HIPO: A HIerarchical Prompt Optimization Framework with Task Awareness and Fine-Grained Debugging

Lu Qi[1,2], Lei Chai[1,2], Hongrui Yu[1,2], Binhang Qi[1,2,3], Hailong Sun[1,2,*]
[1]State Key Laboratory of Complex & Critical Software Environment (CCSE), Beihang University, China
[2]Zhejiang Key Laboratory of Industrial Big Data and Robot Intelligent Systems, Hangzhou Innovation Institute of Beihang University, China
[3]National University of Singapore, Singapore
{qiluxt, chailei, yuhongrui, sunhl}@buaa.edu.cn, qibh@nus.edu.sg

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities across diverse natural language processing tasks. However, their performance often hinges on carefully designed prompts, whose creation requires substantial human effort. While numerous automatic prompt optimization techniques have been proposed, existing methods typically apply the same prompt across all samples within a dataset, ignoring variation in sample difficulty. To address these limitations, we propose HIPO, a *HIerarchical Prompt Optimization* framework that shifts the paradigm from dataset-level to sample-level optimization. Our framework first employs a lightweight *router model*, trained offline, to predict the difficulty of each sample at test time. Based on this prediction, HIPO dynamically selects a prompt from a five-tiered hierarchy, tailoring complexity to sample difficulty. Furthermore, two refinement stages—*Task Description Prompt Refine* and *Attribution-Based Prompt Refine*—enhance generalizability and fine-grained optimization. Extensive experiments on 27 tasks demonstrate that HIPO outperforms all baselines, achieving state-of-the-art performance on **25%** more tasks than the strongest baseline. Cost analysis further demonstrates substantial efficiency gains, reducing API calls, token consumption, and overall cost by **1.2× to 80×**. Our implementation is publicly available at `https://github.com/LuQiCode/HIPO`.

1 Introduction

Large language models (LLMs) such as GPT-4 (OpenAI et al., 2024) and DeepSeek (DeepSeek-AI, 2025, 2024) have demonstrated impressive performance across a wide range of tasks (Colombo et al., 2024; Zhang et al., 2024; Touvron et al., 2023). Prompting has become the primary mode of interaction between users and these models (Liu et al., 2023). Carefully designed prompts allow users to steer model outputs more effectively, making prompt engineering a crucial factor in eliciting desirable results from LLMs (Wang et al., 2023a,b). However, crafting high-quality prompts often demands significant human expertise, labor, and subjective intuition (Reynolds and McDonell, 2021; Mishra et al., 2021). As LLMs continue to evolve rapidly and user tasks become increasingly diverse, automating prompt engineering has attracted growing attention as a way to reduce human effort and improve adaptability (Pryzant et al., 2023; Zhou et al., 2023).

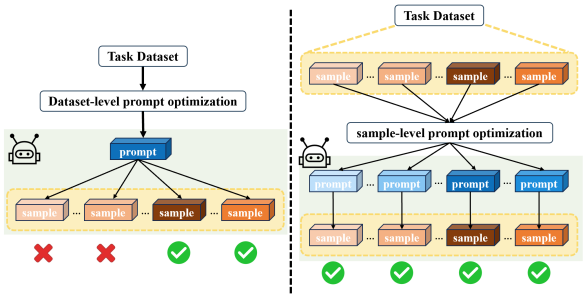

Figure 1: Comparison between dataset-level and sample-level prompt optimization. More details in Sec. 2.

Automated prompt generation refers to methods that automatically generate prompts tailored for a given task, either using LLMs themselves or other optimization strategies (Lester et al., 2021; Wang et al., 2022; Shin et al., 2020; Deng et al., 2022a; Zhang et al., 2022). It has emerged as a key enabler for applying LLMs to various downstream applications and also represents a central challenge in the LLM era (Pryzant et al., 2023; Zhou et al., 2023). Recent methods have focused on enumerating diverse prompts or refining existing ones to optimize instructions for black-box LLMs (Zhou et al., 2023; Lin et al., 2024; Chen et al., 2024; Fernando et al., 2024; Guo et al., 2024; Yang et al., 2024; Wang

*Corresponding Author

Findings of the Association for Computational Linguistics: ACL 2026, pages 19928–19947

et al., 2024; Ye et al., 2024; Hu et al., 2024; Wu et al., 2024). These strategies can generally be categorized into two main types: *continuous* and *discrete* prompt optimization approaches. **Continuous methods**, such as InstructZero (Chen et al., 2024) and Instinct (Lin et al., 2024), reformulate the prompt optimization problem in a continuous space by leveraging soft prompts. Other continuous approaches, such as ZOPO (Hu et al., 2024), adopt a local optimization perspective by generating a large pool of candidate prompts along with their corresponding embeddings. They then apply localized zeroth-order optimization to identify locally optimal prompts. In contrast, **discrete methods**, such as APE (Zhou et al., 2023), OPRO (Yang et al., 2024), PromptBreeder (Fernando et al., 2024) and EvoPrompt (Guo et al., 2024)—generate diverse prompt variants using scoring mechanisms or through evolutionary and self-referential strategies. **However, both continuous and discrete prompt optimization methods typically apply the same prompt across all samples within a dataset, thereby overlooking the varying levels of difficulty among individual samples, as illustrated in the left part of Fig. 1.** Specifically, for relatively simple instances, this can lead to over-optimization—resulting in unnecessary computational overhead and, in some cases, even degraded performance due to the use of overly complex prompts. Conversely, for more challenging samples, prompts may be under-optimized, as the presence of easier examples can cause the optimization algorithm to converge prematurely.

To further investigate the aforementioned issues, we conduct a comprehensive empirical study on prompt optimization (Sec. 3). Specifically, we examined the proportions of samples passed and failed under the base instruction (*i.e.*, the initial instruction without any optimization) across three tasks in the Instruction Induction dataset (Honovich et al., 2023), as well as the changes in these proportions after applying discrete prompt optimization APE (Zhou et al., 2023) and continuous prompt optimization ZOPO (Hu et al., 2024). We observed that after optimization with either APE or ZOPO, a significant portion of samples that initially succeeded under the base instruction began to fail, while many of the originally challenging samples continued to fail. This highlights considerable room for further improvement. This finding supports our hypothesis that existing optimization algorithms, which focus on dataset-level prompt optimization (Sec. 2), can negatively affect simple samples and lead to degraded performance, while simultaneously failing to sufficiently optimize complex samples.

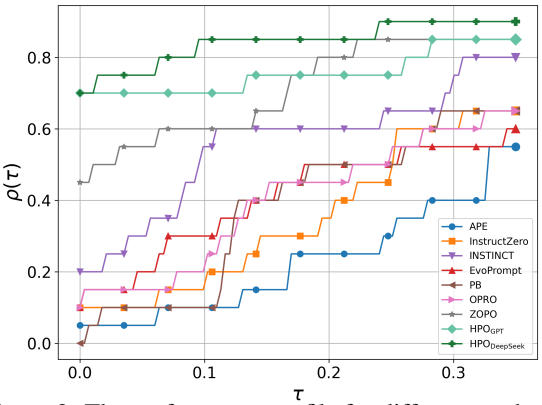

Figure 2: The performance profile for different methods on 20 instruction induction tasks. A higher $\rho(\tau)$ is better. More details in Sec. 5.

