

Beyond Prompt Content: Enhancing LLM Performance via Content-Format Integrated Prompt Optimization

Yuanye Liu^{*†} Jiahang Xu^{*◊} Li Lyra Zhang Qi Chen Xuan Feng
 Yang Chen Zhongxin Guo Yuqing Yang Peng Cheng

[†]Fudan University Microsoft Research Asia

Abstract

Large Language Models (LLMs) have shown significant capability across various tasks, with their real-world effectiveness often driven by prompt design. While recent research has focused on optimizing prompt content, the role of prompt formatting—a critical but often overlooked dimension—has received limited systematic investigation. In this paper, we introduce Content-Format Integrated Prompt Optimization (CFPO), an innovative methodology that jointly optimizes both prompt content and formatting through an iterative refinement process. CFPO leverages natural language mutations to explore content variations and employs a dynamic format exploration strategy that systematically evaluates diverse format options. Our extensive evaluations across multiple tasks and open-source LLMs demonstrate that CFPO demonstrates measurable performance improvements compared to content-only optimization methods. This highlights the importance of integrated content-format optimization and offers a practical, model-agnostic approach to enhancing LLM performance. Code is available at <https://github.com/HenryLau7/CFPO>.

1 Introduction

Large Language Models (LLMs) have demonstrated impressive achievements across various domains (OpenAI, 2024a). The effectiveness of LLMs in real-world applications is fundamentally dependent on the design of effective prompts, which serve as an essential interface between human users or developers and the LLM system. Studies have shown that expert-designed prompts could significantly enhance LLM performance (Brown et al., 2020; Wei et al., 2023; Schulhoff et al., 2024).

Despite its significance, manual prompt design is fraught with challenges. LLMs are notoriously

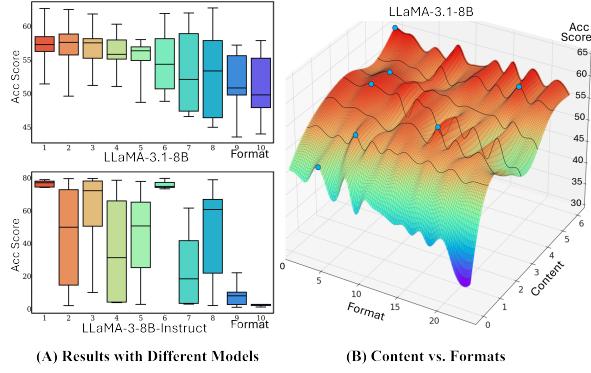


Figure 1: The crucial role of prompt formatting and its interaction with content. (A): Model-specific format sensitivity: LLM performance varies significantly across different prompt formats on GSM8K task. (B): Content-format interdependence: The optimal format for a prompt depends on its content, highlighting the need for joint optimization.

sensitive to subtle variations in prompt phrasing and structure, with performance often differing markedly across models and tasks based on these nuances (Zhuo et al., 2024a; Chatterjee et al., 2024; Jiang et al., 2022; Salinas and Morstatter, 2024; Zhuo et al., 2024b; Sclar et al., 2024). To alleviate these difficulties, automated prompt optimization techniques, often leveraging enhanced LLMs to optimize prompts, have proven to be effective to adapt and refine prompts (Pryzant et al., 2023; Schnabel and Neville, 2024; Yang et al., 2024).

However, existing research primarily focuses on optimizing *prompt content*, while overlooking a critical and unexplored dimension: the ***prompt formatting***. Prompt formatting refers to the arrangement and presentation of prompt content while preserving its semantic meaning. As LLMs are applied to increasingly complex tasks, structuring prompts into distinct components (e.g., instructions, examples, queries) becomes paramount for effectively conveying the desired task and context. Thus, the manner in which a prompt is formatted can significantly impact performance.

Our preliminary investigations (Figure 1) high-

^{*}Equal contribution.

[†]Yuanye Liu did the work during an internship at MSRA.

[◊]Corresponding author: jiahangxu@microsoft.com

light the significant impact of prompt format on LLM performance. We observed that different LLMs exhibit distinct format preferences, with formats performing well on one model sometimes failing on another. This underscores the existence of sophisticated, model-specific format biases (Sclar et al., 2024). Furthermore, even within a single LLM, the optimal format varies depending on the specific prompt content. This complex interplay between content and format suggests that a one-size-fits-all approach to prompt formatting is unlikely to succeed, highlighting the need for joint optimization strategies that consider content and format as interdependent variables.

To address these limitations, we introduce **Content-Format Integrated Prompt Optimization (CFPO)**, an innovative methodology that concurrently optimizes both prompt content and format through an iterative refinement process. CFPO employs distinct optimization strategies tailored to the unique search spaces of content and format. Content optimization is guided by performance feedback and Monte Carlo sampling, leveraging natural language mutations to enhance prompt effectiveness. For format optimization, CFPO explores a discrete set of format options through a dynamic exploration strategy designed to identify optimal formats without requiring prior knowledge.

Specifically, CFPO’s format optimizer draws upon principles of structured thinking and defines a hierarchical template that clearly demarcates content elements from their formatting attributes. This allows for targeted optimization of both intra-component styling (e.g., how an example is presented) (Voronov et al., 2024a; Salinas and Morstatter, 2024) and inter-component arrangement (e.g., the order and connectors between components) (He et al., 2024), adapting to the specific needs of different prompt components and their interactions.

Our primary contributions are threefold:

- We propose CFPO, an innovative approach to simultaneously optimizes prompt content and format using an iterative process.
- We introduce an efficient strategy for dynamic format optimization that iteratively generates and evaluates format candidates using a scoring system to select superior options.
- We conduct extensive evaluations across diverse tasks and open-source LLMs, showing that CFPO consistently enhances model performance in a measurable and effective way.

2 Related Work

2.1 Optimization via LLM

The remarkable capacity of LLMs has been demonstrated in various tasks as optimizers, leveraging their ability to enhance performance, such as code generation (Haluptzok et al., 2023; Zelikman et al., 2024; Askari et al., 2024), tool-making (Cai et al., 2024), and agent system design (Hu et al., 2024). However, recent studies indicate that LLMs face significant challenges in achieving completely automatic optimization. These models often rely on human intervention for designing workflows and struggle with tasks requiring complex decomposition and iterative refinement (Zhang et al., 2024; Li et al., 2024).

2.2 Prompt Optimization Techniques

Automatic Prompt Optimization Automatic prompt optimization plays a crucial role in enhancing the performance of LLMs by refining prompts without requiring human intervention. Various approaches have been explored to search for the optimal prompt, including reinforcement learning (Zhang et al., 2023), Monte Carlo Search (Zhou et al., 2023), Monte Carlo Tree Search (MCTS) (Wang et al., 2024b), and agent-driven frameworks (Wang et al., 2024a; Khattab et al., 2024; WHO, 2023). Notably, feedback-based methods have emerged as a significant approach (Pryzant et al., 2023; Das et al., 2024). These methods iteratively evaluate prompts on a batch of evaluation data, using error cases to guide subsequent mutation and improvement (Pryzant et al., 2023; Das et al., 2024). In each iteration, the prompt optimizer evaluates the prompt on the evaluation set and feeds the incorrectly predicted test cases back to the optimizer to guide improvement in the next mutation step. However, existing automatic prompt optimization techniques often lack the capacity for fine-grained modifications. While (Khattab et al., 2024; Schnabel and Neville, 2024) introduce phrase-level mutations, a systematic approach to optimizing *prompt format* is lacking. This leaves a gap to comprehensively adapt prompt structure for optimal performance.

Prefix Tuning offers an effective strategy for adapting large language models (LLMs) to specific tasks by learning continuous, task-specific vectors that are prepended to the input sequence (Li and Liang, 2021; Wang et al., 2023; Guo et al., 2024; Gu et al., 2022; Wang et al., 2023). While

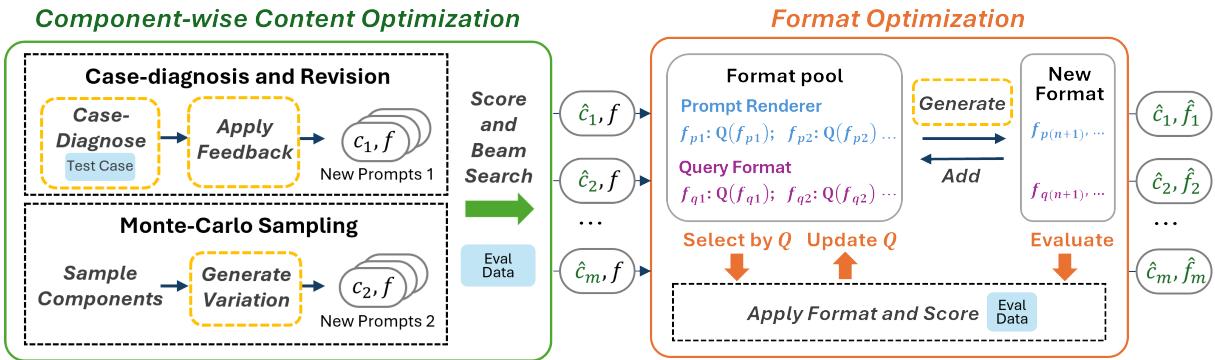


Figure 2: Illustration of the CFPO pipeline within a single iteration round. In the initial Component-wise Content Optimization stage, case-diagnosis and Monte-Carlo sampling are employed for content mutation. Subsequently, the Format Optimization stage identifies the most suitable format for each content candidate. The yellow dashed line indicates where the LLM optimizer is employed to guide the optimization process.

demonstrating impressive performance gains, a key limitation of Prefix Tuning lies in its reliance on access to the LLM’s internal parameters for training. This requirement presents a significant obstacle when working with black-box LLMs accessed through APIs or in resource-constrained environments where full fine-tuning is infeasible.

2.3 Prompt Structure and Sensitivity

Structured Prompting, which organizes prompts into components like instructions and examples, is a widely recommended practice (OpenAI, 2024b; Google, 2024). Frameworks like LangGPT (Wang et al., 2024a) further advance this paradigm by introducing reusable prompt designs inspired by software engineering principles, showcasing the potential of structured approaches (Fernando et al., 2023). While these efforts have primarily focused on optimizing the content and organization of prompt components, less attention has been paid to the impact of *prompt formatting* within this structured context.

