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

**Anonymous ACL submission**

## 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 this link.

## 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).

However, manual design of prompts presents significant challenges, primarily due to the high sensitivity of LLMs to subtle variations in prompt characteristics, including both textual content and

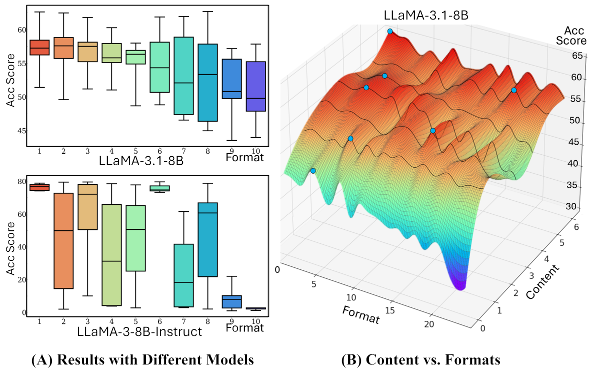


Figure 1: The crucial role of prompt formatting and its interaction with content. **(A)**: Model-specific format biases: Illustrates the performance sensitivity of two LLMs to different format styles on the GSM8K task, showing substantial variability in the effectiveness of 10 randomly selected formats. **(B)**: For seven different prompt contents evaluated across 24 distinct formats, performance variations show the complex, interdependent relationship between prompt content and structure, demonstrating that no single format universally maximizes effectiveness.

structural format (Jiang et al., 2022; Zamfirescu-Pereira et al., 2023; Salinas and Morstatter, 2024). These sensitivities are further complicated by variations across different models and tasks (Zhuo et al., 2024; Sclar et al., 2024). To alleviate these difficulties, automated prompt optimization techniques, often leveraging the power of LLMs themselves, have proven to be an effective approach 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 largely unexplored dimension: the **prompt formatting**.

Our preliminary investigations, as illustrated in Figure 1, provide valuable insights into the role of prompt format in prompt optimization. We have observed that different LLMs display distinct preferences, with some formats performing well on one model but failing on another. This suggests sophis-

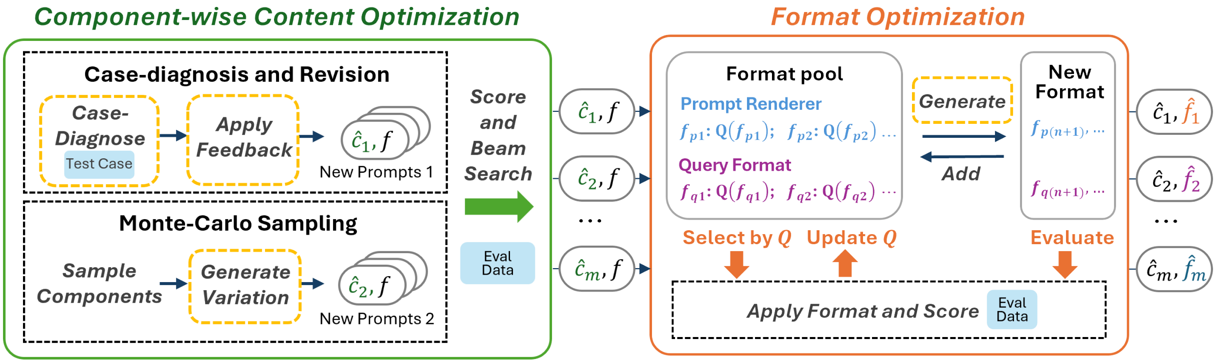


Figure 2: Illustration of the CFPO pipeline within a single iteration round. In the initial Component-wise Content Optimization stage, case-diagnosis and Monte-Carlo sampling are employed for content mutation. Subsequently, the Format Optimization stage identifies the most suitable format for each content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

ticated, model-specific format biases (Sclar et al., 2024). Furthermore, we have identified a complex interplay between prompt content and format, where no single format consistently outperforms others across all contents. This lack of a universally optimal format highlights the impracticality of predefining format, and underscores the need for a joint optimization approach that treats prompt 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 leverages the principles of structured thinking, operating along two key dimensions: the *Prompt Renderer*, which governs the organizational structure of all components within a prompt (He et al., 2024), and the *Query Format*, which dictates the presentation of in-context learning examples and queries (Voronov et al., 2024a; Salinas and Morstatter, 2024). By integrating these two dimensions, CFPO defines a structured template that effectively distinguishes between content and format types, enabling the efficient identification of high-performing prompts.