Motivated by these insights, we propose a novel ***HI**erarchical **P**rompt **O**ptimization* framework (**HIPO**) (Fig. 4), which achieves substantially improved prompt optimization performance, as illustrated in Fig. 2. Unlike prior works that predominantly focus on dataset-level prompt optimization, our approach shifts toward sample-level optimization, enabling finer-grained optimization (Fig. 1, where color intensity indicates the level of difficulty or complexity). Specifically, we classifies prompts into five complexity levels, allowing samples of varying difficulty to be paired with appropriately complex prompts. Notably, in the second level, we introduce heuristic, user-configurable thinking styles, which enable iterative refinement of task descriptions and allow HIPO to generalize across a wide range of natural language processing tasks. Additionally, in the fifth level, we incorporate attribution analysis to identify and refine issues within problem descriptions, thereby enhancing the overall performance of HIPO.

In summary, our main contributions are as follows:

- To the best of our knowledge, we are the first to conduct a thorough empirical study comparing dataset-level and sample-level prompt optimization strategies. Our findings highlight the necessity of shifting from dataset-level to sample-level optimization to achieve more efficient and effective prompt optimization (Sec. 3).
- Drawing on the insights gained from our empirical analysis, we propose HIPO based on sample-level strategies, which significantly outperforms

existing baselines in both optimization performance and query efficiency (Sec. 4).

- We conduct extensive experiments across a diverse set of natural language processing tasks to validate the effectiveness, scalability, and low-overhead nature of our proposed framework (Sec. 5).

2 Problem Formulation

The goal of prompt optimization is to find the textual prompt p^* that achieves the best performance on a given dataset D when using a given LLM $\mathcal{M}_{task}$ as the task model. More specifically, assume that all datasets can be formatted as textual input-output pairs, *i.e.*, $D = \{(x, y)\}$. A training set D_{train} is provided for optimizing the prompt, along with a validation set D_{val} for validation and a test set D_{test} for final evaluation. Following the notations in Zhou et al. (2023), the prompt optimization problem can be described as:

$$p^* = \arg\max_{p} \sum_{(x,y)\in D_{train}} f(\mathcal{M}_{task}(x; p), y), \tag{1}$$

where $\mathcal{M}_{task}(x; p)$ is the output generated by the task model when conditioning on the prompt p, and f is a per-example evaluation function. For example, if the evaluation metric is exact match, $f(\mathcal{M}_{task}(x; p), y) = \mathbb{1}[\mathcal{M}_{task}(x; p) = y]$. After obtaining p^*, it is subsequently evaluated on D_{test}. We refer to the optimization strategy defined in Equation 1 as dataset-level prompt optimization, as it yields a single optimized prompt for the entire dataset. However, we argue that this paradigm overlooks the varying levels of difficulty across individual samples. In practice, it often leads to over-optimization for easier instances, which results in unnecessary computational overhead or even degraded performance, while more challenging examples may suffer from under-optimization due to premature convergence. We empirically validate these insights in Sec. 3. Furthermore, to address this issue, we propose a new paradigm called sample-level prompt optimization, which adapts the prompt to each individual input. We formulate as follows:

$$p_x^* = \arg\max_{p} f(\mathcal{M}_{task}(x; p), y), \quad \forall (x, y) \in D_{test}. \tag{2}$$

In this formulation, the optimal prompt p_x^* is determined for each individual input x, allowing the model to dynamically adapt prompts based on instance-specific difficulty or characteristics.

3 Empirical Study on Prompt Optimization

3.1 Motivation and Background

Most prior works on prompt optimization adopt a dataset-level paradigm (Zhou et al., 2023; Lin et al., 2024; Chen et al., 2024; Fernando et al., 2024; Guo et al., 2024; Yang et al., 2024; Wang et al., 2024; Ye et al., 2024; Hu et al., 2024; Wu et al., 2024) (Sec. 2), where the goal is to identify a single optimized prompt that performs best on a given task dataset. Specifically, the entire dataset is treated as a whole, and the optimization process attempts to find a universal prompt p^* that maximizes the average performance across a validation set. While this approach has shown promising results in various benchmarks, it implicitly assumes that all instances in the dataset can be equally well served by the same prompt. However, this assumption often fails in practice. Task datasets typically contain instances of varying difficulty. A prompt that works well for easy examples may not generalize to harder ones, and vice versa.

3.2 Experimental Setup

We design our empirical study to answer the following question: ***Does dataset-level prompt optimization disproportionately benefit easy examples while failing to adequately optimize harder ones?*** To answer the question, we select three subtasks from the Instruction Induction dataset (Honovich et al., 2023): `antonyms`, `negation`, and `taxonomy_animal`. For each subtask, we first estimate the difficulty of each sample by evaluating whether a fixed vanilla prompt p (*e.g.*, *"Output the list of animals."*) can elicit the correct output from the model. Samples that are correctly answered under this vanilla prompt are labeled as *easy*, while those that are not are labeled as *hard*, resulting in two subsets per subtask: easy samples and hard samples. Next, we follow the dataset-level optimization paradigm described in Equation 1 to derive a single optimized prompt p^* for each subtask. Specifically, we apply both the APE and ZOPO algorithms to obtain p^* based on the full dataset of each subtask. The evaluation function f is defined as exact match accuracy, and we use ChatGPT-3.5-Turbo as the task model $\mathcal{M}_{task}$. Finally, we evaluate the performance gain of the optimized

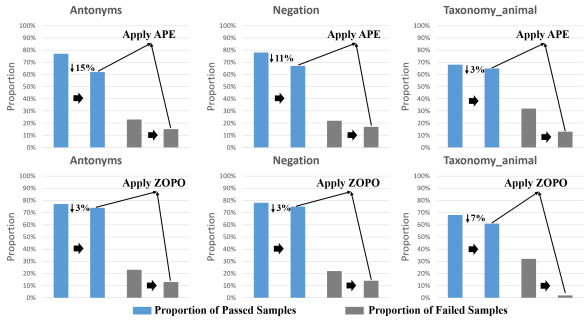

Figure 3: Proportions of passed and failed samples under the vanilla prompt on three Instruction Induction tasks, and changes after applying APE (discrete) and ZOPO (continuous) prompt optimization.

prompt p^* over the vanilla prompt p on both the easy and hard subsets of each subtask. By comparing the accuracy improvement (Δ accuracy) in these two subsets, We aim to empirically verify whether dataset-level optimization indeed harms the performance on easier examples while offering limited benefits for harder ones.