Sensitivity of Prompt Variations LLMs exhibit significant sensitivity to prompt variations. Studies have shown that even semantically similar but unseen instructions can lead to performance degradation (Sun et al., 2023; Mizrahi et al., 2024). To address this brittleness, researchers have proposed methods and metrics to systematically evaluate and understand prompt sensitivity (Zhuo et al., 2024b; Chatterjee et al., 2024). These findings underscore the necessity of prompt optimization for robust LLM performance.

Format Sensitivity of Prompt Studies have highlighted the impact of formatting on prompt performance (Salinas and Morstatter, 2024). Sclar et al. (2024) revealed that modifications to separators and spacing within a query could substantially im-

pact performance. He et al. (2024) reveals that the format of prompts significantly impacts GPT-based models’ performance, with no single format excelling universally. Voronov et al. (2024b) focuses on the format of few-shot examples and suggests that it is beneficial to maintain a consistent format across examples. Despite the growing recognition of formatting’s influence, current prompt engineering practices rely heavily on empirical observations and lack systematic design principles.

3 CFPO: Content-Format Integrated Prompt Optimization

The effectiveness of LLMs is profoundly influenced by both the content and format of prompts. To address this, we propose Content-Format Integrated Prompt Optimization (CFPO), a novel framework designed to jointly optimize prompt content and structure for enhanced LLM performance. CFPO, illustrated in Figure 2, employs a dual-optimizer approach in each iteration. The subsequent sections provide a detailed explanation of our methodology, beginning with the structured prompt template we utilize (Section 3.1), followed by a description of the Component-wise Content Optimization (Section 3.2), the Format Optimization (Section 3.3), and finally, the integration of these optimizers into the complete CFPO framework (Section 3.4). Implementation details and meta-prompts are provided in Appendix A.

3.1 Structured Prompt Template

To facilitate fine-grained and targeted optimization, our framework adopts a structured prompt template inspired by prompt engineering guidelines from OpenAI (2024b) and Google (2024). Our template decomposes prompts into distinct functional com-

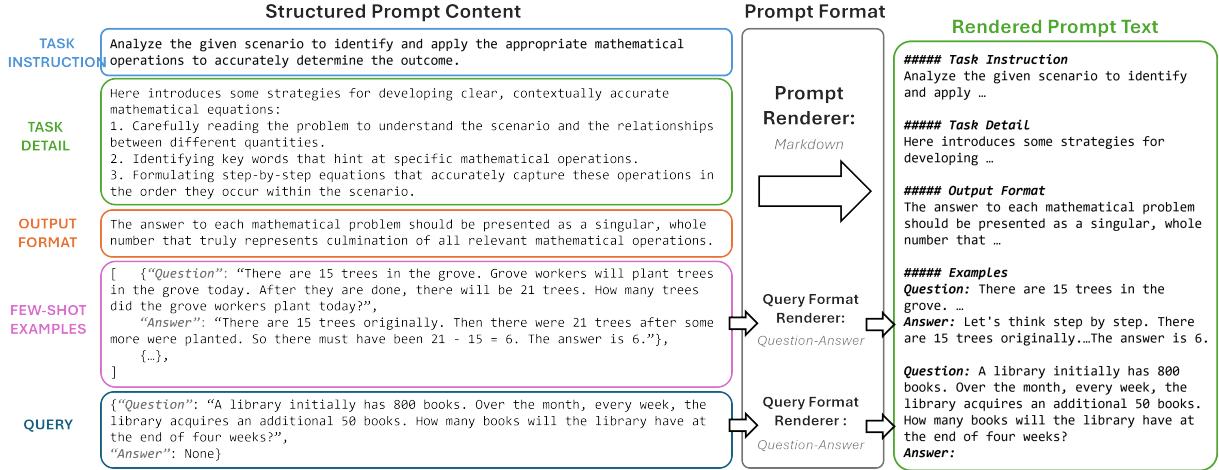


Figure 3: An illustrative example of our Structured Prompt Template. This template systematically organizes the prompt into distinct components, each serving a specific functional role. When formulating a prompt, the template first employs a Query format to present examples and queries, and then integrates all content components via the Prompt Renderer to construct the comprehensive prompt string.

ponents, allowing for both detailed analysis and selective modification. The template distinguishes between content-based and format-based components, providing flexibility and adaptability to diverse user needs. A set of common and generally applicable elements is illustrated in Figure 3.

The **Content-based Components** define the information provided to the LLM, including:

- **Task Instruction** defines the primary goal, guiding the model’s overall behavior.
- **Task Detail** offers supplementary task-specific information, including resolution steps.
- **Output Format** specifies the desired output structure (e.g., JSON, bullet points, etc.).
- **Few-shot Examples** provide specific instances for contextual learning patterns.
- **Query** shows the question or request to be answered by the LLM.

The **Format-based Components** dictate how the prompt is assembled and presented, including:

- **Prompt Renderer** defines how to aggregate all components into a structured prompt.
- **Query Format**: defines how to structure the rendering of examples and queries.

Note that this template is highly adaptable. Users can readily adjust the template by adding or deleting components, or incorporating additional formatting elements (e.g., tables, structured documents) within existing components to suit their specific requirements. By decoupling format from content, this structured template empowers users to perform targeted and precise optimizations, leading to improved prompt effectiveness.

3.2 Component-wise Content Optimization

Building upon existing prompt optimization strategies, which often rely on either feedback-driven refinement or Monte-Carlo sampling (Section 2.2), CFPO introduces a more targeted and efficient approach to content optimization. While feedback-driven methods diagnose weaknesses through failure analysis, and Monte-Carlo sampling diversifies perspectives with semantic variations, CFPO innovates along two key dimensions.

First, CFPO expands the diagnostic phase beyond traditional failure analysis by incorporating *correct cases*. This allows the LLM to identify and reinforce successful aspects of the prompt, complementing error correction. Second, CFPO adopts a *component-wise optimization strategy*. Instead of treating the prompt as a monolithic block, CFPO targets specific content-based components for individual refinement, enabling more precise and effective optimization.

3.3 Format Optimization Design

To efficiently navigate the prompt format space, CFPO incorporates a format optimizer based on a dynamic format pool and an LLM-assisted generation module. The optimizer iteratively explores, evaluates, and refines formatting choices.

3.3.1 Format Pool with Scoring System

The format pool maintains a diverse collection of prompt format configurations, organized hierarchically to represent variations at both macro (e.g., overall prompt structure) and micro (e.g., component-internal rendering) levels. In this work,

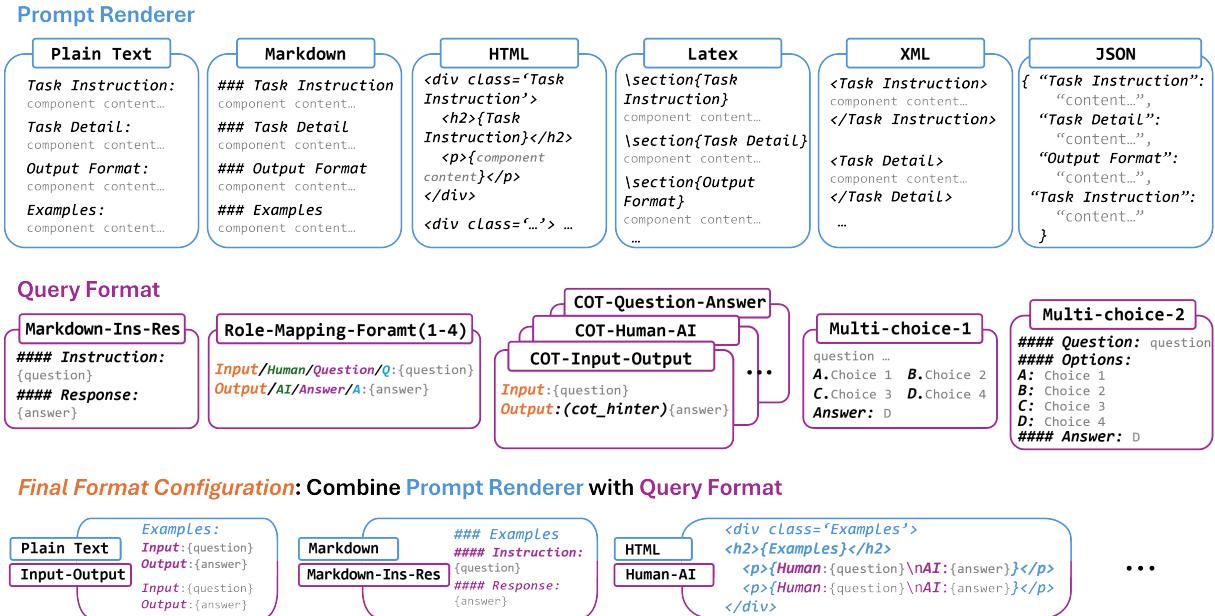


Figure 4: Built-in formats and rendering effects in our initial format pool. The final format configuration is achieved by selecting and combining elements from both the *Prompt Renderer* and the *Query Format* categories.

we prototype this structure with two distinct renderers: a *Prompt Renderer* defining the global structure, and a *Query Format* governing the rendering of individual components, such as examples and queries, as shown in Figure 4.

Each format f in the pool is associated with a performance score, $Q(f)$, which cumulatively reflects its effectiveness across different prompt contents. When a format f is evaluated on a new content c , its score is updated as follows: $Q(f) \leftarrow Q(f) + m(c, f)$, where $m(c, f)$ is a task-specific metric function. For example, in reasoning tasks, $m(\cdot)$ represents the average accuracy (0 or 1) on an evaluation dataset. The number of evaluations for each format, $N(f)$, is also tracked to enable score normalization and fair comparison.

To initiate the format exploration process, we define an initial format search space, \mathcal{F} , consisting of a set of predefined and commonly used formats (Figure 4), along with diverse variations that introduce subtle changes (e.g., spacing, punctuation, special symbols). This initial format pool serves as the foundation for subsequent exploration.