Our primary contributions are threefold: (1) We propose **CFPO**, an innovative approach to simultaneously optimizes prompt content and format using an iterative process. (2) We introduce an efficient strategy for dynamic format optimization that generates new formats in an iterative manner and evaluates formats instance through a scoring system to select the best option. (3) Through extensive evaluations across diverse tasks and multiple open-source LLMs, we demonstrate that CFPO consistently improves LLM performance in a measurable and effective manner.

## 2 Related Work

**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).

**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), feedback-based methods (Pryzant et al., 2023; Das et al., 2024), and agent-driven frameworks (Wang et al.,

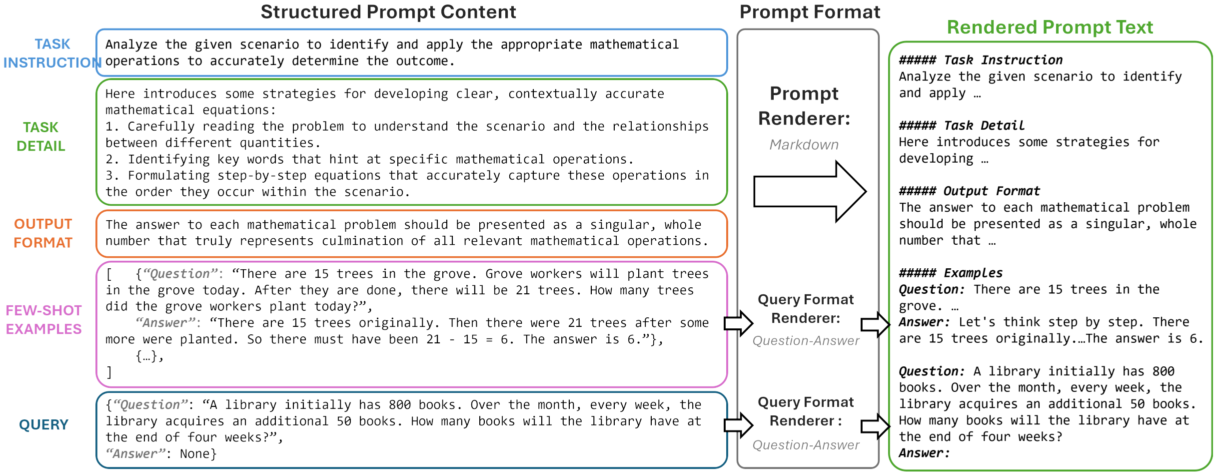


Figure 3: An illustrative example of our Structured Prompt Template. This template systematically organizes the prompt into distinct components, each serving a specific functional role. When formulating a prompt, the template first employs a Query format to present examples and queries, and then integrates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

2024a; Khattab et al., 2024; WHO, 2023). While these methods focus on optimizing the overall prompt, they often lack the capability for fine-grained modifications. (Khattab et al., 2024; Schnabel and Neville, 2024) introduce phrase-level mutations, but they fail to address format mutations or implement them in a systematic manner.

**Prompt structure and format** Structured prompting, which organizes prompts into distinct components such as instructions, examples, and queries, holds significant potential in prompt engineering (Fernando et al., 2023). Empirical rules for prompt design always lack integration with automatic optimization techniques, limiting their scalability and effectiveness (Nigh, 2023; Google, 2024). Frameworks like LangGPT (Wang et al., 2024a) have introduced structured prompting paradigms, emphasizing reusable designs inspired by programming principles. However, these efforts primarily focus on content-level refinements and fail to adequately address the critical role of prompt formatting. 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 impact 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. However, despite the recognition of formatting's importance, there remains a lack of comprehensive understanding regarding the optimization of prompt format in a systematic manner.

## 3 CFPO: Content-Format Integrated Prompt Optimization

As we have established, the effectiveness of LLMs is profoundly influenced by both the content and format of prompts. Existing automated prompt optimization methods have largely overlooked the format dimension, which exhibits a strong model bias. To address this critical limitation, we introduce Content-Format Integrated Prompt Optimization (CFPO) framework that jointly optimizes both prompt content and format. This contrasts with prior approaches focusing solely on content optimization. Our goal is to identify an optimal prompt $p^*$, comprising both content ($c^*$) and format ($f^*$), that maximizes performance on an evaluation dataset $\mathcal{D}$, given by:

$$p^* : (c^*, f^*) = \arg \max_{c \in \mathcal{L}, f \in \mathcal{F}} m(c, f|\mathcal{D}), \quad (1)$$

within the coherent natural language space $\mathcal{L}$ and the space of all possible formats $\mathcal{F}$, guided by a metric function $m(\cdot)$ that assesses the prompt's quality.