3.3 Results and Findings

The results, as shown in Fig. 3, reveal that for *easy* samples(blue bars), which already achieve perfect performance under vanilla prompt, the use of optimized prompts often leads to performance degradation. This holds true regardless of whether the optimized prompt is produced by a discrete or continuous optimization algorithm. For instance, the instruction optimized by APE results in performance drops across all three datasets, with a notable decline of 15% observed on the `antonyms` task. A similar pattern is observed for the instruction optimized by ZOPO. In contrast, *hard* samples(grey bars), which have more room for improvement, benefit less from dataset-level optimization. Specifically, many samples that fail under the vanilla prompt continue to fail even after applying prompts optimized by APE or ZOPO. ***These findings indicate existing optimization algorithms, which focus on dataset-level prompt optimization (Sec. 2), may harm simple samples by degrading their performance, while failing to adequately improve the performance on complex ones.***

4 The Methodology

Inspired by the insights established in Sec 3, we propose HIPO, a hierarchical prompt optimization framework designed to tailor prompt complexity to individual sample difficulty. HIPO employs a two-stage process: an **offline training stage** for a difficulty router and an **online inference stage** for efficient, sample-aware prompt generation, as shown in Fig 4.

4.1 HIPO Framework Design

The core objective of HIPO is to select the most appropriate prompt for each sample, balancing between avoiding over-optimization on easy cases and enhancing performance on hard ones. To this end, we design five levels of prompting, each tailored to different sample complexities. A representative example from a code generation task is shown in Fig. 5 to illustrate the layered architecture.

The five distinct levels of prompt complexity in HIPO are grounded in Bloom's taxonomy of human cognitive functions (Bloom et al., 1956), which organizes cognitive processes into six hierarchical dimensions in ascending order of complexity: Remembering, Understanding, Applying, Analyzing, Evaluating, and Creating. HIPO systematically evaluates the extent to which each prompt level influences these cognitive dimensions and defines the five prompt levels in the following order: **Level 1: Expert Profile Prompting**, **Level 2: Task Description Prompting**, **Level 3: Few-shot Chain-of-Thought (CoT) Prompting**, **Level 4: Generated Knowledge Prompting**, and **Level 5: Attribution-based Prompting** (see more details in Appx. A). These levels are designed to progressively engage deeper cognitive processes.

4.1.1 Task Description Prompt Refine

This stage aims to iteratively refine the task description through a systematic, feedback-driven process. Specifically, the refinement procedure consists of the following four key modules:

(1) Mixing & Mutate Module: Given an initial task description, this module generates a set of prompt variations by combining the original instruction with user-defined heuristic *thinking styles* (see more details in Appx. E). These heuristics guide the LLM to reinterpret or reframe the task from diverse perspectives, resulting in richer and more instructive prompt candidates. These are then passed to the next module as candidate prompts.

(2) Scoring Module: This component evaluates each candidate prompt based on its performance over a randomly sampled mini-batch of 5 training examples with known ground truths. Specifically, the prompt that successfully solves the highest number of training examples in the mini-batch

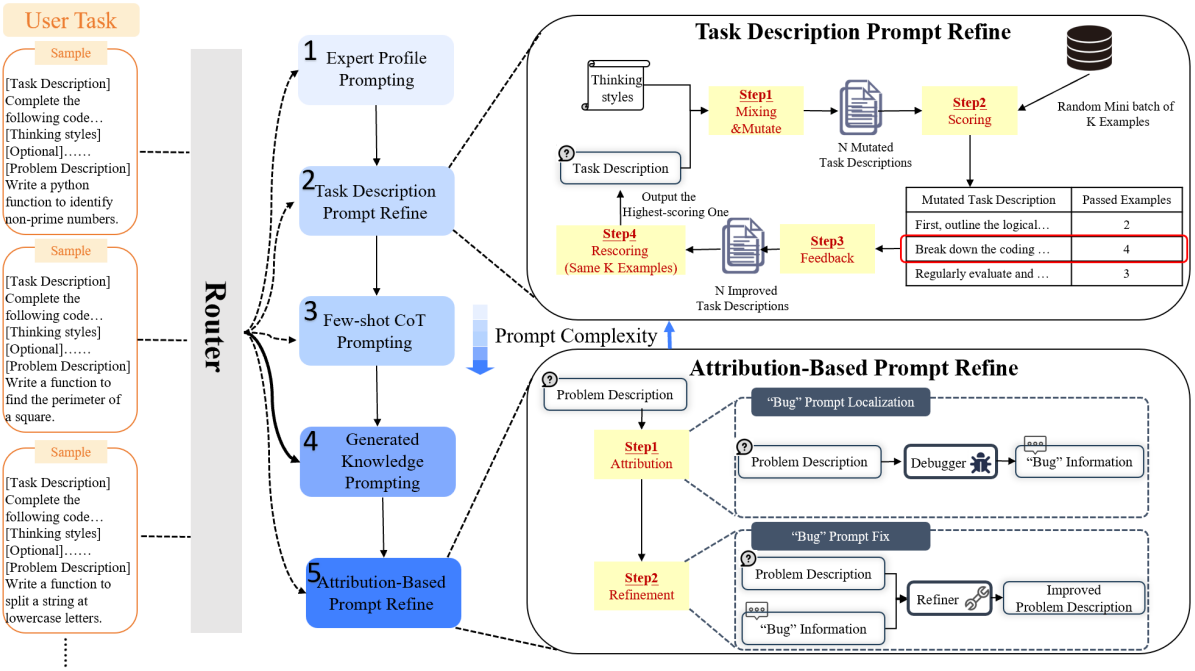

Figure 4: Overview of our framework's architecture. In an **offline stage**, we leverage our five-level prompt hierarchy, ranging from simple Expert Profiles to complex Attribution-based Refinement, to generate training data. This data is used to train a lightweight **router model** capable of predicting sample difficulty. In the **online inference stage** (depicted here), the router receives a problem description and directly predicts the optimal complexity level. HIPO then constructs a single, targeted prompt corresponding to that level for efficient, non-iterative execution. This design enables sample-aware prompt optimization.

Hierarchical Prompt Structure

You are a seasoned software engineer with a strong background in algorithm design and computational problem-solving. You specialize in creating efficient……	①Expert Profile
Complete the provided code, first ensure a deep understanding of the problem requirements and expected output. Break the problem into smaller parts and draft pseudocode……	②Task Description
def order_by_points(nums): """ Sorts a list of integers based on the sum of their absolute digits (ignoring signs), and in case of ties, preserves their original order. """ return sorted(nums, key=lambda x: (sum(map(int, str(abs(x)))), nums.index(x)))	③Demonstration Examples
The function order_by_points(nums) sorts a list of integers by the sum of the digits of their absolute values in ascending order. If multiple numbers……	④Generated Knowledge

def order_by_points(nums):
"""
Write a function which sorts the given list of integers in ascending order according to the sum of their digits. Note: if there are several items with similar sum of their digits, order them based on their index in original list.
For example:
>>> order_by_points([1, 11, -1, -11, -12]) == [-1, -11, 1, -12, 11]
"""
	⑤Problem Description (Specific Sample Need To Be Solved)

Figure 5: Illustration of the five-level prompting strategies in a code generation task.

is selected as the best-performing candidate. The scoring mechanism can be task-specific or use an LLM-based evaluator. Both approaches are supported. The use of randomly sampled mini-batches enhances the robustness of evaluation and avoids overfitting to a fixed set of examples.