3.3.2 LLM-assisted Format Generation

To overcome the limitations of a static format pool and promote continuous adaptation, we introduce an LLM-based format generator, LLM_{f_gen} . This module autonomously generates novel formats by leveraging information about the existing formats in the pool and their performance characteristics. By dynamically creating new formats, we aim to es-

cape local optima and discover potentially superior formatting strategies.

In each optimization round, the format generator is provided with the existing formats and their normalized performance scores, $\frac{Q(f)}{N(f)}$, and tasked with creating new formats or variations that might improve performance. This iterative process not only diversifies the format pool but also ensures that our system can adapt to and incorporate a wide range of formats, thereby enhancing its utility and effectiveness. Additional details and generated examples about the format generation process are available in Appendix A.3 and Appendix D.1.

3.3.3 Search Format via Format Optimizer

For each content candidate generated by the content optimizer, the format optimizer aims to identify the most appropriate format from format pool. To navigate the balance between exploring new formats and exploiting known effective ones, we implemented the Upper Confidence Bounds applied to Trees (UCT) algorithm (Kocsis and Szepesvári, 2006). The UCT algorithm employs a selection criterion given by:

$$UCT(f) = \frac{Q(f)}{N(f)} + \alpha \sqrt{\frac{\sum_f N(f)}{N(f)}} \quad (1)$$

where α serves as a balancing hyper-parameter, adjusting the trade-off between exploration and exploitation.

The overall process, outlined in Algorithm 1, selects $2k$ formats for evaluation in each optimiza-

Algorithm 1 Searching Optimal Format Given a Prompt Candidate

Input: $p_0 = (c_0, f_0)$: initial prompt, $p = (c, \cdot)$: current prompt candidate (with content c), \mathcal{F} : dynamic format pool, k : number of formats, $m(\cdot)$: evaluation metric, \mathcal{D} : evaluation data.

- 1: Initialize: $Q(f) \leftarrow m(c_0, f)$, $N(f) \leftarrow 1$ for all $f \in \mathcal{F}$
- 2: **Format Selection:** $\mathcal{F}_{select} \leftarrow \{f \in \mathcal{F} : f \text{ is in the top } k \text{ w.r.t. } UCT(f)\}$
- 3: **Format Generation:**
- 4: **for** each $i = 0, 1, \dots, k$ **do**
- 5: Generate format: $f_{new} \leftarrow LLM_{f_gen}(\mathcal{F})$
- 6: Collect f_{new} to \mathcal{F}_{gen} , and add f_{new} to \mathcal{F}
- 7: **end for**
- 8: **Format Evaluation:**
- 9: **for** each $f \in \mathcal{F}_{select} \cup \mathcal{F}_{gen}$ **do**
- 10: Evaluate $m(c, f)$ with dataset \mathcal{D}
- 11: $Q(f) \leftarrow Q(f) + m(c, f)$
- 12: $N(f) \leftarrow N(f) + 1$
- 13: Update $UCT(f)$ by Eq. 1
- 14: **end for**
- 15: $\hat{f} \leftarrow \arg \max_{f \in \mathcal{F}_{select} \cup \mathcal{F}_{gen}} m(c, f)$

Output: The optimal format \hat{f} for content c

tion round: k existing formats from the pool (based on UCT score), and k new formats generated by the LLM_{f_gen} . The selected formats from both the existing pool (\mathcal{F}_{select}) and the newly generated pool (\mathcal{F}_{gen}) are then evaluated using a predefined metric function $m(\cdot)$, and the best-performing format among the tested candidates will be identified. The result is then incorporated into the pool for future iterations. By iteratively evaluating formats, the format optimizer ensures a balance between exploring new formats and refining current ones, converging to the best format configuration.

3.4 Integrated Optimizer Design

CFPO integrates content and format optimization within an iterative loop, as illustrated in Figure 2. In each iteration, the content optimizer proposes candidate prompts, and the format optimizer identifies the most effective format for each candidate. This ensures that each prompt benefits from the most effective formatting. Furthermore, effective formats can improve the diversity and performance of content candidates, thereby helping content optimizer for beam search in the next iteration.

In summary, the content and format optimizers

work in tandem, leveraging the LLM to enable rapid adaptation and customization. This iterative and collaborative process optimizes both the content and the format of prompts, leading to significant improvements in prompt quality.

4 Experiments

4.1 Experimental Setups

Models. We selected four open-source Large Language Models (LLMs) as our primary evaluation targets, including two foundational models, Mistral-7B-v0.1 (Jiang et al., 2023) and LLaMA-3.1-8B (Meta, 2024b), as well as two instruction-tuned models, LLaMA-3-8B-Instruct (Meta, 2024a) and Phi-3-Mini-Instruct (Microsoft, 2024). For content mutation and format generation during the optimization process, we employed GPT-4 (2024-05-01-preview) (OpenAI, 2024a).

Datasets. Our evaluation benchmark was designed to comprehensively assess model performance across a range of task types and difficulty levels, emphasizing diverse query formats. Specifically, we employed the following tasks:

- **Classification:** The *Implicatures* task from the Big-Bench benchmark (bench authors, 2023).
- **Multiple-choice:** ARC-Challenge (Clark et al., 2018) task.
- **Reasoning:** GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021; Lightman et al., 2023) requiring complex reasoning abilities.

Baselines. We compared CFPO with several commonly used and popular baselines. GrIPS (Prasad et al., 2023) performs syntactic phrase-level edits in instruction, representing a non-LLM-based optimization approach. APE (Zhou et al., 2023) and ProTeGi (Pryzant et al., 2023) both employ LLM to optimize prompt content, but differ in mutation strategy. APE adopts an instruction induction approach, while ProTeGi leverages test cases feedback with LLM to guide the mutation process. SAMMO (Schnabel and Neville, 2024) introduces a structured framework that incorporates a preliminary format mutation strategy, which relies on random selection from a predefined format pool. All methods were evaluated using consistent experimental configurations to ensure a fair comparison. We also report common baseline prompts for reasoning tasks, including 8-shot for GSM8K and 4-shot for MATH500.

Method	Mistral-7B-v0.1	LLaMA-3.1-8B	LLaMA-3-8B-Instruct	Phi-3-Mini-Instruct
<i>Big-Bench Classification</i>				
Baseline	56.00	64.00	70.00	54.00
GRIPS	86.00	67.00	84.00	69.00
APE	73.00	65.00	60.00	63.00
ProTeGi	83.00	81.00	82.00	76.00
SAMMO	86.00	80.00	86.00	78.00
CFPO (Ours)	94.00	90.00	91.00	87.00
<i>ARC-Challenge</i>				
Baseline	67.15	73.81	75.94	84.39
GRIPS	77.05	77.90	79.61	87.46
APE	75.85	77.05	78.67	87.63
ProTeGi	76.54	77.22	79.86	87.54
SAMMO	77.22	77.13	79.86	87.03
CFPO (Ours)	79.35	78.50	80.63	88.23
<i>GSM8K</i>				
Baseline (1-shot cot)	36.85	50.03	74.00	83.45
Baseline (8-shot cot)	38.21	51.02	73.46	85.75
GRIPS	39.04	50.27	74.53	83.47
APE	40.33	52.39	75.13	83.85
ProTeGi	45.72	54.74	75.36	84.84
SAMMO	43.82	54.74	75.89	84.76
CFPO (Ours)	53.22	63.38	80.74	89.16
<i>MATH-500</i>				
Baseline (1-shot cot)	4.60	10.58	12.20	12.60
Baseline (4-shot cot)	10.20	23.40	14.00	40.40
GRIPS	13.40	15.80	23.60	10.80
APE	11.60	12.80	22.80	30.60
ProTeGi	10.80	17.00	18.40	28.80
SAMMO	12.20	15.40	25.80	42.40
CFPO (Ours)	14.80	26.99	33.33	44.20

Table 1: Main results of CFPO and state-of-the-art methods on four datasets.

Implementation Details. To evaluate the generated prompts, we sample subsets from the training split for each benchmark (sizes: 50, 500, 500, and 300 examples respectively). Beam search (budget of 8) is used during prompt mutations. Each experiment was capped at 20 iterations. Early stopping was implemented, halting the process if performance plateaus. The number of prompt components the LLM could modify decreased linearly from 4 to 1 over the iterations (see Appendix B.1). We start with a single in-context example without any further instruction as the initial prompt for each model and task, except for GrIPS which requires an initial instruction. Detailed parameter settings and procedure are in the Appendix B.

4.2 Main Results

Table 1 summarizes the performance of CFPO compared with several state-of-the-art methods across four datasets. The results highlight the superior performance of CFPO, significantly outperforming the baseline prompt and competing methods.

The effectiveness of CFPO is particularly evident when compared to methods like GRIPS, which relies solely on phrase-level mutations, yield-

ing only marginal improvements. This highlights the importance of iterative feedback for effective prompt refinement, a feature shared by ProTeGi, SAMMO, and CFPO. Furthermore, CFPO incorporates integrated format optimization that contributes significantly to the performance gains, particularly for previously challenging tasks.

The benefits of CFPO are particularly pronounced in reasoning tasks, such as GSM8K and MATH, known for their sensitivity to prompt structure. While improvements are observed on both datasets, the impact is more noticeable on GSM8K compared to the more complex MATH dataset, suggesting that task difficulty can moderate the attainable performance gains from prompt optimization.

Beyond performance metrics, we further evaluate the stability of CFPO and the computational cost, as detailed in Appendix C. Examples of optimal prompts are detailed in Appendix E.

4.3 Ablation Study

Impact of the CFPO Pipeline Components. To understand the contribution of each component in CFPO, we compared the full CFPO approach against the following ablated variants: (1) CFPO_f

Task	Method	Llama3.1	Llama3-Ins
BBC	ProTeGi	81.00	82.00
	CFPO _f	83.00	86.00
	CFPO _c	85.00	85.00
	CFPO _{c+f}	88.00	89.00
	CFPO	90.00	91.00
GSM8K	ProTeGi	54.74	75.36
	CFPO _f	52.46	76.65
	CFPO _c	58.07	77.71
	CFPO _{c+f}	61.94	79.30
	CFPO	63.38	80.74

Table 2: Performance comparison of the full CFPO pipeline against ablated variants with format-only (CFPO_f), content-only (CFPO_c), and sequential content-then-format (CFPO_{c+f}) optimization. CFPO_c also compared to ProTeGi (Pryzant et al., 2023).