To effectively search this complex space, CFPO employs a two-pronged iterative approach, detailed in Figure 2, that consists of two concurrently-run optimizers: a *Component-wise Content Optimizer* and a *Format Optimizer*. The content optimizer refines the textual content of a prompt, while the format optimizer explores the structural arrangement of its elements. Importantly, our framework

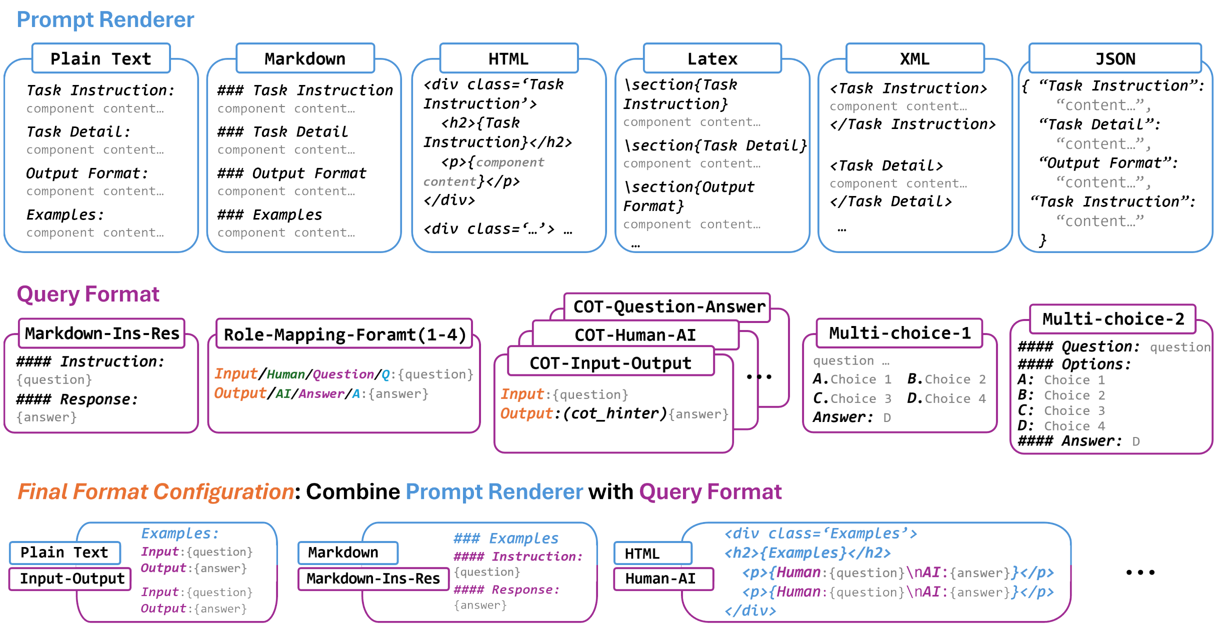


Figure 4: Built-in formats and rendering effects in our initial format pool. The final format configuration is achieved by selecting and combining elements from both the *Prompt Renderer* and the *Query Format* categories.

acknowledges the inherent interdependence of content and format and thus iterates between optimizing them, to find their optimal combination. The following sections detail our structured prompt template (Section 3.1), our innovative format optimization approach (Section 3.2), and finally our integrated optimization process (Section 3.3).

### 3.1 Structured Prompt Template

To enable fine-grained and targeted optimization, our framework adopts a structured prompt template inspired by guidelines from OpenAI (2024b) and Google (2024). This template decomposes prompts into distinct functional components, facilitating both analysis and selective mutations. Specifically, our template divides a prompt into content-based components and format-based components, as illustrated in Figure 3.

The **Content-based Components** are:

**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 contextual learning patterns, consisting of:

- ***Examples***: specific instances pertinent to the task, including inputs and expected outputs.
- ***Example Hinter*** (optional): a brief hint indicating that examples segment will follow, e.g., 'Here are some examples:'.
- ***CoT Hinter*** (optional): encourages a chain-of-thought reasoning process, e.g., 'Let's think step by step'.