(3) Feedback Module: If the best-performing prompt fails to pass all examples in the mini-batch, the failed examples are recorded. For code generation tasks, this includes error messages; for other NLP tasks, it involves logging mismatches with the ground truth. These failure cases are then provided as additional context to the LLM, which is tasked with further refining the current task description. The resulting refined prompts are appended to the candidate instruction pool for re-evaluation.

(4) Rescoring Module: Finally, the newly generated candidate prompts from the feedback loop are re-evaluated using the same scoring procedure as in Step (2). The most effective task description is then selected based on its overall performance, concluding the iterative refinement process.

4.1.2 Attribution Based Prompt Refine

This stage aims to refine problem descriptions via a two-step, attribution-driven process to improve clarity, specificity, and instructional effectiveness. The core assumption is that prompts, like traditional programs for computers, can be seen as "programs" for language models. This stage first identifies potential "bugs" in the prompt through attribution analysis, then applies targeted refinements accordingly. It consists of two main components:

(1) Attribution Module: Given a raw *problem description* (Fig. 5), this module identifies salient and influential components—typically key words or phrases—that significantly impact the model's output (see App. F). Instead of relying on gradients, we adopt a black-box, model-consistent strategy: the LLM itself is asked, via carefully designed meta-prompts, to introspect and verbalize which parts of the prompt most likely influence its own reasoning. This reflective querying approach draws on the LLM's internal alignment between linguistic

cues and output behavior, enabling reliable attribution without requiring access to internal states.

(2) Refinement Module: Using an LLM as the refiner, this module improves the original problem description based on the identified attributed components. It performs targeted edits to enhance the instruction (see details in Appx. F). The result is a optimized version of the original problem description instruction. By grounding refinement in attribution, this process ensures changes are purposeful and semantically aligned with the problem's core intent.

4.2 Offline Stage: Training the Router

The core of our efficient sample-level assignment is a lightweight **router model**. The purpose of this model is to learn a mapping from a given sample to one of the five complexity levels in our HIPO hierarchy. To achieve this, we first need to generate a training dataset where each sample is labeled with its "optimal" prompt level.

We generate these labels by running our full HIPO pipeline on the *training set*. For each training sample, we iteratively apply prompts from Level 1 to Level 5. After each level, we use the ground-truth label to check if the generated output is correct. The first level at which the sample is solved correctly is recorded as its optimal complexity label. This process yields a labeled dataset of the form $\{(x_i, y_i, l_i)\}_{i=1}^{N}$, where x_i is the input sample, y_i is its ground-truth output, and $l_i \in \{1, 2, 3, 4, 5\}$ is its optimal HIPO level.

Using this labeled data, we then train a simple yet effective text classification model as our router. The router takes a new, unseen sample as input and predicts its required prompt complexity level, l_{pred}.

4.3 Online Stage: Sample-Aware Inference

At test time, the HIPO execution flow is highly efficient. An unseen test sample is first passed to our pre-trained router model to predict the necessary prompt complexity level, l_{pred}. The framework then constructs a single, targeted prompt by cumulatively combining all components from Level 1 up to the predicted level l_{pred}. Finally, this tailored prompt is used with the task model ($\mathcal{M}_{task}$) to generate the final answer.

5 Experiments

5.1 Datasets and Baselines

We evaluate the effectiveness of HIPO on a total of 27 tasks, including 20 tasks from the Instruction Induction benchmark (Honovich et al., 2023), 2 code generation tasks: HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021), 2 sentiment analysis tasks: SST-2 (Socher et al., 2013) and SST-5 (Socher et al., 2013), and 3 arithmetic reasoning tasks: GSM8K (Cobbe et al., 2021), AQUARAT (Ling et al., 2017), and SVAMP (Patel et al., 2021).

We compare HIPO against seven representative state-of-the-art (SOTA) methods, covering both continuous and discrete prompt optimization approaches: Instinct (Lin et al., 2024), InstructZero (Chen et al., 2024), ZOPO (Hu et al., 2024), PromptBreeder (Fernando et al., 2024), EvoPrompt (Guo et al., 2024), APE (Zhou et al., 2023), and OPRO (Yang et al., 2024). Specifically, for the Instruction Induction benchmark, we use the performance profile (Dolan and Moré, 2002), defined in Appx. C.1, as the overall evaluation metric that measures the frequency (i.e., $\rho(\tau)$) of a method within some distance (i.e., τ) from the highest accuracy achieved by any method.

5.2 Implementation Details

We conduct experiments using both proprietary and open-source large language models. For the proprietary model, we use ChatGPT-3.5-Turbo to ensure fair comparison and budget cost. For the open-source counterpart, we adopt DeepSeek-V3. All evaluations are conducted on the full test sets of each dataset (see more details in Appx. C.2). To ensure robustness, we report the average performance over three independent runs. Specifically, we limit the number of thinking styles to 5 and use a mini-batch size of 5 training examples during prompt scoring and refinement.

5.3 Experimental Results And Analysis

Superior performance of HIPO. For better comparison, we follow the experimental setup of Lin et al. (2024) and report results on 20 challenging tasks. All experiments are conducted using the same black-box LLM (`GPT3.5Turbo`) under a zero-shot setting, ensuring fair and consistent comparisons across all methods. As shown in Table 1, HIPO outperforms all baselines, achieving the highest accuracy on 14 out of 20 tasks (70%), compared

Table 1: Average test accuracy achieved by the best instruction generated by different SOTA algorithms on 20 instruction induction tasks. We **bold** the highest accuracy when comparing HIPO_{GPT} with baseline methods. *Note: "GPT" refers to ChatGPT-3.5-Turbo.*

