7	**50.8**	5.2	44.0	**44.5**	12.0	43.5	**50.7**	10.3	39.0	+6.8			-0.50s
Qwen2-1.5B	LLPO	CoT	**62.8**	3.9	33.3	44.4	4.4	**51.2**	**51.5**	4.5	44.0	**61.0**	5.3	33.7	+14.4			+0.01s
Qwen2-7B	LLPO	CoT	35.9	1.9	**62.2**	40.1	5.1	**54.8**	32.5	4.5	**63.0**	48.3	2.8	**48.9**	-18.0			-1.48s
Claude-Haiku-3.5	LLPO	BPO	**56.4**	2.6	41.0	**61.5**	4.0	34.5	**65.5**	3.0	31.5	**60.4**	3.0	36.6	+25.1	0.02s	2.14s (↑ 87×)	-1.60s
GPT-3.5-turbo	LLPO	BPO	**49.4**	2.5	48.1	45.2	6.0	**48.8**	**48.0**	8.0	44.0	**61.0**	1.8	37.2	+6.4			-2.27s
Qwen2-0.5B	LLPO	BPO	42.3	5.1	**52.6**	42.5	7.5	**50.0**	45.0	7.0	**48.0**	**49.3**	9.5	41.2	-3.2			-2.91s
Qwen2-1.5B	LLPO	BPO	**55.1**	4.5	40.4	44.4	4.8	**50.8**	**48.0**	8.0	44.0	**60.4**	3.6	36.0	+9.2			-2.26s
Qwen2-7B	LLPO	BPO	39.7	2.6	**57.7**	44.0	4.8	**51.2**	44.0	5.5	**50.5**	**59.8**	2.6	37.6	-2.4			-3.87s
Claude-Haiku-3.5	LLPO	PAS	26.9	5.2	**67.9**	**46.4**	7.2	**46.4**	45.0	6.0	**49.0**	**48.3**	3.8	47.9	-11.2	0.02s	1.23s (↑ 50×)	-1.59s
GPT-3.5-turbo	LLPO	PAS	30.8	1.3	**67.9**	38.1	5.9	**56.0**	26.5	7.5	**66.0**	**49.7**	2.8	47.5	-23.1			-1.72s
Qwen2-0.5B	LLPO	PAS	33.3	4.5	**62.2**	40.5	5.9	**53.6**	38.5	9.0	**52.5**	44.2	9.3	**46.5**	-14.6			-2.71s
Qwen2-1.5B	LLPO	PAS	44.9	2.5	**52.6**	42.5	3.9	**53.6**	44.0	6.5	**49.5**	**53.1**	3.9	43.0	-3.6			-2.28s
Qwen2-7B	LLPO	PAS	42.3	3.9	**53.8**	39.7	4.0	**56.3**	34.5	3.0	**62.5**	48.5	1.2	**50.3**	-14.5			-3.07s
Claude-Haiku-3.5	LLPO	FIPO	**58.3**	2.0	39.7	**52.0**	3.2	44.8	**68.0**	1.5	30.5	**64.4**	2.7	32.9	+23.7	0.02s	21.12s (↑ 861×)	-20.90s
GPT-3.5-turbo	LLPO	FIPO	45.5	1.9	**52.6**	46.0	3.6	**50.4**	**48.5**	5.5	46.0	**67.9**	1.4	30.7	+7.1			-21.08s
Qwen2-0.5B	LLPO	FIPO	35.3	4.4	**60.3**	41.7	5.1	**53.2**	39.5	6.5	**54.0**	44.0	5.5	**50.5**	-14.4			-21.58s
Qwen2-1.5B	LLPO	FIPO	46.2	3.2	**50.6**	42.9	1.1	**56.0**	44.0	4.5	**51.5**	**54.3**	3.5	42.2	-3.2			-20.81s
Qwen2-7B	LLPO	FIPO	42.3	3.9	**53.8**	47.6	3.2	**49.2**	42.5	5.0	**52.5**	**56.4**	2.2	41.4	-2.0			-21.98s

Table 14: Performance comparison of LLPO with ModernBERT-large under different clustering methods.