Task	Method	Llama3.1	Llama3-Ins
<i>Impact of Format Generation</i>			
BBC	w/o Format Gen	88.00	87.00
	with Format Gen	90.00	91.00
GSM8K	w/o Format Gen	62.70	78.85
	with Format Gen	63.38	80.74
<i>Different Format Selection Strategies</i>			
BBC	Random	85.00	87.00
	UCT($\alpha = 0$)	86.00	88.00
	UCT(ours)	90.00	91.00
GSM8K	Random	62.40	78.82
	UCT($\alpha = 0$)	63.23	79.08
	UCT(ours)	63.38	80.74

Table 3: Ablation of format generation and comparison of format selection strategies.

(format-only): optimizes the prompt format while holding content constant, (2) CFPO_c (content-only): optimizes the prompt content while holding format constant, and (3) CFPO_{c+f} (sequential): performs iterative content optimization followed by a separate, single-step format optimization. We also include ProTeGi (Pryzant et al., 2023) as a baseline. While CFPO_c shares the goal of content optimization with ProTeGi, CFPO_c incorporates correct cases in diagnosis and component-wise mutation as key innovations. Table 2 presents the results of this analysis. CFPO_c significantly outperforms ProTeGi, demonstrating the effectiveness of our content optimization strategy. Furthermore, the ablated variants all underperform compared to the complete CFPO pipeline. These results underscore the interdependence of content and format in prompt optimization, highlighting the importance of joint optimization for best performance.

Analysis of Format Generation. We compared the full CFPO approach against variant that uses format from initial format pool without using LLM

Optimizer	Mistral	Llama3.1	Llama3-Ins	Phi-3
Qwen2.5-14B	50.49	58.76	80.14	88.48
GPT-4 (CFPO)	53.22	63.38	80.74	89.16

Table 4: Performance of CFPO using different optimizer models.

for generation. As shown in Table 3, the version with format generation consistently outperforms the variant using only the initial pool. This highlights the benefit of our format exploration mechanism in expanding the prompt space. (See Appendix D.2 for the ablation study with different format generation models.)

Analysis of Format Selection Strategies. We further examined the effectiveness of our UCT-based format selection strategy. We compared it against two baselines: random selection from the format pool and a greedy selection approach (equivalent to setting $\alpha = 0$ in Eq. (1), disabling exploration). Table 3 shows that CFPO with UCT-based selection consistently achieves the best performance across all settings, demonstrating the efficacy of balancing exploration and exploitation in format searching.

Exploring Different Optimizer Models. While GPT-4 was used as the primary optimizer model for CFPO, we also investigated the feasibility of using a more accessible model , Qwen2.5-14B-Instruct. Table 4 presents the performance of CFPO with Qwen2.5-14B-Instruct as the optimizer. The results are encouraging, demonstrating that CFPO can be effectively adapted to smaller-sized open-sourced models, albeit with some performance trade-offs, increasing its potential for broader applicability.

5 Conclusion

This paper introduces Content-Format Integrated Prompt Optimization (CFPO), an innovative methodology that concurrently optimizes both prompt content and format. By leveraging a structured prompt template, CFPO decouples these elements, enabling integrated optimization and addressing a significant gap in current research that largely overlooks the critical influence of format. Our results demonstrate the substantial significant influence of format on LLM performance, underscoring the necessity of a joint optimization approach. These findings emphasize the importance of integrating content and format considerations in prompt engineering. CFPO represents a significant advancement, empowering developers to design effective and robust prompts and unlocking the full potential of LLMs across diverse applications.

Limitations While the proposed method demonstrates promising results, there are several limitations worth noting. First, the effectiveness of the approach is task- and model-dependent. While the method generates promising prompts for specific tasks and models, it may not generalize as effectively to others—particularly tasks that are less sensitive to prompt structure or models that already possess strong reasoning capabilities, thereby limiting its broader applicability. Moreover, the iterative nature of the optimization process, with multiple mutation strategies, introduces computational complexity, which could hinder scalability in resource-constrained environments. Finally, we acknowledge the inherent difficulty in definitively separating content and format, especially in real-world scenarios where structured data can be embedded within content. Future work will focus on addressing these limitations, including exploring the application of CFPO to complex agentic tasks and improving the stability and scalability of the optimization process.

References

- Arian Askari, Christian Poelitz, and Xinye Tang. 2024. [Magic: Generating self-correction guideline for in-context text-to-sql](#).
- BIG bench authors. 2023. [Beyond the imitation game: Quantifying and extrapolating the capabilities of language models](#). [Transactions on Machine Learning Research](#).
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared 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 M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In [Proceedings of the Annual Conference on Neural Information Processing Systems \(NeurIPS\)](#).
- Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. [Large language models as tool makers](#).
- Anwoy Chatterjee, H S V N S Kowndinya Renduchintala, Sumit Bhatia, and Tanmoy Chakraborty. 2024. [Posix: A prompt sensitivity index for large language models](#). [Preprint](#), arXiv:2410.02185.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. [Think you have solved question answering? try arc, the ai2 reasoning challenge](#). [Preprint](#), arXiv:1803.05457.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reichiro Nakano, et al. 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.
- Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2023. [Promptbreeder: Self-referential self-improvement via prompt evolution](#). [Preprint](#), arXiv:2309.16797.
- Google. 2024. [Prompting guide 101](#). <https://workspace.google.com/resources/ai/writing-effective-prompts/>.
- Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2022. [PPT: Pre-trained prompt tuning for few-shot learning](#). In [Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics \(Volume 1: Long Papers\)](#), pages 8410–8423, Dublin, Ireland. Association for Computational Linguistics.
- Yanhui Guo, Shaoyuan Xu, Jinmiao Fu, Jia Liu, Chaosheng Dong, and Bryan Wang. 2024. [Q-tuning: Queue-based prompt tuning for lifelong few-shot language learning](#). [Preprint](#), arXiv:2404.14607.
- Patrick Halupczok, Matthew Bowers, and Adam Tauman Kalai. 2023. Language models can teach themselves to program better.
- Jia He, Mukund Rungta, David Koleczek, Arshdeep Sekhon, Franklin X Wang, and Sadid Hasan. 2024. Does prompt formatting have any impact on llm performance? [Preprint](#), arXiv:2411.10541.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. [arXiv preprint arXiv:2103.03874](#).
- Shengran Hu, Cong Lu, and Jeff Clune. 2024. [Automated design of agentic systems](#). [Preprint](#), arXiv:2408.08435.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. [Mistral 7b](#). [Preprint](#), arXiv:2310.06825.

- Ellen Jiang, Kristen Olson, Edwin Toh, Alejandra Molina, Aaron Donsbach, Michael Terry, and Carrie J Cai. 2022. *Promptmaker: Prompt-based prototyping with large language models*. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, CHI EA '22, New York, NY, USA*. Association for Computing Machinery.
- Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. 2024. Dspy: Compiling declarative language model calls into self-improving pipelines. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.
- Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. *Transactions on Machine Learning Research*.
- Xiang Lisa Li and Percy Liang. 2021. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.
- Hunter Lightman, Vineet Kosaraju, Yura Burda, Harry Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.
- Meta. 2024a. Introducing meta llama3: The most capable openly available llm to date.
- Meta. 2024b. The llama 3 herd of models. *Preprint, arXiv:2407.21783*.
- Microsoft. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint, arXiv:2404.14219*.
- Moran Mizrahi, Guy Kaplan, Dan Malkin, Rotem Dror, Dafna Shahaf, and Gabriel Stanovsky. 2024. *State of what art? a call for multi-prompt llm evaluation*. *Preprint, arXiv:2401.00595*.
- OpenAI. 2024a. Gpt-4 technical report. *Preprint, arXiv:2303.08774*.
- OpenAI. 2024b. Prompt generation. <https://platform.openai.com/docs/guides/prompt-generation/>.
- Archiki Prasad, Peter Hase, Xiang Zhou, and Mohit Bansal. 2023. Grips: Gradient-free, edit-based instruction search for prompting large language models. *Preprint, arXiv:2203.07281*.
- Reid Pryzant, Dan Iter, Jerry Li, Yin Tat Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, page 7957–7968.
- Abel Salinas and Fred Morstatter. 2024. The butterfly effect of altering prompts: How small changes and jailbreaks affect large language model performance. *Preprint, arXiv:2401.03729*.
- Tobias Schnabel and Jennifer Neville. 2024. Symbolic prompt program search: A structure-aware approach to efficient compile-time prompt optimization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–686.
- Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yin-heng Li, Aayush Gupta, HyoJung Han, Sevien Schulhoff, Pranav Sandeep Dulepet, Saurav Vidyadhara, Dayeon Ki, Sweta Agrawal, Chau Pham, Gerson Kroiz, Feileen Li, Hudson Tao, Ashay Srivastava, Hevander Da Costa, Saloni Gupta, Megan L. Rogers, Inna Goncenarenco, Giuseppe Sarli, Igor Galynker, Denis Peskoff, Marine Carpuat, Jules White, Shyamal Anadkat, Alexander Hoyle, and Philip Resnik. 2024. The prompt report: A systematic survey of prompting techniques.
- Melanie Sclar, Yejin Choi, Yulia Tsvetkov, and Alane Suhr. 2024. Quantifying Language Models' Sensitivity to Spurious Features in Prompt Design or: How I learned to start worrying about prompt formatting. In *Proceedings of the International Conference on Learning Representations (ICLR)*.
- Jiuding Sun, Chantal Shaib, and Byron C. Wallace. 2023. Evaluating the zero-shot robustness of instruction-tuned language models. *Preprint, arXiv:2306.11270*.
- Anton Voronov, Lena Wolf, and Max Ryabinin. 2024a. Mind your format: Towards consistent evaluation of in-context learning improvements. *Preprint, arXiv:2401.06766*.
- Anton Voronov, Lena Wolf, and Max Ryabinin. 2024b. Mind your format: Towards consistent evaluation of in-context learning improvements. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 6287–6310, Bangkok, Thailand. Association for Computational Linguistics.
- Ming Wang, Yuanzhong Liu, Xiaoyu Liang, Songlian Li, Yijie Huang, Xiaoming Zhang, Sijia Shen, Chaofeng Guan, Daling Wang, Shi Feng, Huaiwen Zhang, Yifei Zhang, Minghui Zheng, and Chi Zhang. 2024a. Langgpt: Rethinking structured reusable prompt design framework for llms from the programming language.