Tasks	APE	InstructZero	INSTINCT	EvoPrompt	PB	OPRO	ZOPO	HIPO_{GPT}
antonyms	63.7	82.7	84.7	84.0	78.0	79.0	85.2	**88.0**
auto_categorization	25.0	25.7	25.0	31.0	24.0	24.0	32.7	**33.0**
auto_debugging	29.2	37.5	29.2	33.0	25.0	37.5	41.7	**50.0**
cause_and_effect	57.3	81.3	58.7	84.0	82.7	82.7	**94.7**	68.0
common_concept	6.9	8.6	21.3	11.1	10.9	8.6	**23.5**	3.3
diff	67.3	69.3	**100.0**	27.3	71.3	**100.0**	**100.0**	**100.0**
informal_to_formal	57.4	53.1	55.3	51.6	54.2	48.0	**61.3**	32.6
letters_list	**100.0**	59.0	**100.0**	**100.0**	99.3	99.7	**100.0**	**100.0**
negation	75.3	77.7	81.7	86.0	70.7	73.3	**86.3**	64.0
object_counting	36.3	36.0	34.0	55.0	29.3	36.0	52.3	**67.0**
odd_one_out	63.3	61.3	70.0	10.0	66.7	47.3	32.0	**76.0**
orthography_starts_with	45.7	50.7	66.7	15.0	59.8	33.5	56.5	**68.0**
rhymes	15.7	**100.0**	**100.0**	59.7	45.0	23.0	**100.0**	20.0
second_word_letter	74.7	43.3	10.0	24.7	88.7	86.7	25.7	**100.0**
sentence_similarity	0.0	0.0	14.0	2.0	0.0	2.7	7.6	**20.0**
sum	67.3	**100.0**	**100.0**	**100.0**	98.3	**100.0**	**100.0**	**100.0**
synonyms	36.0	27.7	30.7	40.3	36.3	40.0	**43.3**	37.5
taxonomy_animal	34.7	71.7	85.7	83.0	29.7	30.0	90.0	**96.0**
word_sorting	33.0	31.0	51.3	48.0	45.7	50.3	60.0	**74.0**
word_unscrambling	44.0	55.0	63.3	51.3	51.0	61.3	59.3	**69.0**

Table 2: Generalization results of HIPO across multiple datasets and task domains. Evaluation metrics for different tasks are detailed in Appx. C.1.

Task	Dataset	Orig.	HIPO	Imp.
Code Generation	HumanEval	92.6	100	+7.4
	MBPP	84.2	92.5	+8.3
	Average	**88.4**	**96.3**	**+7.9**
Sentiment Analysis	SST-2	94.7	95.9	+1.2
	SST-5	54.8	68.6	+13.8
	Average	**74.8**	**82.3**	**+7.5**
Arithmetic Reasoning	GSM8K	92.8	95.7	+2.9
	AQUARAT	88.6	89.8	+1.2
	SVAMP	94.7	95.0	+0.3
	Average	**92.0**	**93.5**	**+1.5**

to ZOPO's 9 tasks (45%). Moreover, HIPO attains the best performance profile across varying τ values (Fig. 2), further demonstrating its robustness.

Generalization of HIPO. In real-world applications, the generalizability of automatic prompt optimization algorithms is critical for their practical adoption. To evaluate the robustness and task-level generalization capability of HIPO, we conduct experiments on seven datasets spanning three representative task domains: two code generation datasets, two sentiment classification datasets, and three arithmetic reasoning datasets. Due to budget constraints, we evaluate HIPO only under the DeepSeek-V3 backbone. As shown in Table 2, HIPO improves performance across all evaluated datasets.

Table 3: Cost analysis of various prompt optimization methods, averaged per task across the 20 tasks in the Instruction Induction benchmark. HIPO achieves competitive performance with the lowest API calls and overall cost.

Method	API Calls	Total Tokens	Cost($)
APE	757	3.8M	7.6
InstructZero	1730	0.11M	0.23
INSTINCT	1730	0.11M	0.23
EvoPrompt	5000	0.4M	0.8
PB	18600	1.49M	2.9
ZOPO	1514	7.6M	15.2
HIPO	**681**	**0.09M**	**0.19**

Cost Analysis of HIPO. While achieving high accuracy is essential, the efficiency of prompt generation is equally critical for real-world applications. To this end, we perform an in-depth cost analysis showing that HIPO not only achieves superior accuracy compared to baseline methods, but also incurs minimal computational overhead. Our evaluation includes metrics such as the total number of API calls, tokens processed, and estimated cost.

As shown in Table 3, HIPO requires the fewest API calls among all methods, consumes the least number of tokens, and achieves a cost that is 1.2x to 80x cheaper compared to continuous methods like

Table 4: Ablation study showing performance after progressively adding each prompt level in the HIPO pipeline.

Configuration	II	MBPP	SST-5	GSM8K
Base (no prompt levels)	50.6	84.2	54.8	92.8
w/ level1	58.0	84.2	54.9	93.6
w/ level1+2	61.4	87.8	62.9	94.7
w/ level1+2+3	66.3	90.8	67.6	95.6
w/ level1+2+3+4	66.8	92.3	68.1	95.6
w/ level1+2+3+4+5 (full)	**67.9**	**92.5**	**68.6**	**95.7**

INSTINCT, InstructZero, and ZOPO, and 4x to 40x cheaper compared to discrete methods like APE, EvoPrompt, and PB—highlighting its practicality and cost-effectiveness.

5.4 Ablation Study

We conducted an ablation study to assess the incremental impact of each prompt level in HIPO. Starting from a base configuration, we progressively added each level and evaluated performance on Instruction Induction (20 tasks), MBPP, SST-5, and GSM8K. As shown in Table 4, each added level consistently improves performance, validating the effectiveness of the hierarchical design.

6 Related Work

Continuous Prompt Optimization. Continuous methods such as InstructZero (Chen et al., 2024) and Instinct (Lin et al., 2024) treat prompt optimization as a continuous learning problem by using soft prompts, which are trainable embeddings prepended to inputs to guide open-source LLMs. These methods often adopt feedback-driven strategies like Bayesian optimization or neural networks to iteratively refine the prompt vectors. ZOPO (Hu et al., 2024) offers a more efficient alternative by encoding discrete prompts with model embeddings and using a Neural Tangent Kernel-based Gaussian Process to guide zeroth-order optimization toward locally optimal, high-performing regions. While these approaches offer flexibility and adapt well to specific models, they face key limitations: (i) additional training incurs high computational costs, (ii) soft prompts lack interpretability, reducing transparency, and (iii) in complex reasoning tasks, the unclear link between embeddings and performance often leads to inconsistent results.

Discrete Prompt Optimization. Discrete prompt optimization aims to improve prompt quality by manipulating token sequences, which is especially relevant for black-box LLMs that only accept textual (i.e., discrete) inputs. A common approach is to generate multiple candidate prompts and select the best through heuristic or search-based strategies. Early work (Deng et al., 2022b; Zhang et al., 2023) used reinforcement learning to optimize prompts based on LLM output distributions. However, such methods are less applicable to black-box APIs (e.g., ChatGPT), where internal distributions are inaccessible. To address this, newer approaches avoid reliance on internals. Zhou et al.(Zhou et al., 2023) use a Monte Carlo framework to iteratively resample and select candidate prompts, while EvoPrompt(Guo et al., 2024) and PromptBreeder (Fernando et al., 2024) apply model-free evolutionary algorithms with mutations and crossovers, guided by meta-prompts. Other innovations include implicit optimization methods: OPRO (Yang et al., 2024) evaluates prompts on fixed datasets to drive refinement, and Pryzant et al. (Pryzant et al., 2023) guide LLMs to perform gradient-like refinements implicitly. While effective, these methods often require many queries and rely on powerful LLMs like GPT-3.5, making them resource-intensive.

7 Conclusion

We introduce HIPO, a framework for dynamic, sample-level prompt engineering that uses an offline-trained router for efficient inference. Experiments show HIPO outperforms state-of-the-art methods and is 1.2× to 80× more cost-effective. Future work could extend this approach to other modalities, such as image and speech generation.