These datasets differ fundamentally from standard instruction-following benchmarks in that they demand explicit procedural reasoning.

Unlike the win–tie–lose evaluation protocol used in our main experiments, these benchmarks are evaluated using accuracy-based scoring. A prediction is considered correct if the ground-truth answer is explicitly contained in the model's generated output. For example, if the gold answer is 74, the prediction is counted as correct when the generated text includes "74".

Table 18 reports the accuracy results across different prompt optimization strategies and base LLMs. LLPO achieves better or comparable performance than offline-trained prompt optimizers, including BPO, FIPO, and PAS, on both BBH and GSM8K. At the same time, LLPO preserves its efficiency advantage, as shown in Table 19, where it consistently incurs substantially lower prompt optimization latency than autoregressive optimization methods.

However, both LLPO and offline-trained prompt optimization methods underperform Chain-of-Thought prompting and, in some cases, the original prompts on these reasoning-intensive benchmarks. We attribute this gap to the design goal of SCP, which is intended as a general-purpose prompt optimization framework rather than a method specialized for mathematical or causal reasoning. These results collectively clarify an important boundary of applicability for SCP-based prompt optimization. While LLPO is highly effective for broad instruction-following tasks, it is less suitable for domains that critically depend on explicit reasoning chains rather than structural constraint satisfaction alone.