**Query** shows the question or request to be answered by the LLM.

The **Format-based Components** are:

**Query Format**: defines how to structure the rendering of examples and queries.

**Prompt Renderer** defines how to aggregate all components into a structured prompt.

The formulation of the structured prompt template is fundamental to our optimization approach. This design yields two key advantages: first, it facilitates a structured component functionality where each part serves a specific purpose, promoting a more organized prompting framework; second, it enables fine-grained optimization by decoupling format from content, thus allowing targeted and precise modifications of individual components.

### 3.2 Format Optimizer Design

The key aspect of our work is the format optimization methodology. To efficiently explore the extensive range of prompt formats, the CFPO format optimizer adopts an approach that utilizes a format pool with a scoring system and an LLM-assisted format generation module. It strategically explores, evaluates, and refines formatting choices, all while learning from previous iterations.

#### 3.2.1 Format Pool with Scoring System

The format pool is designed to hold the format configurations we use to generate prompts. As shown in Figure 4, these configurations are separated into two dimensions: the *Prompt Renderer*, which dictates the overall structure of the prompt, and the *Query Format*, which governs the rendering of in-context examples and queries. This distinction allows us to explore both macro and micro-level formatting variations.

To dynamically evaluate the potential of each format, we developed a scoring system for assessing the performance of each format $f$, represented as $Q(f)$. This system updates the performance score of $f$ across various prompt contents using the formula $Q(f) \leftarrow Q(f) + \sum_c m(c, f)$, where $c$ represents each content instance in current round. Additionally, we maintain $N(f)$ to count the number of times a format has been visited, which facilitates score normalization.

To initialize the exploration, we constructed an initial format search space $\mathcal{F}$, comprising a set of predefined commonly used formats, as illustrated in Figure 4. We also incorporate diverse variations of these predefined formats into the initial search space, such as adjustments to spacing, punctuation, and the use of special symbols. This establishes a starting point for our optimization.

#### 3.2.2 LLM-assisted Format Generation

The variability of format space requires an automated process for effective expansion and exploration. To that end, we introduce an LLM-based format generator, $LLM_{f\_gen}$, which autonomously generates new formats based on information in the existing format pool.

This evolutionary approach integrates the format generation into each optimization round, allowing for the creation of new and potentially beneficial formats. To enhance the efficiency of this process, we guide the LLM towards more promising areas by informing it of the performance function, $\frac{Q(f)}{N(f)}$. 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. More detailed information of our format generation process is provided in the Appendix A.2.

#### 3.2.3 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:

$$UCT(f) = \frac{Q(f)}{N(f)} + \alpha \sqrt{\frac{\sum_f N(f)}{N(f)}} \tag{2}$$

where $\alpha$ 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 optimization round: $k$ promising 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 op-

---

**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.

Initialize: $Q(f) \leftarrow m(c_0, f)$, $N(f) \leftarrow 1$ for all $f \in \mathcal{F}$
**Format Selection**: $\mathcal{F}_{select} \leftarrow \{f \in \mathcal{F} : f$ is in the top $k$ w.r.t. $UCT(f)\}$
**Format Generation**:
**for** each $i = 0, 1, ..., k$ **do**
    Generate format: $f_{new} \leftarrow LLM_{f\_gen}(\mathcal{F})$
    Collect $f_{new}$ to $\mathcal{F}_{gen}$, and add $f_{new}$ to $\mathcal{F}$
**end for**
**Format Evaluation**:
**for** each $f \in \mathcal{F}_{select} \cup \mathcal{F}_{gen}$ **do**
    Evaluate $m(c, f)$ with dataset $\mathcal{D}$
    $Q(f) \leftarrow Q(f) + m(c, f)$
    $N(f) \leftarrow N(f) + 1$
    Update $UCT(f)$ by Eq. 2
**end for**
$\hat{f} \leftarrow \arg\max_{f \in \mathcal{F}_{select} \cup \mathcal{F}_{gen}} m(c, f)$

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

timizer ensures a balance between exploring new formats and refining current ones, converging to the best format configuration.

### 3.3 Integrated Optimizer Design

CFPO orchestrates the Component-wise Content Optimization and Format Optimization within an iterative framework to jointly optimize content and format. This iterative process (illustrated in Figure 2) is key to our methodology.