8 Limitations

While HIPO demonstrates consistent improvements in both performance and efficiency, we note several limitations that suggest directions for future work.

First, the effectiveness of the framework depends on the quality of the lightweight router model. Although the router is inexpensive to train and performs robustly in our experiments, its predictions may be affected by the diversity and coverage of the offline training data. Improving the router with richer supervision signals or more adaptive training strategies could further enhance routing accuracy.

Second, HIPO relies on a fixed hierarchical prompt structure with a predefined number of complexity levels. While this design is simple and effective across a wide range of tasks, more flexible or adaptive prompt hierarchies may better capture

task-specific characteristics. Exploring automatic ways to adjust the number or form of prompt levels is a promising direction for future research.

Acknowledgements

This work was supported partly by National Key Research and Development Program of China under Grant No. 2024YFB3309602, partly by National Natural Science Foundation of China under Grant No. 62472017, and partly by Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Benjamin S Bloom and 1 others. 1956. Taxonomy of. *Educational Objectives*.

Lichang Chen, Jiuhai Chen, Tom Goldstein, Heng Huang, and Tianyi Zhou. 2024. Instructzero: efficient instruction optimization for black-box large language models6518. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

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

Pierre Colombo, Telmo Pessoa Pires, Malik Boudiaf, Dominic Culver, Rui Melo, Caio Corro, Andre F. T. Martins, Fabrizio Esposito, Vera Lúcia Raposo, Sofia Morgado, and Michael Desa. 2024. Saullm-7b: A pioneering large language model for law. *Preprint*, arXiv:2403.03883.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *Preprint*, arXiv:2412.19437.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

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

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

Elizabeth D Dolan and Jorge J Moré. 2002. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91:201–213.

Chrisantha Fernando, Dylan Banarse, Henryk Michalewski, Simon Osindero, and Tim Rocktäschel. 2024. Promptbreeder: self-referential self-improvement via prompt evolution. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

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

Or Honovich, Uri Shaham, Samuel R. Bowman, and Omer Levy. 2023. Instruction induction: From few examples to natural language task descriptions. In *Proc. ACL*, pages 1935–1952.

Wenyang Hu, Yao Shu, Zongmin Yu, Zhaoxuan Wu, Xiaoqiang Lin, Zhongxiang Dai, See-Kiong Ng, and Bryan Kian Hsiang Low. 2024. Localized zeroth-order prompt optimization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059.

Xiaoqiang Lin, Zhaoxuan Wu, Zhongxiang Dai, Wenyang Hu, Yao Shu, See-Kiong Ng, Patrick Jaillet, and Bryan Kian Hsiang Low. 2024. Use your INSTINCT: Instruction optimization for llms using neural bandits coupled with transformers. In *Proc. ICML*.

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

Jiacheng Liu, Alisa Liu, Ximing Lu, Sean Welleck, Peter West, Ronan Le Bras, Yejin Choi, and Hannaneh Hajishirzi. 2022. Generated knowledge prompting

for commonsense reasoning. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3154–3169, Dublin, Ireland. Association for Computational Linguistics.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9).

Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2021. Reframing instructional prompts to gptk's language. *ACL Findings*, pages 589–612.

OpenAI, R, and other et. al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

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 *The 2023 Conference on Empirical Methods in Natural Language Processing*.

Laria Reynolds and Kyle McDonell. 2021. Prompt programming for large language models: Beyond the few-shot paradigm. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–7.

Taylor Shin, Yasaman Razeghi, Robert L Logan IV, Eric Wallace, and Sameer Singh. 2020. Autoprompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Zhen Wang, Rameswar Panda, Leonid Karlinsky, Rogerio Feris, Huan Sun, and Yoon Kim. 2022. Multitask prompt tuning enables parameter-efficient transfer learning. In *The Eleventh International Conference on Learning Representations*.

Yurong Wu, Yan Gao, Bin Benjamin Zhu, Zineng Zhou, Xiaodi Sun, Sheng Yang, Jian-Guang Lou, Zhiming Ding, and Linjun Yang. 2024. StraGo: Harnessing strategic guidance for prompt optimization. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10043–10061, Miami, Florida, USA. Association for Computational Linguistics.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Michihiro Yasunaga, Xinyun Chen, Yujia Li, Panupong Pasupat, Jure Leskovec, Percy Liang, Ed H. Chi, and Denny Zhou. 2024. Large language models as analogical reasoners. In *The Twelfth International Conference on Learning Representations*.

Qinyuan Ye, Mohamed Ahmed, Reid Pryzant, and Fereshte Khani. 2024. Prompt engineering a prompt engineer. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 355–385, Bangkok, Thailand. Association for Computational Linguistics.

Tianjun Zhang, Xuezhi Wang, Denny Zhou, Dale Schuurmans, and Joseph E Gonzalez. 2022. Tempera: Test-time prompt editing via reinforcement learning. In *The Eleventh International Conference on Learning Representations*.

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

Xinlu Zhang, Chenxin Tian, Xianjun Yang, Lichang Chen, Zekun Li, and Linda Ruth Petzold. 2024. Alpacare:instruction-tuned large language models for medical application. *Preprint*, arXiv:2310.14558.

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

A Cognitive Theory

According to Bloom's Revised Taxonomy of the Cognitive Domain (Bloom et al., 1956), cognitive processes are categorized into six hierarchical levels: **Remembering**, **Understanding**, **Applying**, **Analyzing**, **Evaluating**, and **Creating**. Higher levels in this hierarchy represent increasing levels of cognitive complexity. The six cognitive dimensions are described as follows:

- **Remembering**: Retrieving relevant knowledge from long-term memory. For example, recalling the dates of important historical events or identifying the components of a bacterial cell. Common verbs include *cite*, *define*, *describe*, *identify*, *list*, and *recall*.
- **Understanding**: Constructing meaning from instructional messages through interpretation, classification, comparison, inference, and other processes. For instance, classifying types of diseases or comparing rituals of different religions. Typical verbs include *classify*, *compare*, *contrast*, *explain*, *paraphrase*, and *summarize*.
- **Applying**: Using learned information or procedures in new situations. For example, applying Newton's First Law to solve a physics problem or performing multivariate statistical analysis on a new dataset. Associated verbs include *apply*, *calculate*, *demonstrate*, *execute*, *solve*, and *use*.
- **Analyzing**: Breaking material into constituent parts and determining how the parts relate to one another and to an overall structure or purpose. This might involve analyzing relationships between organisms in an ecosystem or between characters in a story. Common verbs include *analyze*, *break down*, *categorize*, *compare*, *connect*, and *differentiate*.
- **Evaluating**: Making judgments based on criteria and standards. Examples include detecting inconsistencies in a process or assessing whether a scientific conclusion is supported by data. Frequently used verbs are *assess*, *compare*, *criticize*, *evaluate*, *judge*, and *justify*.
- **Creating**: Putting elements together to form a novel, coherent, or functional whole, or reorganizing elements into a new pattern or structure. This includes designing an original piece of art, writing a report, or proposing a new hypothesis. Suitable verbs include *assemble*, *build*, *create*, *design*, *develop*, and *generate*.