K-means clustering

Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR	*Opt.Latency*		*ΔTot.Latency*
	A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win		A	B	
Claude-Haiku-3.5	LLPO	ori.	**58.3**	3.2	38.5	**65.5**	5.5	29.0	**77.5**	3.5	19.0	**51.5**	3.7	44.8	+30.4	0.03s	0.0s	+0.91s
GPT-3.5-turbo	LLPO	ori.	46.8	5.1	**48.1**	**55.2**	5.5	39.3	**60.5**	6.5	33.0	**53.1**	4.7	42.2	+13.3			+0.27s
Qwen2-0.5B	LLPO	ori.	**53.2**	7.1	39.7	**46.0**	8.4	45.6	**49.5**	10.0	40.5	**50.9**	10.7	38.4	+8.9			+0.96s
Qwen2-1.5B	LLPO	ori.	**57.7**	6.4	35.9	**50.4**	4.4	45.2	**58.0**	4.5	37.5	**56.4**	3.0	40.6	+15.8			+0.58s
Qwen2-7B	LLPO	ori.	**50.0**	3.2	46.8	**50.8**	5.2	44.0	40.5	5.0	**54.5**	46.5	2.2	**51.3**	-2.2			+0.99s
Claude-Haiku-3.5	LLPO	CoT	**51.9**	0.7	47.4	**57.5**	5.2	37.3	**61.0**	7.0	32.0	**59.0**	3.2	37.8	+18.9	0.03s	0.0s	+0.40s
GPT-3.5-turbo	LLPO	CoT	38.5	3.8	**57.7**	**48.8**	6.8	44.4	**50.0**	8.5	41.5	**54.7**	3.9	41.4	+1.8			+0.07s
Qwen2-0.5B	LLPO	CoT	**53.8**	9.7	36.5	**51.6**	7.9	40.5	**44.5**	11.0	**44.5**	**51.1**	9.1	39.8	+9.9			-0.05s
Qwen2-1.5B	LLPO	CoT	**71.8**	3.8	24.4	**52.8**	3.9	43.3	**58.5**	5.0	36.5	**58.0**	5.8	36.2	+25.2			-0.57s
Qwen2-7B	LLPO	CoT	37.8	7.1	**55.1**	39.3	4.7	**56.0**	28.0	3.0	**69.0**	47.1	2.2	**50.7**	-19.7			-0.49s
Claude-Haiku-3.5	LLPO	BPO	**52.6**	1.9	45.5	**61.9**	4.0	34.1	**70.5**	3.0	26.5	**60.4**	2.8	36.8	+25.6	0.03s	2.14s (↑ 79×)	-1.57
GPT-3.5-turbo	LLPO	BPO	40.4	3.8	**55.8**	**48.4**	5.6	46.0	45.0	8.0	**47.0**	**56.2**	3.4	40.4	+0.2			-2.28s
Qwen2-0.5B	LLPO	BPO	**50.6**	5.8	43.6	46.0	6.8	**47.2**	**46.0**	8.0	**46.0**	**50.7**	9.1	40.2	+4.1			-2.46s
Qwen2-1.5B	LLPO	BPO	**64.1**	3.2	32.7	**51.6**	5.1	43.3	**58.0**	5.5	36.5	**59.4**	4.0	36.6	+21.0			-2.84s
Qwen2-7B	LLPO	BPO	47.4	3.2	**49.4**	40.1	5.1	**54.8**	44.0	2.5	**53.5**	**56.4**	2.0	41.6	-2.9			-2.88s
Claude-Haiku-3.5	LLPO	PAS	39.7	2.0	**58.3**	45.2	4.4	**50.4**	39.5	7.0	**53.5**	43.6	3.9	**52.5**	-11.7	0.03s	1.23s (↑ 45×)	-1.56s
GPT-3.5-turbo	LLPO	PAS	21.8	3.2	**75.0**	36.5	7.2	**56.3**	29.0	5.5	**65.5**	47.7	1.8	**50.5**	-28.1			-1.73s
Qwen2-0.5B	LLPO	PAS	39.1	2.6	**58.3**	41.3	8.7	**50.0**	43.0	7.0	**50.0**	**47.3**	8.1	44.6	-8.1			-2.26s
Qwen2-1.5B	LLPO	PAS	**52.6**	4.5	42.9	44.8	4.4	**50.8**	43.5	7.5	**49.0**	**48.3**	4.2	47.5	-0.3			-2.87s
Qwen2-7B	LLPO	PAS	32.1	2.5	**65.4**	37.7	4.8	**57.5**	30.5	2.0	**67.5**	42.2	3.1	**54.7**	-25.7			-2.08s
Claude-Haiku-3.5	LLPO	FIPO	**53.8**	2.6	43.6	**53.6**	3.1	43.3	**70.0**	3.0	27.0	**65.5**	1.4	33.1	+24.0	0.03s	21.12s (↑ 778×)	-20.86s
GPT-3.5-turbo	LLPO	FIPO	47.4	1.3	**51.3**	**51.2**	2.4	46.4	**51.0**	2.5	46.5	**65.5**	1.6	32.9	+9.5			-21.10s
Qwen2-0.5B	LLPO	FIPO	43.6	3.2	**53.2**	38.5	7.1	**54.4**	40.5	6.5	**53.0**	45.1	4.8	**50.1**	-10.8			-21.13s
Qwen2-1.5B	LLPO	FIPO	**49.4**	1.9	48.7	48.0	1.6	**50.4**	**53.0**	5.0	42.0	**54.5**	2.7	42.8	+5.3			-21.40s
Qwen2-7B	LLPO	FIPO	42.3	3.9	**53.8**	42.5	2.7	**54.8**	38.0	5.0	**57.0**	**49.9**	2.0	48.1	-10.3			-20.99s