Xinyuan Wang, Chenxi Li, Zhen Wang, Fan Bai, Haotian Luo, Jiayou Zhang, Nebojsa Jojic, Eric P Xing, and Zhiting Hu. 2024b. Promptagent: Strategic planning with language models enables expert-level prompt optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogério Feris, Huan Sun, and Yoon Kim. 2023. Multitask prompt tuning enables parameter-efficient transfer learning. *Preprint*, arXiv:2303.02861.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

WHO. 2023. Auto-gpt. <https://github.com/Significant-Gravitas/AutoGPT>.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V. Le, Denny Zhou, and Xinyun Chen. 2024. Large Language Models as Optimizers. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation. In *Conference On Language Model (COLM)*.

Jiayi Zhang, Jinyu Xiang, Zhaoyang Yu, Fengwei Teng, Xionghui Chen, Jiaqi Chen, Mingchen Zhuge, Xin Cheng, Sirui Hong, Jinlin Wang, Bingnan Zheng, Bang Liu, Yuyu Luo, and Chenglin Wu. 2024. Aflow: Automating agentic workflow generation.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E. Gonzalez. 2023. Tempera: Test-time prompting via reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

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. *Preprint*, arXiv:2211.01910.

Jingming Zhuo, Songyang Zhang, Xinyu Fang, Haodong Duan, Dahua Lin, and Kai Chen. 2024a. ProSA: Assessing and understanding the prompt sensitivity of LLMs. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1950–1976, Miami, Florida, USA. Association for Computational Linguistics.

Jingming Zhuo, Songyang Zhang, Xinyu Fang, Haodong Duan, Dahua Lin, and Kai Chen. 2024b. Prosa: Assessing and understanding the prompt sensitivity of llms. *Preprint*, arXiv:2410.12405.

A Appendix: Detailed Optimization Process and Meta-Prompts

A.1 Meta-Prompt Header Setup

At the beginning of the prompt, we introduce the task and provide a detailed explanation of the prompt’s components, followed by the current version of the prompt. Below is the structure of the meta-prompt header, where placeholders are denoted in [ALL CAPS]:

I'm trying to write a prompt to [TASK INTENTION].

The current prompt consists of several key components, including:
[DESCRIPTION OF COMPONENTS]

The complete prompt is as follows:
"""[CURRENT PROMPT]"""

A.2 Content Optimization

A.2.1 Case-diagnosis and Revision

As described in Section 3.4, content optimization is achieved through an iterative process of case-diagnosis and feedback guided mutation. To facilitate this process, we utilize three distinct meta-prompts, each tailored to a specific task within content optimization.

Case Diagnosis Meta-Prompt. This meta-prompt analyzes the current prompt’s performance against a set of test cases. It identifies areas for improvement and suggests specific modifications for the next iteration.

[META PROMPT HEADER]

Upon evaluating the current prompt, this prompt gets the following examples wrong:
[INCORRECT CASES]

Meanwhile, this prompt gets the following examples correct:
[CORRECT CASES]

Please review the provided examples of correct and incorrect answers, and identify [NUM OF DIAGNOSED COMPONENTS] specific area for improvement in the prompts. Each suggestion should focus on A SPECIFIC segment of the prompt that needs optimization. For each suggestion, provide a comprehensive explanation that encapsulates all the evaluation results. If you believe the EXAMPLES segment needs improvement, you may suggest one example that can be added, removed, or altered to enhance the EXAMPLES segment based on the examples given. If you think there is no need for improvement, do not return any prompt segment.

Please encapsulate each suggestion using the following format:

<START>

```
<Prompt segment: [Segment name]>  
[Suggestion goes here]  
<END>
```

Feedback Application Meta-Prompt. Based on the diagnosis, this meta-prompt generates targeted textual changes to enhance the prompt’s performance. It directly modifies the identified components of the prompt based on the feedback.

[META PROMPT HEADER]

The existing [COMPONENT NAME] segment contains:
[CURRENT CONTENT FOR THE COMPONENT]

Here are some suggestions for improving the [COMPONENT NAME] segments:
[GENERATED DIAGNOSES]

Based on the above information, I wrote [NUMBER OF GENERATED CONTENT] distinct and improved versions of the [COMPONENT NAME] segment within the prompt.

Each revised segment is encapsulated between <START> and <END>. In case this segment is an empty string, generate a suitable one referring to the suggestion.

The [NUMBER OF GENERATED CONTENT] revised [COMPONENT NAME] segments are:

Feedback Application Meta-Prompt (for Examples). This meta-prompt specifically handles the optimization of few-shot examples. It revises examples by adding, deleting, or modifying one single instances, ensuring that the in-context learning process is effective.

[META PROMPT HEADER]

The existing EXAMPLES segment contains:
[CURRENT IN-CONTEXT EXAMPELS IN PROMPT]

Here are some suggestions for enhancing the EXAMPLES segment:
[GENERATED DIAGNOSES]

Based on the above information, I have crafted [NUMBER OF GENERATED EXAMPLES] improved version of the EXAMPLES segment within the prompt. Each revision represents ONLY ONE of the following specific actions:

1. Addition: Incorporating one new example into the existing set.
2. Deletion: Eliminating one single example from the current set.
3. Modification: Changing the content of an example while maintaining its contextual relevance.

Please present the results without indicating which action was taken. Each refined EXAMPLES segment is marked by <START> and <END>.

The [NUMBER OF GENERATED EXAMPLES] revised EXAMPLES are:

A.2.2 Monte-Carlo Sampling

Monte-Carlo Sampling Meta-Prompt explores a wider range of semantically equivalent yet syntactically varied instructions, enhancing the chances of discovering more effective prompts.

[META PROMPT HEADER]

Please create a different version of [COMPONENT NAME] segment without changing its semantic meaning. In case this segment is an empty string, generate a suitable one. The existing [COMPONENT NAME] segment contains:
[CURRENT CONTENT FOR THE COMPONENT]

The varied [COMPONENT NAME] segment is as follows:

Monte-Carlo Sampling Meta-Prompt (for Examples) refines few-shot examples by strategically adding, deleting, or modifying single instances to ensure their effectiveness.

[META PROMPT HEADER]

The existing EXAMPLE set contains:
[CURRENT IN-CONTEXT EXAMPELS IN PROMPT]

Please generate a variation of the EXAMPLES set within the prompt while keeping the semantic meaning. The revision shoud represent ONLY ONE of the following specific actions:

1. Addition: Incorporating one new example into the existing set.
2. Deletion: Eliminating one single example from the current set.
3. Modification: Changing the content of an example while maintaining its contextual relevance.

Please present the results without indicating which action was taken. The varied EXAMPLES segment is as follows:

A.3 Format Generation

Our format generation process is a two-step procedure designed to create diverse and effective prompt formats. We focus on generating two key components of a prompt’s format: the *Prompt Renderer* and the *Query Format*. The appendix presents examples of the format generated using this pipeline.

Step 1: Format Description Generation. For each component (i.e., *Prompt Renderer* and the *Query Format*), we first generate a natural language description of the format, alongside an example of how this format would render a sample input. This description acts as a blueprint, guiding the subsequent code generation. We utilize a meta-prompt to instruct an LLM to perform this task. The meta-prompt takes existing format examples as context and generates new format descriptions along with

rendered results. As an illustrative example, here is a conceptual outline of the meta-prompt employed for generating new *Query Format* descriptions:

[META PROMPT HEADER]

We have some preset QUERY_FORMAT candidates, here are our whole search pool:
[ALL EXISTING QUERY FORMATS DESCRIPTION]

Here are two examples from our QUERY_FORMAT candidates as for your reference:
<Format name: Question-Answer>
[RENDERED EXAMPLE 1]

<Format name: Instruction-Response>
[RENDERED EXAMPLE 2]

Please generate ONE new format for the QUERY_FORMAT segment, its description and render the provided example using this new format. The new format could either be a completely new format or a variation of an existing format.

If you choose to generate a completely new format, please ensure that the new format is conventional, structured, and aligned with commonly used query formats. Avoid overly creative or unconventional formats that deviate significantly from standard practices. The new format should be distinct from the existing formats.

The variation can focus on two parts, Casing and Separator:

Casing refers to both the capitalization of the text (e.g., `f(x) = x.title()`, `f(x) = x.upper()`, `f(x) = x.lower()`) and the specific wording or phrasing used (e.g., changing "question" to "instruction" or "input").

Separator: the punctuation or symbols used to separate the question and answer, there are some candidates as for your reference `{'', ' ', '\\n', '--', ';\\n', '||', '<sep>', '\\n', ':', '.'}`.

Note that focus solely on the format itself without altering the content of the question and answer. The format should remain focused on the existing structure (e.g., Question/Answer or Instruction/Response) without modifying the content or introducing any new sections. Avoid the use of underlines or any unconventional formatting styles among words. The format name should only include alphanumeric characters and underscores. Special characters such as `|`, `!`, `#`, `@`, and spaces should be avoided.