In **HIPO**, we define five levels of prompting complexity, each mapped to different degrees of cognitive engagement based on Bloom's taxonomy: **Remembering**, **Understanding**, **Applying**, **Analyzing**, **Evaluating**, and **Creating**. The complexity of each prompt level is determined by the breadth and depth of its impact across these six cognitive dimensions.

Level 1: Expert Profile Prompting. This level defines the role or domain expertise of the language model (e.g., "You are a math teacher"). It is primarily contextual and does not actively stimulate higher-order cognitive functions. While it can provide subtle priming effects, it does not significantly require the model to engage in any of the six dimensions. Hence, its cognitive influence is minimal, and it represents the lowest complexity prompt.

Level 2: Task Description Prompting. This level involves presenting a high-level description of the task objectives, expectations, and constraints. Its primary impact is on:

- **Understanding:** By providing an explicit description of what is to be done, the prompt helps the model grasp task semantics, clarify ambiguity, and align its behavior with task intent.
- **Applying:** Task descriptions serve as an operational guide for the model, directing it to perform specific actions in accordance with the described requirements.

Level 3: Few-shot Chain-of-Thought (CoT) Prompting. This level leverages in-context examples that guide the model to reason through problems step-by-step. Following Yasunaga et al. (2024), it includes analogical problem examples and their solutions. The cognitive demands here include:

- **Applying:** The model must generalize from in-context examples and apply learned patterns to new inputs.
- **Analyzing:** CoT prompts require the model to decompose problems, establish intermediate reasoning steps, and draw logical inferences.

Compared to Level 2, this form of prompting increases complexity by introducing structured reasoning, thus engaging deeper levels of cognition.

Level 4: Generated Knowledge Prompting. Inspired by Liu et al. (2022), this level enhances prompts by incorporating external or model-generated factual or conceptual knowledge. This knowledge serves as scaffolding to support inference and address knowledge gaps. Its primary impact is on:

- **Analyzing:** The model must integrate external knowledge with task context and extract relevant information.
- **Evaluating:** It is also required to assess the relevance and correctness of injected knowledge before using it in downstream reasoning.

By introducing new, non-trivial knowledge components, this level increases the model's need to reason with, critique, and synthesize new information, thereby deepening cognitive engagement.

Level 5: Attribution-based Prompting. As the most sophisticated prompting strategy, this level dynamically refines problem phrasing by identifying and modifying its most influential components via attribution analysis (See Sec. 4.1.2). This allows for precise, feedback-driven prompt iteration. It engages all six cognitive dimensions, especially:

- **Evaluating:** The model must judge which parts of the prompt contribute most to performance, based on internal attribution signals or external metrics.
- **Creating:** It synthesizes revised prompts that incorporate insights from attribution, enabling problem redefinition or reformulation in a more effective form.

This level emphasizes meta-level reasoning, self-reflection, and adaptive improvement, marking it as the highest in prompt complexity.

In summary, the ascending order of our prompt levels reflects a gradual increase in cognitive engagement. Lower levels set context and ensure basic understanding, while higher levels require reasoning, critical evaluation, and creative reformulation—mirroring the progression of complexity in human learning tasks.

B Dataset details

B.1 Instruction Induction Dataset details

Table 5 presents 20 challenging tasks selected from the Instruction Induction dataset, along with their corresponding task descriptions. Instruction Induction is a widely used benchmark, and the selected tasks cover a broad range of language understanding abilities. Notably, they include all nine tasks from the BigBench-Hard subset. Specifically, the selected tasks span emotional understanding, context-free question answering, reading comprehension, summarization, algorithmic reasoning, and various types of reasoning tasks (e.g., arithmetic, commonsense, symbolic, and other forms of logical reasoning). We selected tasks for which the data is publicly available.

B.2 Code Generation Dataset details

We conduct experiments on three representative code generation benchmarks, including **HumanEval** and **MBPP**, both in Python. The details of these benchmarks are as follows:

- **HumanEval** (Chen et al., 2021) is a Python function-level code generation benchmark consisting of 164 hand-crafted programming problems. Each problem includes an English description, a function signature, and a set of test cases, with an average of 7.7 test cases per problem.
- **MBPP** (Austin et al., 2021) is another Python function-level benchmark comprising 974 problems that cover simple numeric manipulations and basic usage of standard libraries. Each problem is defined by an English requirement, a function signature, and three manually written test cases used for evaluation.

B.3 Sentiment Analysis Dataset details

We also evaluate our method on two widely-used sentiment analysis benchmarks: **SST-2** and **SST-5**. The details of these datasets are as follows:

- **SST-2** (Socher et al., 2013) is a binary sentiment classification benchmark derived from the Stanford Sentiment Treebank. Each sentence in the dataset is labeled as expressing either positive or negative sentiment. It consists of 67,349 training examples and 872 test examples.

Table 5: Instruction Induction Dataset

Task	Description
antonyms	Make the pairs of words opposite.
auto categorization	Create a list of things that the input could be associated with, and the output would be the category that the input belongs to
auto debugging	Identify and fix bugs in a given program, or provide the final output if no bugs are found
cause and effect	identify the sentence that is the cause of the effect in the input sentence pair
common concept	"involve" the objects mentioned in the input, so the answer would be "involve oscillations" for the input "guitars, pendulums"
diff	Find the difference between the two numbers
informal to formal	convert the input sentence into an output sentence that is grammatically correct and idiomatic in English
letters list	output the input with a space after each letter
negation	make the output false by adding the word "not" to the input
object counting	output the number of objects in the input list
odd one out	find the word that is most dissimilar to the others in the group
orthography starts with	output the word that starts with the letter that was inputted
rhymes	output the first word that appeared in the input text
second word letter	takes a string as input and returns the first character that is a vowel.
sentence similarity	Find the difference between the two sentences and the output was 4 - almost perfectly
sum	add the numbers of the two input numbers
synonyms	create a list of words that could be used in the same way as the original words
taxonomy animal	output the name of an animal that starts with the letter
word sorting	sort the input words alphabetically
word unscrambling	output the word that is formed by rearranging the letters of the given word

- **SST-5** (Socher et al., 2013) is a fine-grained sentiment classification task that extends SST-2 by categorizing sentences into five sentiment labels: very negative, negative, neutral, positive, and very positive. The dataset contains 8,544 training examples and 2,210 test examples.