**Component-wise Content Optimization:** This stage employs two primary strategies for mutating the content of prompts. The first strategy is case-diagnosis and revision, leveraging test cases to assess the efficacy of the current prompt. The outcomes of these test cases, including both correct and incorrect samples, are analyzed by the LLM optimizer. This optimizer evaluates the performance and pinpoints specific components in need of optimization. Subsequently, targeted feedback is applied to these identified components for enhancement, resulting in improved prompts. For example, if the output is not in the specified format, the output format component will be altered. Additionally, a Mote-Carlo sampling strategy is employed to enhance the optimization robustness by generating synthetic content with same semantics for randomly selected components. After this step, we select top-performing content candidates based on an evaluation dataset for the next stage.

**Format Optimization:** As discussed in Section 3.2, this stage identifies the most effective format for each candidate prompt content from the previous content-optimization step. The format optimizer applies our dynamic format exploration and evaluation process, tracking performance, and updating the format pool's scoring system. The Format Optimizer meticulously tracks the performance of all evaluated formats, providing valuable insights to guide the selection of formats in subsequent iterations. Simultaneously, it retains only the most effective format for each prompt, ensuring the diversity of prompt content candidates during beam search.

In summary, the two optimizers work in tandem, leveraging the strengths of the LLM to facilitate swift adaptation and customization. Importantly, this iterative process allows for the optimization of format and content, thereby significantly enhancing the quality of the generated prompts.

## 4 Experiments

### 4.1 Experimental Setups

**Dataset and Models.** To rigorously evaluate CFPO, we selected a diverse set of tasks and models. Our benchmark tasks span various domains and complexities, including:

- **Reasoning**: GSM8K (Cobbe et al., 2021) and MATH500 (Hendrycks et al., 2021; Lightman et al., 2023) which require complex mathematical reasoning abilities.
- **Multiple-choice**: ARC-Challenge (Clark et al., 2018), demanding understanding and selection among alternatives.
- **Classification**: The *Implicatures* task from the Big-Bench benchmark (bench authors, 2023) to evaluate classification proficiency.

Our model selection includes a mix of foundational and instruction-tuned models to understand the generalizability of our approach:

- **Foundational Models**: Mistral-7B-v0.1 (Jiang et al., 2023) and LLaMA-3.1-8B (Meta, 2024b) represent pre-trained models.
- **Instruction-Tuned Models**: LLaMA-3-8B-Instruct (Meta, 2024a) and Phi-3-Mini-Instruct (Microsoft, 2024) represent models specifically fine-tuned for instruction following.

Furthermore, we use GPT-4 (2024-05-01-preview) as the LLM optimizer for content mutation and format generation (OpenAI, 2024a).

**Implementation Details.** The training process involved 20 iterative rounds, each consisting of content and format optimization. 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. The number of prompt-structured components decreases progressively from 4 to 1, narrowing the search space over time to enhance efficiency. 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 $\alpha$ is set to $1e-3$. Beam search, with a budget of 8, is employed during mutations to ensure effective exploration. Eval data sizes are configured as 50, 300, 500, and 500 for BigBench-Classification, MATH500, GSM8K, and ARC-Challenge, respectively. The best-performing prompt on the evaluation set for each method was selected and reported on the test set.

| Method | Mistral-7B-v0.1 | LLaMA-3.1-8B | LLaMA-3-8B-Instruct | Phi-3-Mini-Instruct |
|---|---|---|---|---|
| | | ***GSM8K*** | | |
| 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** |
| | | ***MATH-500*** | | |
| 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** |
| | | ***ARC-Challenge*** | | |
| 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** |
| | | ***Big-Bench Classification*** | | |
| 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** |

Table 1: Main results on math reasoning tasks and commonsense reasoning tasks.

**Baselines.** To evaluate the effectiveness of CFPO, we compared against 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. This choice of baselines enables a comprehensive assessment of CFPO's capabilities against various types of optimization approaches. All methods were evaluated using consistent experimental configurations to ensure a fair comparison.

**Initial Prompts.** To establish a reasonable starting point, we employed 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. Chain-of-Thought examples were employed for the reasoning tasks. We also report common baseline prompts, including 8-shot for GSM8K and 4-shot for MATH500. A comprehensive list of our initial prompts is in Appendix C.