Please encapsulate the new query format using the following format:

<START>
<Format name: [format name]>
<Description: [format description]>
[The example rendered by the newly generated format]
<END>

Step 2: Format Code Generation. Based on the natural language description and rendered example produced in Step 1, we subsequently generate the corresponding code implementation of the new format. This code will be used by the system to render prompts according to the defined format. We again leverage a meta-prompt to instruct the LLM, this time to generate the executable code. As an illustrative example, here is a conceptual outline of the meta-prompt employed for generating the code representation of a new *Query Format*:

[META PROMPT HEADER]

We have some preset QUERY_FORMAT candidates, here are our whole search pool:
[ALL EXISTING QUERY FORMATS DESCRIPTION]

Here are two code implementations from our QUERY_FORMAT candidates as for your reference:

<Format name: Question-Answer>
<Renderer code>
[Question-Answer RENDERER CODE]
<Extractor code>
[Question-Answer EXTRACTOR CODE]

<Format name: Instruction-Response>
<Renderer code>
[Instruction-Response RENDERER CODE]
<Extractor code>
[Instruction-Response EXTRACTOR CODE]

Here is the example rendered by the new format:
[RENDERED RESULTS]

Please generate the code for this provided example based on the new QUERY_FORMAT. Ensure that both the renderer and extractor functions are included. The generated code should be plain Python code without any Markdown syntax or language identifiers such as ````python` or `python`. Please output the code directly without any additional formatting. If you need to use any additional and specific packages, please import them in the code. Note that the generated functions should include properly indented blocks, so they can execute without errors. Note that the renderer function name should be `query_renderer_{format_name}` and the extractor function name should be `query_extractor_{format_name}`.

Please encapsulate the code using the following format:

<START>
<Format name: {format_name}>
<Description: {format_description}>
<Renderer code>
[Renderer code]
<Extractor code>
[Extractor code]
<END>

B Appendix: Experimental Setup

B.1 Hyperparameter in Optimization Procedure

During content optimization, case-diagnosis and Monte Carlo sampling each generate 4 prompts per round. A set of 40 test cases is used, with 5 correct and incorrect cases leveraged for case-diagnosis. During optimization, the number of prompt components (e.g., Task Instruction, Task Detail, Output Format, Few-shot Examples) the LLM could modify simultaneously was dynamically adjusted across all the iterations. The number of components that could be modified decreases linearly from 4 to 1 over the iterations, promoting broad exploration of the prompt space initially and fine-grained refinement in later stages.

For format optimization, 4 UCT-selected formats and 4 newly generated formats are used to generate new prompts. The coefficient in the UCT selection process α is set to $1e - 3$.

Each experiment was capped at a maximum of 20 optimization iterations. An early stopping criterion was implemented, halting the process if the performance did not improve for a specified number of consecutive iterations.

B.2 Model Generation Parameters

Key generation parameters for both the LLM optimizer (GPT-4) and the target evaluation models are detailed below.

- **LLM Optimizer (GPT-4):** We utilized the following settings to generate and refine prompts: `top_p=1.0`, `max_tokens=4096`, `seed=42`, and `temperature=1.0`.
- **Evaluation Models:** The target evaluation model, evaluated on the generated prompts, was configured with: `top_p=0.1`, `max_tokens=256`, `seed=42`, `temperature=0.0`, `repetition_penalty=1.0`, and `stop='\\n'`. The stop parameter ensured the model ceased generation upon encountering a newline character. All generation processes were implemented using the vLLM library (Kwon et al., 2023).

B.3 Evaluation Metric

Performance across all tasks was evaluated using the *exact match* metric. For direct answer tasks (BigBench-Classification and ARC-Challenge), the model’s direct prediction was assessed for an exact

string match with the ground truth. For Chain-of-Thought reasoning tasks (GSM8K and MATH500), the final numerical answer was extracted from the generated explanation and compared to the correct solution for an exact match. Finally, the best-performing prompt identified on the evaluation set for each optimization method is reported on the corresponding test set.

B.4 Initial Prompt

B.4.1 Big-Bench Classification

Prompt Renderer: Directly Joint

Query Format: Input-Output

Examples:

Input: Speaker 1: 'You do this often?' Speaker 2: 'It's my first time.'

Output: no

`{{Query placeholder}}`

B.4.2 ARC-Challenge

Prompt Renderer: Directly Joint

Query Format: MultiChoice_QA

You are a commonsense helper. I will provide several examples and a presented question. Your goal is to pick the most reasonable answer among the given options for the current question. Please respond with the corresponding label (A/B/C/D) for the correct answer.

Here are some examples:

Question: Forests have been cut and burned so that the land can be used to raise crops. Which consequence does this activity have on the atmosphere of Earth?

Choices:

A: It reduces the amount of carbon dioxide production

B: It reduces the production of oxygen

C: It decreases the greenhouse effect

D: It decreases pollutants in the air

Answer: B

`{{Query placeholder}}`

B.4.3 GSM8K

Prompt Renderer: Directly Joint

Query Format: QA

Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

A: There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been $21 - 15 = 6$. The answer is 6.

`{{Query placeholder}}`

Task	Method	Llama3.1	Llama3-Ins
MMLU	Baseline	78.84	82.03
	CFPO	81.74	83.77

Table 5: Ablation of format generation and comparison of format selection strategies.

B.4.4 MATH500

Prompt Renderer: Directly Joint Query Format: Question-Answer

A chat between a curious user and an AI assistant. The assistant gives step-by-step solutions to the user's questions. In the end of assistant's response, a final answer is given in the format of "The answer is: <ANSWER>.".

Here are some examples:

```
Question: Let \[f(x) = \left\{ \begin{array}{ll} ax+3, & \text{if } x > 2, \\ x-5 & \text{if } -2 \leq x \leq 2, \\ 2x-b & \text{if } x < -2. \end{array} \right. \] Find $a+b$ if the piecewise function is continuous (which means that its graph can be drawn without lifting your pencil from the paper ).
```

Answer: Let's think step by step. For the piecewise function to be continuous, the cases must "meet" at \$x=2\$ and \$x=-2\$. For example, \$ax+3\$ and \$x-5\$ must be equal when \$x=2\$. This implies \$a(2)+3=2-5\$, which we solve to get \$a=-6\$. Similarly, \$x-5\$ and \$2x-b\$ must be equal when \$x=-2\$. Substituting, we get \$-2-5=2(-2)-b\$, which implies \$b=3\$. The answer is: \$a+b=-3+3=\boxed{0}\$.

```
 {{Query placeholder}}
```

C Appendix: CFPO Performance, Stability, and Cost Analysis

C.1 CFPO Performance on MMLU

To further evaluate the Content-Format Integrated Prompt Optimization (CFPO) approach, we assessed its performance on the MMLU benchmark. Specifically, we focused on high school history categories (including *high_school_european_history*, *high_school_us_history* and *high_school_world_history*) to evaluate CFPO's effectiveness on knowledge-based tasks.

Table 5 summarizes the MMLU performance, showcasing improvements achieved with CFPO, which reinforce the generalizability of CFPO, extending its demonstrated benefits from reasoning tasks (as presented in the main paper) to knowledge-based tasks. This expanded validation will be integrated into the revised manuscript.

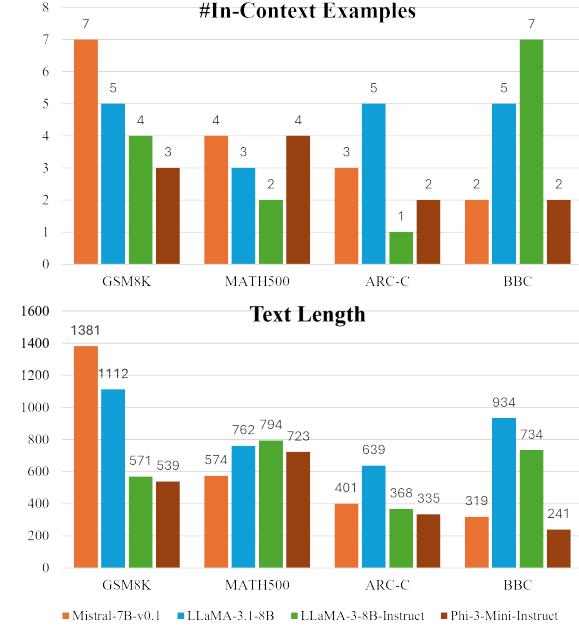


Figure 5: Overview of in-context examples and text lengths for optimized prompts on various tasks and models.

Model	CFPO	Reproduced CFPO
Mistral-7B-v0.1	53.22	51.87 ± 0.22
LLama-3.1-8B	63.38	62.70 ± 0.59
LLama-3.8B-Instruct	80.74	80.67 ± 0.17
Phi-3-Mini-Instruct	89.16	88.89 ± 0.31

Table 6: Stability Analysis: Comparison of reported CFPO scores with reproduced results (4 runs each).

C.2 Impact of In-context Examples and Prompt Length

Figure 5 presents an overview of the number of in-context examples and the text length of optimized prompts across various tasks and models. An interesting pattern emerges: pre-trained models consistently prefer prompts with longer text and more in-context examples compared to instruction-tuned models. This observation suggests that pre-trained models benefit more from explicit context and detailed reasoning steps, which align with their less task-specialized nature. In contrast, the relative insensitivity of instruction-tuned models to prompt length and in-context examples supports the notion that these models have already trained with task-specific knowledge during fine-tuning, reducing their dependence on highly detailed prompts.

C.3 Stability and Convergence Analysis

A crucial aspect of our framework is its stability. To quantify this, we performed multiple independent runs of CFPO and analyzed the variance in

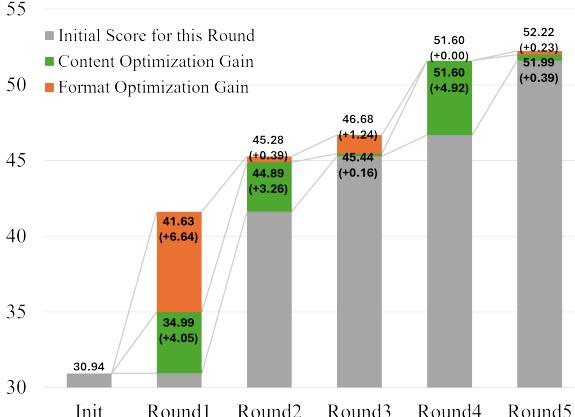


Figure 6: Convergence behavior of format and content optimization (Mistral-7B-v0.1 on GSM8K). Values indicate performance scores, with improvements from the previous sub-round in parentheses.

performance. Table 6 presents a comparison of the originally reported CFPO scores with the mean and standard deviation of scores obtained from four independent runs. The consistently low standard deviations across all models demonstrate the robustness and stability of our CFPO framework.