Agglomerative clustering

Base LLM	Method		Koala Eval			Self-Instruct Eval			Dolly Eval			Arena-Hard Eval			ΔWR	*Opt.Latency*		*ΔTot.Latency*
	A	B	A win	tie	B win	A win	tie	B win	A win	tie	B win	A win	tie	B win		A	B	
Claude-Haiku-3.5	LLPO	ori.	**50.0**	2.6	47.4	**69.8**	4.8	25.4	**75.5**	3.5	21.0	**53.1**	4.3	42.6	+28.0	0.03s	0.0s	+1.18s
GPT-3.5-turbo	LLPO	ori.	**57.1**	3.8	39.1	**59.1**	5.2	35.7	**57.0**	10.5	32.5	**57.0**	3.2	39.8	+20.8			+0.44s
Qwen2-0.5B	LLPO	ori.	**48.7**	6.4	44.9	44.8	8.0	**47.2**	44.0	11.5	**44.5**	**52.5**	10.3	37.2	+4.1			+0.53s
Qwen2-1.5B	LLPO	ori.	**60.3**	5.7	34.0	**50.0**	4.4	45.6	**49.0**	8.0	43.0	**60.0**	3.6	36.4	+15.1			+1.09s
Qwen2-7B	LLPO	ori.	**50.0**	3.8	46.2	**50.0**	5.2	44.8	40.5	5.5	**54.0**	**47.9**	4.6	47.5	-1.0			+0.10s
Claude-Haiku-3.5	LLPO	CoT	**51.3**	5.1	43.6	**56.3**	6.8	36.9	**57.5**	6.5	36.0	**57.6**	5.0	37.4	+17.2	0.03s	0.0s	+0.67s
GPT-3.5-turbo	LLPO	CoT	46.2	4.4	**49.4**	**48.4**	5.6	46.0	41.5	11.0	**47.5**	**58.4**	3.0	38.6	+3.3			+0.24s
Qwen2-0.5B	LLPO	CoT	**50.6**	8.4	41.0	43.3	7.1	**49.6**	**50.5**	5.0	44.5	**54.7**	8.7	36.6	+6.9			-0.48s
Qwen2-1.5B	LLPO	CoT	**59.6**	3.2	37.2	43.3	2.7	**54.0**	45.0	8.0	**47.0**	**60.2**	5.1	34.7	+8.8			-0.06s
Qwen2-7B	LLPO	CoT	42.3	2.6	**55.1**	38.9	5.1	**56.0**	30.5	4.5	**65.0**	**49.5**	2.8	47.7	-15.7			-1.38s
Claude-Haiku-3.5	LLPO	BPO	**52.6**	1.9	45.5	**61.9**	3.2	34.9	**69.0**	4.0	27.0	**60.4**	3.2	36.4	+25.0	0.03s	2.14s (↑ 68×)	-1.30s
GPT-3.5-turbo	LLPO	BPO	47.4	3.2	**49.4**	**51.6**	4.4	44.0	**46.0**	9.0	45.0	**61.6**	2.6	35.8	+8.1			-2.11s
Qwen2-0.5B	LLPO	BPO	44.9	7.0	**48.1**	40.1	7.1	**52.8**	**46.5**	8.5	45.0	**51.5**	10.7	37.8	-0.2			-2.89s
Qwen2-1.5B	LLPO	BPO	**58.3**	5.8	35.9	46.4	4.8	**48.8**	43.0	7.0	**50.0**	**58.6**	2.6	38.8	+8.2			-2.33s
Qwen2-7B	LLPO	BPO	**50.0**	1.9	48.1	44.0	4.8	**51.2**	38.0	3.5	**58.5**	**56.2**	2.8	41.0	-2.7			-3.77s
Claude-Haiku-3.5	LLPO	PAS	30.8	5.7	**63.5**	45.2	4.0	**50.8**	43.0	4.5	**52.5**	46.5	3.4	**50.1**	-12.9	0.03s	1.23s (↑ 39×)	-1.28s
GPT-3.5-turbo	LLPO	PAS	26.9	1.3	**71.8**	38.9	6.3	**54.8**	26.0	5.5	**68.5**	46.5	2.2	**51.3**	-27.0			-1.57s
Qwen2-0.5B	LLPO	PAS	36.5	7.1	**56.4**	34.1	7.6	**58.3**	41.0	10.5	**48.5**	44.6	8.3	**47.1**	-13.5			-2.69s
Qwen2-1.5B	LLPO	PAS	**48.7**	3.2	48.1	40.5	2.8	**56.7**	42.5	7.5	**50.0**	**51.7**	3.9	44.4	-21.3			-2.36s
Qwen2-7B	LLPO	PAS	47.4	1.3	**51.3**	40.5	4.3	**55.2**	39.5	2.5	**58.0**	46.7	3.0	**50.3**	-10.2			-2.96s
Claude-Haiku-3.5	LLPO	FIPO	**55.1**	0.0	44.9	**52.8**	4.7	42.5	**69.0**	2.0	29.0	**62.2**	1.6	36.2	+21.6	0.03s	21.12s (↑ 676×)	-20.59s
GPT-3.5-turbo	LLPO	FIPO	**53.8**	1.3	44.9	**52.0**	3.6	44.4	**50.0**	4.0	46.0	**68.5**	1.6	29.9	+14.8			-20.93s
Qwen2-0.5B	LLPO	FIPO	39.1	3.2	**57.7**	35.3	7.2	**57.5**	37.5	5.0	**57.5**	44.6	4.9	**50.5**	-16.7			-21.56s
Qwen2-1.5B	LLPO	FIPO	45.5	2.6	**51.9**	43.3	3.1	**53.6**	45.0	3.0	**52.0**	**57.4**	3.2	39.4	-1.4			-20.89s
Qwen2-7B	LLPO	FIPO	48.1	3.2	**48.7**	45.6	2.8	**51.6**	40.0	3.5	**56.5**	**53.5**	3.3	43.2	-3.2			-21.88s