### 4.2 Main Results

Table 1 summarizes the performance of CFPO in comparison with several state-of-the-art methods across four datasets. The results highlight the superior performance of CFPO, significantly outperforming the baseline prompt as well as competing methods. We observed that pre-trained models exhibit greater sensitivity to prompt formatting, leading to substantial improvements when optimized by CFPO. Notably, optimized prompts for pre-trained models tend to be longer and incorporate more in-context examples, suggesting that these characteristics better align with the optimization needs of pre-trained models (see Appendix D.1). In contrast, instruction-tuned models display relatively more robust results and smaller gains, likely due to their inherent adaptability and generalization.

For the reasoning tasks, GSM8K and MATH, prompt optimization is especially impactful due to the sensitivity of these tasks to prompt structure. CFPO, which integrates unified content and format optimization, delivers significant performance gains. Specifically, the improvement for GSM8K is more evident compared to the more challenging MATH task, where the inherent complexity limits the magnitude of improvement. Moreover, feedback-based methods like ProTeGi, SAMMO, and CFPO, consistently outperform the other baselines because they leverage iterative feedback for prompt refinement. In contrast, GRIPS, which is limited to phrase-level mutations, exhibits marginal improvements. These results underline the effectiveness of the integrated optimization strategy adopted by CFPO. The selected optimal prompts discovered by our approach can be found in Appendix D.

### 4.3 Ablation Study

**Impact of the Format Optimizer.** CFPO incorporates a unique format optimization process, leveraging LLM for format generation and a UCT-based strategy for format selection. To evaluate its effectiveness, we evaluated two variations of our method: (1) CFPO$_c$, which optimizes content while keeping format fixed, and (2) CFPO$_c$+Format, which first optimizes content, then performs a separate format optimization step. Table 2 shows that both CFPO$_c$ and CFPO$_c$+Format underperform compared to the full CFPO approach, highlighting the importance of the integrated content and format optimization approach. The need for a joint optimization process which addresses the interdependence of content and format is essential for prompt optimization.

| Task | Method | LLaMA-3.1-8B | LLaMA-3-8B-Instruct |
|---|---|---|---|
| GSM8K | ProTeGi | 54.74 | 75.36 |
| | CFPO$_c$ | 58.07 | 77.71 |
| | CFPO$_c$+Format | 61.94 | 79.30 |
| | **CFPO** | **63.38** | **80.74** |
| BBC | ProTeGi | 81.00 | 82.00 |
| | CFPO$_c$ | 85.00 | 85.00 |
| | CFPO$_c$+Format | 88.00 | 89.00 |
| | **CFPO** | **90.00** | **91.00** |

Table 2: Ablation study of the format optimizer and content optimizer. CFPO$c$ performs content optimization with a fixed format. CFPO$c$+Format performs format optimization after content optimization.

**Effectiveness of Format Generation.** We compared the full CFPO approach against a variant that uses format from initial format pool without using LLM for generation. As presented in Table 4, CFPO with format generation consistently outperforms the baseline relying solely on the initial pool. These results demonstrate the effectiveness of the proposed format exploration mechanism in enhancing both the quality and diversity of prompts.

| Task | Method | LLaMA-3.1-8B | LLaMA-3-8B-Instruct |
|---|---|---|---|
| GSM8K | w/o Format Gen | 62.70 | 78.85 |
| | with Format Gen | **63.38** | **80.74** |
| BBC | w/o Format Gen | 88.00 | 87.00 |
| | with Format Gen | **90.00** | **91.00** |

Table 3: Impact of format generation during prompt optimization.

**Effectiveness of Format Selection.** We further evaluated our UCT-based format selection process, compared it to a random selection from the format pool and a greedy selection without exploration (using $\alpha = 0$ in Eq. (2)). As presented in Table3, CFPO consistently achieves the best performance across all experimental settings, demonstrating the efficacy of the UCT-based selection strategy.

| Task | Method | LLaMA-3.1-8B | LLaMA-3-8B-Instruct |
|---|---|---|---|
| GSM8K | Random | 62.40 | 78.82 |
| | UCT($\alpha = 0$) | 63.23 | 79.08 |
| | UCT(ours) | **63.38** | **80.74** |
| BBH | Random | 85.00 | 87.00 |
| | UCT($\alpha = 0$) | 86.00 | 88.00 |
| | UCT(ours) | **90.00** | **91.00** |

Table 4: Impact of different format selection strategies during optimization.