Furthermore, we examined the convergence behavior of content and format optimization. Preliminary analysis using Mistral-7B-v0.1 on GSM8K (Figure 6) indicates that format optimization may converge faster than content optimization. The performance gains in each sub-round (content or format optimization) are shown in parentheses. This suggests that the optimization schedule could be refined.

This difference in convergence speed could stem from the distinct optimization strategies: during content optimization, a diverse set of correct and incorrect examples is resampled from the training set in each round. In contrast, format optimization relies on generating variations of existing formats, leading to reduced diversity as the UCT score converges. We hypothesize that allocating more resources to format optimization in early rounds, and subsequently shifting focus to content optimization, could improve overall efficiency. This is an area for further investigation in future work.

C.4 Cost Analysis

An important consideration is the computational cost associated with CFPO. Table 7 breaks down the average token usage, API calls, and estimated cost per round of optimization for each operation within CFPO. These costs are calculated using current API pricing for the utilized LLMs.

Operation	Tokens (I/O)	API Calls	Est. Cost
Case-diagnose	2k/0.2k	32	\$ 0.83
Apply Feedback	0.5k/0.1k	110	\$ 0.93
Gen Variation	0.5k/0.2k	81	\$ 0.89
Gen Format	0.5k/0.1k	2	\$ 0.016
Gen Format Code	0.6k/0.2k	2	\$ 0.024
Total (per Round)	159.6k/36.6k	237	\$ 2.69

Table 7: Average token usage, API calls, and estimated cost per round of CFPO optimization. **Tokens (I/O)**: typical input/output tokens per round, **API Calls**: average API calls per round, **Est. Cost**: estimated cost in \$.

Task	Avg. Cost to Tar.	Avg. Time to Tar.
BBC	\$14.10	5.24 h
ARC-C	\$16.81	6.25 h
GSM8K	\$28.25	10.50 h
MATH500	\$19.85	7.38 h

Table 8: Average cost and time required for CFPO to reach the reported target state-of-the-art performance on each benchmark.

Furthermore, CFPO incorporates an early stopping mechanism that terminates the optimization process when performance improvement plateaus. This prevents unnecessary iterations and reduces overall cost. Table 8 details the average cost and time required to reach the reported performance on each benchmark task. As demonstrated in Table 8, the average computational cost per task remains manageable.

D Appendix: Format Generation Examples and Ablation Study

D.1 Examples of Generated Format

Here we select several format generated by GPT-4 in CFPO process.

D.1.1 Query Format

Highlight_Separator_Case

```
QUESTION > Statement 1 | Every element of a group generates a cyclic subgroup of the group.
Statement 2 | The symmetric group S_10 has 10 elements.
OPTIONS > (A) True, True (B) False, False (C) True, False (D) False, True
ANSWER > C
```

Cascading_Statements

```
Question: Statement 1 | Every element of a group generates a cyclic subgroup of the group.
Statement 2 | The symmetric group S_10 has 10 elements.
Options:
-A True, True
-B False, False
-C True, False
```

-D False, True
Answer: C

QA_Titlecase_Separator

Question || In 3 years, Jayden will be half of Ernesto's age. If Ernesto is 11 years old, how many years old is Jayden now?

Answer || Let's think step by step. Ernesto = 11 + 3 = <<11+3=14>>14 Jayden = 14/2 = <<14/2=7>>7 in 3 years Now = 7 - 3 = <<7-3=4>>4 Jayden is 4 years old.

QA_Brackets_Colon_Newline

[Question]:
In 3 years, Jayden will be half of Ernesto's age.
If Ernesto is 11 years old, how many years old is Jayden now?

[Answer]:
Let's think step by step.
Ernesto = 11 + 3 = <<11+3=14>>14 Jayden = 14/2 = <<14/2=7>>7 in 3 years Now = 7 - 3 = <<7-3=4>>4 Jayden is 4 years old.

QA_CapsBold_ColonNewline

QUESTION:
In 3 years, Jayden will be half of Ernesto's age.
If Ernesto is 11 years old, how many years old is Jayden now?

ANSWER:
Let's think step by step.
Ernesto = 11 + 3 = <<11+3=14>>14 Jayden = 14/2 = <<14/2=7>>7 in 3 years Now = 7 - 3 = <<7-3=4>>4 Jayden is 4 years old.

D.1.2 Prompt Renderer

Concise_Bullet_Points_Renderer

- Task Instruction: Write a function that returns the sum of two numbers.
- Task Detail: The function should take two numbers as input and return their sum.
- Examples: Input: 1, 2
Output: 3
- Query: Input: 1, 2
Output:

Tabular_Sections_Renderer

Task Instruction	Write a function that returns the sum of two numbers.
Task Detail	The function should take two numbers as input and return their sum.
Examples	Input: 1, 2 Output: 3
Query	Input: 1, 2 Output:

Checklist_Format_Renderer

- [] **Task Instruction**
Write a function that returns the sum of two numbers.
- [] **Task Detail**

Format Generation Model	Accuracy
Claude-3.5-haiku	58.91
Llama-3.3-70B-Instruct	61.71
Gemini-2.0	62.09
DeepSeek-R1	62.24
GPT-4 (CFPO)	63.38

Table 9: GSM8K Accuracy with Different LLMs for Format Generation (Using GPT-4 as content optimizer and LLaMA-3.1-8B as the target model).

The function should take two numbers as input and return their sum.

- [] **Examples**

Input: 1, 2
Output: 3

- [] **Query**

Input: 1, 2
Output:

D.2 Ablation Study of Format Generation Models

To investigate the potential reliance on a specific LLM for format generation within our framework, we conducted an ablation study. This study evaluated the performance of several alternative LLMs in generating the formats used by CFPO, as shown in Table 9. We systematically replaced the original format generation LLM (GPT-4) with alternative models: Gemini-2.0, DeepSeekR1, Claude-3.5-Haiku, and Llama-3.3-70B-Instruct. In these experiments, GPT-4 was consistently used for content optimization, and LLaMA-3.1-8B remained the target model for evaluating performance on the GSM8K benchmark.

The results demonstrate that the choice of LLM for format generation has a relatively minor impact on overall accuracy. While variations in the specific generated formats are observed across different LLMs, the consistent performance suggests that CFPO's effectiveness is not critically dependent on a single LLM. Specifically, as long as the LLM demonstrates the capability to generate variations and code based on the provided schema, the downstream performance on GSM8K remains stable. This robustness suggests that the core principles of CFPO are transferable and not tied to the idiosyncrasies of a particular LLM. This suggests CFPO is not overly sensitive to the choice of format generation model.

E Appendix: Examples of CFPO Optimal Prompt

Here we selected several optimal prompts searched by CFPO.

LLaMA-3.1-8B on ARC-C

```
<div class='TaskInstruction'>
  <h2>TaskInstruction</h2>
  <p>Your mission is to meticulously assess each situation presented alongside a specific question, employing your critical thinking and analytical skills. Your task comprises not only identifying the most logical and coherent choice (A/B/C/D) but also thoroughly evaluating how each option connects or diverges from the question's essence. This requires a deep engagement with both the query and the choices, ensuring your reasoning is firmly anchored in the specifics of the options provided. It is essential to weave direct elements from the choices into your analysis, demonstrating a detailed understanding of how each option relates to the core question, and articulating why alternatives may be less fitting given the scenario. This approach ensures a nuanced and well-justified selection process, grounded in the interplay between the question context and the specific details of the available choices .</p>
</div>
<div class='TaskDetail'>
  <h2>TaskDetail</h2>
  <p>In addressing the questions set before you, it is imperative to delve deeper than mere superficial observations or initial judgments. Each scenario or question must be examined not just in its immediate context but within a broader spectrum, looking into the underpinning mechanisms or far-reaching effects of each option presented. This necessitates a thorough exploration of the larger implications and the scientific or logical foundations that dictate the outcomes. For instance, in environmental matters, it is vital to assess not just the immediate effects but the sustained impact on the ecosystem. In the realm of science, such as when discerning chemical processes, it is crucial to understand the molecular or atomic level changes that classify a reaction as a chemical change. This enhanced level of scrutiny and deeper analysis will lead to more accurate and well-founded choices, ensuring your responses are not just correct, but are also backed by a solid understanding of the underlying principles or long-term consequences.</p>
</div>
<div class='OutputFormat'>
  <h2>OutputFormat</h2>
  <p>For every query presented, your task is to identify the right choice from the options (A/B/C/D) accompanied by a concise rationale for your selection. This format is vital as it showcases the thought process leading to your decision, facilitating a comprehensive grasp and interaction with the task.</p>
```

```
</div>
<div class='Examples'>
  <h2>Examples</h2>
  <p>Here are some examples:
```

Question: Forests have been cut and burned so that the land can be used to raise crops. Which consequence does this activity have on the atmosphere of Earth?

- A: It reduces the amount of carbon dioxide in the atmosphere
 - B: It reduces the availability of oxygen
 - C: It lessens the greenhouse effect
 - D: It lowers the levels of pollutants in the air
- Answer: B

Question: What is the most critical practice to ensure electrical safety while operating devices ?

- A: Ensure the device does not come into contact with water.
 - B: Use the device with hands covered in oil.
 - C: Operate the device with wet hands.
 - D: Leave the device plugged in when not in use.
- Answer: A

Question: Placing a plant cell in a hypertonic solution typically results in which of the following?

- A: The cell expanding as it absorbs water.
 - B: No significant change due to the rigid cell wall.
 - C: The cell shrinking as water exits the cell.
 - D: Rapid division of the cell.
- Answer: C

Question: What is the primary effect of using fossil fuels on global climate change?

- A: It leads to a significant reduction in greenhouse gases.
 - B: It decreases the Earth's surface temperature.
 - C: It increases the amount of greenhouse gases in the atmosphere.
 - D: It contributes to a decrease in carbon dioxide levels.
- Answer: C

Question: The process of photosynthesis in plants primarily involves which of the following transformations?