Table 15: Performance comparison of LLPO with DeBERTa-v3-large under different clustering methods.

Claude-Haiku-3.5

Method	tot.latency	opt.latency	inf.latency	ttft	rtt	back.latency
LLPO	6.612s	0.011s	6.601s	1.338s	0.047s	1.292s
BPO	8.223s	2.141s	6.082s	1.422s	0.043s	1.380s
PAS	8.206s	1.227s	6.945s	1.117s	0.044s	1.072s
FIPO	27.515s	21.118s	6.397s	1.212s	0.037s	1.175s
ori.	5.739s	–	5.739s	1.187s	0.038s	1.150s
CoT	6.248s	–	6.248s	1.256s	0.045s	1.211s

GPT-3.5-Turbo

Method	tot.latency	opt.latency	inf.latency	ttft	rtt	back.latency
LLPO	1.826s	0.011s	1.733s	0.496s	0.003s	0.493s
BPO	4.154s	2.141s	1.858s	0.519s	0.003s	0.515s
PAS	3.608s	1.227s	2.347s	0.502s	0.003s	0.499s
FIPO	22.971s	21.118s	1.853s	0.520s	0.003s	0.516s
ori.	1.603s	–	1.603s	0.515s	0.003s	0.512s
CoT	1.800s	–	1.800s	0.535s	0.003s	0.532s

Table 16: Latency breakdown (total latency, optimization latency, inference latency, TTFT, RTT, and backend latency) of different prompt optimization methods. Results for Claude-Haiku-3.5 are shown above and results for GPT-3.5-Turbo are shown below.