**Effectiveness of the Content Optimizer.** As presented in Table 2, we include ProTeGi (Pryzant et al., 2023), a baseline that optimizes only the content. In contrast, our CFPO$_c$, which incorporates structured prompting and integrates correct cases for diagnosis, achieves significant performance improvements, which highlights the effectiveness of our content optimization strategy.

## 5 Conclusion

This paper introduces Content-Format Integrated Prompt Optimization (CFPO), an innovative methodology that concurrently optimizes both prompt content and format. CFPO incorporates the *Prompt Renderer* and the *Query Format* within a structured prompt template. By leveraging distinct optimization strategies, CFPO discovers high-performing prompts that outperform content-only methods, addressing a critical gap in existing research. 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 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, while the format generation mechanism shows strong potential, its stability can be an issue, as the optimization process may not always yield consistent or optimal results across different datasets or configurations.

## 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.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers.

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, Reiichiro 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/.

Patrick Haluptzok, 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.

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In European conference on machine learning, pages 282–293. Springer.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. Transactions on Machine Learning Research.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri 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.

Matt Nigh. 2023. Chatgpt3 free prompt list. https://github.com/mattnigh/ChatGPT3-Free-Prompt-List.

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. 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. pages 670–686.

Sander Schulhoff, Michael Ilie, Nishant Balepur, Konstantine Kahadze, Amanda Liu, Chenglei Si, Yinheng 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 Goncearenco, 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.

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.

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.

J. D. Zamfirescu-Pereira, Richmond Y. Wong, Bjoern Hartmann, and Qiang Yang. 2023. Why johnny can't prompt: How non-ai experts try (and fail) to design llm prompts.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. 2024. Self-Taught Optimizer (STOP): Recursively Self-Improving Code Generation.

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.

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. 2024. 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 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>
```

## A.3 Content Optimization

### A.3.1 Case-diagnosis and Revision

As described in Section 3.3, 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.3.2 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:
```

## B Appendix: Examples of Generated Format

Here we select several format generated by GPT4 in CFPO process.

### B.1 Query Format

#### 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.
```

#### 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
```

#### 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
```

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

- [ ] **Examples**
Input: 1, 2
Output: 3

- [ ] **Query**
Input: 1, 2
Output:
```

# C Appendix: Initial Prompt

## C.1 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}}
```

## C.2 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 \[f(x) = \left\{
\begin{array}{cl} ax+3, &\text{ if }x>2, \\
x-5 &\text{ if } -2 \le x \le 2, \\
2x-b &\text{ if } x <-2.
\end{array}
\right.\]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 $2$ and $-2$. 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 $2a
=-6 \Rightarrow a=-3$. 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.3 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}}
```

## C.4 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}}
```

# D Appendix: CFPO Results Analysis

## D.1 In-context Examples and Text Length

Figure 5 presents an overview of the number of in-context examples and the text length of optimized

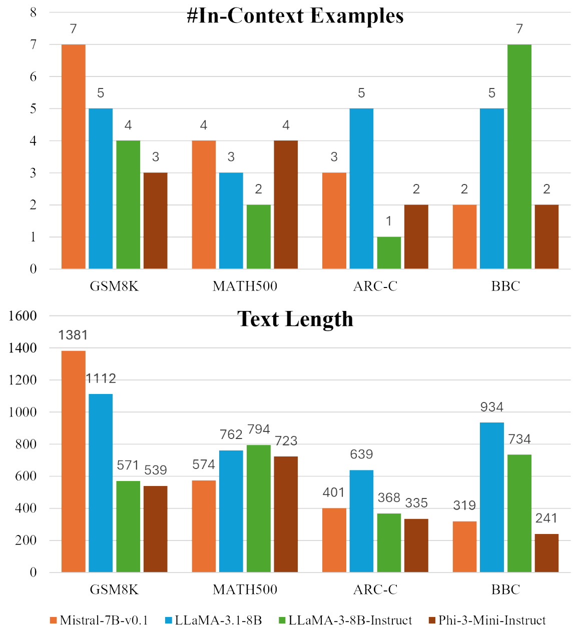


Figure 5: Overview of in-context examples and text lengths for various tasks and models.

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.

## D.2 Examples of Optimal Prompt

Here we selected several optimal prompts searched by CFPO.

### 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}}
```

## 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 }}