- A: Converting oxygen and glucose into carbon dioxide and water
 - B: Transforming water and carbon dioxide into oxygen and glucose
 - C: Changing sunlight into chemical energy without producing oxygen
 - D: Producing carbon dioxide and glucose from oxygen and water
- Answer: B

`{} query {}`

LLaMA-3.1-8B on GSM8K

Understanding the Task: A Foundation for Mathematical Problem-Solving
Your task is to methodically analyze the information provided and logically deduce the correct answer to the mathematical problem. Delve into each relevant detail, ensuring no critical step or aspect is overlooked. Approach

the solution with a detailed-oriented mindset, ensuring every part of the process is considered to arrive at an accurate conclusion. Reflect on all the elements that might influence your reasoning or calculation, striving for thoroughness in your analysis.

Decoding Mathematical Language in Real-World Scenarios

For the most effective problem-solving in mathematics, particularly when faced with intricate calculations over periods or under specific scenarios affecting results, an attentive and systematic method is key. Start by accurately determining the base numerical value. Then proceed by methodically listing every significant change whether it be increases, decreases, or modifications that impacts this base figure as the scenario unfolds, making sure to include each change in your overall computations. It's essential to focus on the concept of compounded operations, whether they're applied annually, monthly, or daily, and to thoughtfully evaluate the consequences of extraordinary events or circumstances (like an unexpected inheritance, a yearly loss, or a singular occurrence with a major impact) that might significantly shift the end calculations. Sharpen your attention on the dynamics of numerical relationships, particularly in cases involving ratios, proportions, and the impact of percentage changes over durations, to avoid common mistakes. Misunderstandings or misapplications of these numerical relationships can frequently cause inaccuracies. Thus, it is critical to scrutinize these mathematical relationships, whether they are of direct or inverse proportions, as well as the aggregate effects of consecutive percentage changes, as outlined in the problem description. This intensified attention is pivotal for an accurate and detailed resolution of complex issues, marked by multiplicative elements and interconnected circumstances. Reflect deeply on the significance of every step in the calculation process, absorbing the nuances of these changes, to systematically arrive at the most precise solution.

Ensuring Your Solution Fits the Scenario Perfectly

In presenting your solution, ensure it comprises both a numerical answer and a meticulously detailed explanation of the process leading to it. Begin with outlining the initial conditions and sequentially narrate the calculations you make at each step, highlighting any compounded operations or adjustments made to account for unique scenarios or conditions. This progression should clearly show how each step contributes to arriving at the final answer. For instance, if the task involves calculating the total costs saved over time with additional periodic benefits, your response should methodically explain: "Starting with an initial savings of X, plus Y every Z period, and considering an additional benefit of A every B period, leads to a total of...". This comprehensive breakdown not only bolsters the understanding of the mathematical principles applied but also

provides a robust framework for identifying and rectifying any potential inaccuracies throughout the problem-solving process.

Examples to Illuminate the Path

To better grasp the concepts, consider the following illustrative examples:

Question: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today? / ANSWER: Think through the problem step by step, diving into each segment for a thorough exploration to piece together the final answer. There are 15 trees originally. Then there were 21 trees after some more were planted. So there must have been $21 - 15 = 6$. The answer is 6.

Question: A book club starts with a membership of 120. If the club increases its membership by 10% in the first year and then loses 5% of its members in the second year, what is the total membership at the end of the second year? / ANSWER: Think through the problem step by step, diving into each segment for a thorough exploration to piece together the final answer. The club starts with 120 members. In the first year, it increases by 10%, which is $0.10 * 120 = 12$, so there are $120 + 12 = 132$ members after the first year. In the second year, the club loses 5% of its members, which is $0.05 * 132 = 6.6$, but since the number of members must be an integer, we consider a loss of 7 members (assuming the figure is rounded up for practical reasons). Therefore, there are $132 - 7 = 125$ members at the end of the second year.

Question: Martin saves \$10 every week. In addition, every third week, he earns an extra \$15 from helping his neighbor. How much has Martin saved after 9 weeks? / ANSWER: Think through the problem step by step, diving into each segment for a thorough exploration to piece together the final answer. Martin saves \$10 each week, so over 9 weeks, he saves $9 * \$10 = \90 . Additionally, every third week, he earns an extra \$15, which occurs three times within 9 weeks (in the 3rd, 6th, and 9th weeks). So, he earns an extra $3 * \$15 = \45 from helping his neighbor. Therefore, the total amount Martin has saved after 9 weeks is $\$90 + \$45 = \$135$.

Question: A teacher divides a class into groups for a project. If the ratio of boys to girls in the class is 3 to 2, and there are 30 students in the class, how many boys are in the class? / ANSWER: Think through the problem step by step, diving into each segment for a thorough exploration to piece together the final answer. The total ratio units for boys to girls in the class is $3 + 2 = 5$. With 30 students in the class, each ratio unit represents $30 / 5 = 6$ students. Therefore, the number of boys, represented by 3 parts of the ratio, is $3 * 6 = 18$. The answer is 18.

Question: Grandma wants to order 5 personalized backpacks for each of her grandchildren's first days of school. The backpacks are 20% off of \$20.00, and having their names monogrammed on the

backpack will cost \$12.00 each. How much will the backpacks cost in total? / ANSWER: Think through the problem step by step, diving into each segment for a thorough exploration to piece together the final answer. The backpacks are 20% off of \$20.00, so the price after the discount is $\$20.00 - (\$20.00 * 20\%) = \$20.00 - \$4.00 = \$16.00$ each. The monogramming costs an additional \$12.00 per backpack. Therefore, the total cost for each backpack is $\$16.00 + \$12.00 = \$28.00$. For 5 backpacks, the total cost will be $5 * \$28.00 = \140.00 . The correct answer is \$140.00.

Query
{{query}}

LLaMA-3-8B-Instruct on MATH-500

- Task Instruction: A chat between a curious user and an AI assistant focused on solving mathematical and reasoning tasks. The assistant is expected to deliver step-by-step solutions to the user's questions, emphasizing mathematical accuracy and rigor throughout the process. It must ensure that each mathematical operation and logical deduction is carefully examined and validated to derive the correct solution. At the conclusion of the response, the final answer should be presented in the format of "The answer is: <ANSWER>.", thereby confirming the solution's validity and demonstrating a thorough understanding of the problem-solving approach.

- Task Detail: In addressing equation-based inquiries, precision in algebra, geometry, piecewise functions, complex numbers, and financial mathematics is paramount. This involves a detailed analysis of each equation, assessing every element and specific condition. For piecewise functions, it's critical to ensure continuity by solving for variables that maintain consistency across sections. In geometry, integrating measurements such as angles, lengths, and areas is fundamental. Algebraic queries require a consideration of all potential solutions and constraints, ensuring a comprehensive resolution. The addition of complex numbers into this mix necessitates a thorough understanding of their properties and operations to accurately determine both real and imaginary solutions. Similarly, tackling financial mathematics problems demands a deep comprehension of concepts such as compound interest, present value, and future value to make precise financial forecasts and comparisons. This holistic approach confirms that all aspects of the problem are considered and that the solution accounts for every requirement, assuring mathematical integrity in the resolution process.

- Output Format: 1. Solutions that involve fractions, square roots, or crucial mathematical figures (e.g., pi) must be simplified to their most fundamental form. This includes reducing fractions to their lowest terms and expressing square roots in their least complex radical form.

2. Avoid the use of decimals unless the question explicitly requires it or they are necessary

for conveying the most precise value possible.

3. Present solutions involving square roots in their reduced radical form, ensuring the simplification process enhances comprehension without diluting mathematical integrity.
4. In scenarios involving complex numbers, represent answers in their standard form ($a + bi$), ensuring both 'a' and 'b' are presented in their simplest, most refined state. This emphasizes the need for a clear, coherent representation of solutions encompassing complex numbers.

5. Conclude your explanation with the statement: "The answer is: <ANSWER>.", reinforcing consistency and clarity across various mathematical challenges. This concluding statement should encapsulate the solution in its simplest and most direct form, reflecting a thorough simplification and rationalization process.

Your explanation must delineate a detailed, step-by-step progression leading to the final solution. This approach is not merely about arriving at the correct answer but about illuminating the path taken to get there, ensuring a deep understanding and clear demonstration of the reasoning behind each step.

- Examples: Here are some examples:

Instruction:
A rectangle ABCD has sides AB = 8 units and BC = 6 units. A circle with a radius r units is inscribed within this rectangle. Calculate the radius r of the inscribed circle, ensuring the answer is in its simplest form.

Response:

We'll approach this problem by breaking it down into manageable steps. We start by understanding that the radius of the inscribed circle is equal to the distance from the center of the rectangle to any of its sides because the circle is perfectly inscribed. In a rectangle, this distance is half the length of the rectangle's shorter side. Therefore, the radius r of the inscribed circle is half the length of BC, which is $6 \div 2 = 3$ units. The answer is: $r=3$.

Instruction:

Given a triangle where two sides are represented by complex numbers $(3 + 4i)$ units and $(1 - 2i)$ units, and the angle between them is 90 degrees, calculate the length of the hypotenuse. Ensure your answer includes a comprehensive breakdown of complex number operations and geometric principles applied.

Response:

We'll approach this problem by breaking it down into manageable steps. We start by acknowledging that the length of a side represented by a complex number can be found using the modulus of that number. The modulus of the first side is $\sqrt{3^2 + 4^2} = 5$ units, and the modulus of the second side is $\sqrt{1^2 + (-2)^2} = \sqrt{5}$ units. Since these sides form a right triangle and we are given that the angle between them is 90 degrees, we can apply the Pythagorean theorem to find the length of the

hypotenuse. The hypotenuse's length squared will be the sum of the squares of the lengths of the other two sides, which is $5^2 + (\sqrt{5})^2 = 25 + 5 = 30$. Thus, the length of the hypotenuse is $\sqrt{30}$ units. The answer is: $\sqrt{30}$.

- Query:
{{query}}