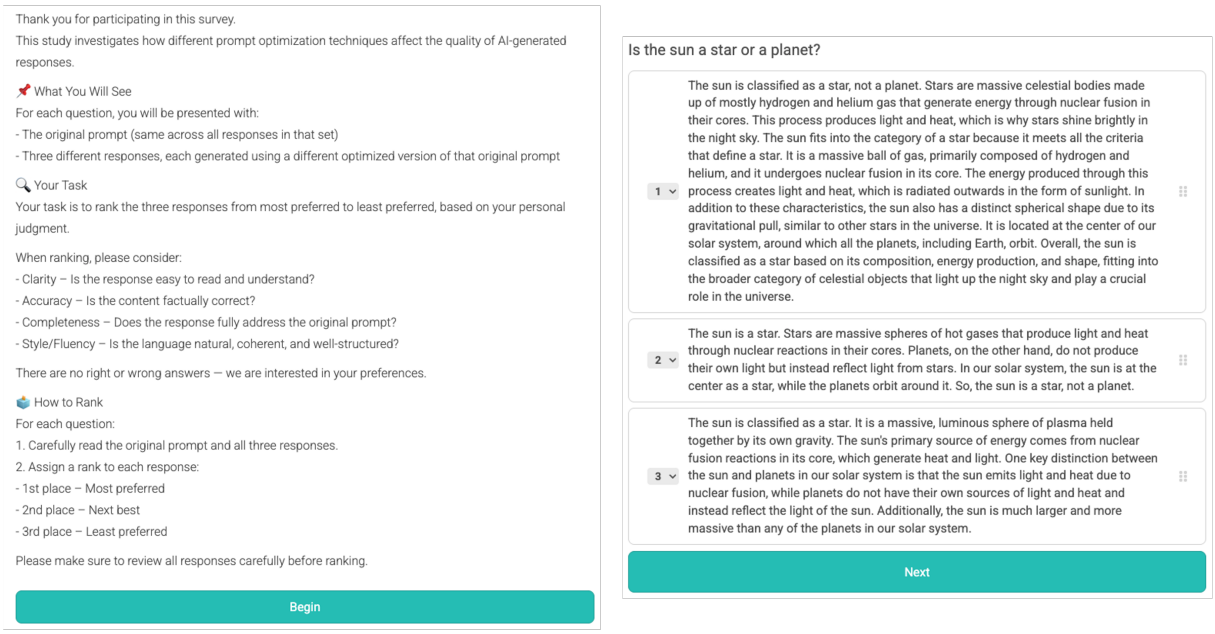

Figure 6: Human evaluation interface, including annotator instructions (left) and the response-ranking interface (right).

	LLPO		**BPO**		**PAS**		**FIPO**		**CoT**		**Ori**	
	input	output	input	output	input	output	input	output	input	output	input	output
Qwen2-0.5B	205.46	257.71	118.0	243.99	131.14	291.98	200.84	258.14	98.94	248.15	91.73	205.39
Qwen2-1.5B	205.46	290.98	118.0	239.71	131.14	300.83	200.84	363.67	98.94	267.08	91.73	213.48
Qwen2-7B	205.46	325.78	118.0	349.65	131.14	403.52	200.84	252.26	98.94	393.99	91.73	322.32

Table 17: Average input and output token counts across four evaluation datasets for different prompting methods (LLPO, BPO, PAS, FIPO, CoT, and Ori) on three Qwen2 models (0.5B, 1.5B, and 7B).

Method	GSM8K						BBH (Casual Judgement)					
	LLPO	BPO	FIPO	PAS	ori.	CoT	LLPO	BPO	FIPO	PAS	ori.	CoT
Claude-Haiku-3.5	86.2	81.3	80.0	86.7	**87.3**	86.9	67.9	59.9	57.8	67.4	69.2	**72.2**
GPT-3.5-Turbo	81.7	73.6	72.5	81.8	81.0	**82.9**	42.2	68.4	47.6	55.1	58.3	**72.7**
Qwen2-0.5B	30.4	37.1	32.4	33.0	40.7	**41.7**	48.7	**54.0**	53.5	47.1	50.3	52.9
Qwen2-1.5B	53.7	54.7	52.8	56.0	61.3	**66.7**	51.3	49.2	51.3	47.1	**57.8**	57.2
Qwen2-7B	84.7	79.5	76.3	82.6	86.3	**88.4**	63.1	**76.5**	51.3	62.0	68.4	70.1

Table 18: Accuracy comparison on the GSM8K and BBH (Causal Judgement) across different prompt optimization methods and base LLMs.

	GSM8K				Causal Judgement			
	LLPO	BPO	PAS	FIPO	LLPO	BPO	PAS	FIPO
avg	0.010s	1.886s	1.156s	16.927s	0.009s	4.975s	1.485s	15.443s
min	0.009s	0.396s	0.486s	3.448s	0.009s	1.175s	0.729s	4.566s
max	0.016s	6.282s	9.492s	300.047s	0.011s	15.312s	26.047s	272.485s

Table 19: Prompt optimization latency statistics of LLPO, BPO, PAS, and FIPO on GSM8K and BBH (Causal Judgement).