B.4 Arithmetic Reasoning Dataset details

We further evaluate our method on arithmetic reasoning tasks using three representative datasets: **GSM8K**, **AQUARAT**, and **SVAMP**. The details of these datasets are as follows:

- **GSM8K** (Cobbe et al., 2021) is a collection of 8.5K high-quality, linguistically diverse grade school math word problems written by human annotators. Each problem requires a step-by-step solution leading to a final integer answer. The dataset contains 7,473 training examples and 1319 test examples.
- **AQUARAT** (Ling et al., 2017) is a large-scale dataset comprising approximately 100,000 algebraic word problems. Each solution is explained step-by-step using natural language, which makes it suitable for evaluating both reasoning ability and interpretability. The training set includes 97,467 examples and the test set includes 254 examples.
- **SVAMP** (Patel et al., 2021) is designed to test arithmetic reasoning on elementary-level problems by introducing simple variations to existing math word problems. It contains one-unknown arithmetic questions up to grade 4 level. The dataset consists of 700 training examples and 300 test examples.

C Details of Experimental Settings

C.1 Evaluation Metrics

For the **Instruction Induction** datasets, following prior works (Zhou et al., 2023; Lin et al., 2024; Hu et al., 2024), we use the F1 score for tasks including *common_concept* and *informal_to_formal*; we use the exact set matching for *orthography_starts_with* and *taxonomy_animal*; we use the set containing for *synonyms*; we use the exact matching metric for the rest of instruction induction tasks; and we use the accuracy metric for the arithmetic reasoning datasets.

Table 6: Test dataset sizes across different benchmark datasets

Datasets	Test dataset size
Instruction Induction ['auto debugging', 'cause and effect', 'common concept','informal to formal','odd one out']	8, 25, 16, 15, 50
Instruction Induction all except ['auto debugging', 'cause and effect', 'common concept','informal to formal','odd one out']	100
HumanEval	164
MBPP	974
SST-2	872
SST-5	2210
GSM8k	1319
AQUARAT	254
SVAMP	300

Given the large number of subtasks in this benchmark, we also use the performance profile (Dolan and Moré, 2002) as the evaluation metric that measures the frequency (i.e., $\rho(\tau)$) of a method within some distance (i.e., τ) from the optimality achieved by any method, defined below

$$\rho_m(\tau) = \frac{1}{|\Pi|} \left|\{\pi \in \Pi : r_\pi^* - r_{\pi,m} \leq \tau\}\right| \quad (3)$$

where Π is the set of all tasks, $r_{\pi,m}$ is the accuracy of method m on task π, and $r_\pi^* = \max\{r_{\pi,m} : \forall m \in \mathcal{M}\}$ is the best performance achieved by any method in $\mathcal{M}$ on task π. Specifically, $\rho(0)$ represents the number of tasks where a method achieves the best performance. In this curve, the x-axis (τ) represents the performance ratio relative to the best-performing method, and the y-axis ($\rho(\tau)$) reflects the fraction of tasks where a method's performance is within this ratio.

For **code generation** tasks, we use Pass@1 as the evaluation metric. For **sentiment analysis** and **arithmetic reasoning** tasks, we use accuracy as the evaluation metric.

C.2 Test Dataset Sizes Across Different Benchmarks

The test dataset sizes used in our experiments across different benchmarks are presented in Table 6 and correspond to the results reported in Sec. 5.3. For all datasets, during the scoring stage described in Sec. 4.1.1, we randomly sample 5 examples from the training set as demonstrations. Apart from this, HIPO does not require any additional training data.

C.3 Implementation Details

In the experiments presented in Table 1 of Sec. 5.3, we employ ChatGPT-3.5-Turbo (accessed via the OpenAI API) as the black-box model to ensure a fair comparison with seven other baseline methods. HIPO_{GPT} represents the performance of HIPO evaluated on 20 datasets using ChatGPT-3.5-Turbo. In Table 1, we include the results of seven baseline methods as reported by Hu et al. (Hu et al., 2024), who also use ChatGPT-3.5-Turbo as the black-box model.

In the experiments shown in Table 2 and Table 4 of Sec. 5.3, to reduce computational costs, we chose the more budget-friendly DeepSeek-V3-250324 as black-box model, whose input price is approximately \$0.278 per 1M tokens and output price is approximately \$1.111 per 1M tokens. *Note:* For comparison, the price of ChatGPT-3.5-Turbo is about \$0.50 per 1M input tokens and \$1.50 per 1M output tokens.

C.4 Details on Compute Resources

All experiments are conducted on a server equipped with dual Intel(R) Xeon(R) Gold 5218 CPUs (32 cores, 64 threads in total) and two NVIDIA A100-SXM4-80GB GPUs (160GB total GPU memory). We mainly perform prompt optimization on the GPT-3.5-Turbo model (for which OpenAI charges USD \$0.5 per 1M tokens for input and USD \$1.5 per 1M tokens for output) and the DeepSeek-V3-250324 model (for which Volcengine charges USD \$0.278 per 1M tokens for input and USD \$1.111 per 1M tokens for output). The execution time of our algorithm on each prompt optimization task (e.g., any task from the 20 Instruction Induction tasks, as well as tasks involving code generation, sentiment analysis, and arithmetic reasoning) varies depending on the number of examples in the dataset, the complexity of the task, and the response speed of the OpenAI or Volcengine APIs. For simpler datasets, our method may complete within a few minutes, whereas for more complex datasets, it may take 20–30 minutes or longer.

D Discussion of potential risks

While our proposed HIPO framework significantly enhances the efficiency and effectiveness of prompt optimization, it shares potential risks associated with automated prompt engineering technologies. First, there is a possibility of misuse; the ability to automatically refine and optimize prompts could theoretically be exploited to generate adversarial inputs or "jailbreak" prompts designed to bypass the safety alignment of Large Language Models (LLMs), potentially eliciting harmful or unethical content. Second, the framework relies on the inherent capabilities of the underlying task model and the router's training data. Consequently, any biases present in the training samples or the base LLM may be preserved or even amplified during the optimization process. We believe that future research should explore integrating safety constraints and alignment objectives directly into the prompt optimization loop to mitigate these risks.

E Thinking Styles

The thinking styles used by HIPO in the tasks of Code Generation, Sentiment Analysis, and Arithmetic Reasoning are listed as follows:

Thinking Styles for Code Generation

```
"Before completing the code, you should first write the function
    implementation process in pseudocode. At the same time, be sure to analyze
     the test cases in the specific problem. Based on the analysis of the test
     cases, complete the code."
"Before completing the code, you should first write the function
    implementation process in pseudocode. This pseudocode may include
    sequential, branch, and loop structures, and then complete the code."
"Break down the coding problem into smaller, manageable parts. Identify core
    components and focus on solving one part at a time."
"Begin by clearly understanding the problem requirements and constraints.
    Create a step-by-step plan before diving into the actual coding."
"Use comments to outline the structure of your code before writing the
    implementation. This helps clarify your logic and ensures you cover all
    necessary components."
```

Thinking Styles for Sentiment Analysis

```
"Focus on the emotional tone and intensity in the text. Think about whether
    the sentiment is strongly expressed or mild, and whether it is positive,
    negative, or neutral. This helps in distinguishing between 'very positive
    ', 'positive', and so on."
"Pay close attention to context and nuance. A word that seems negative might
    be used sarcastically or affectionately, depending on the context.
    Interpreting subtle emotional cues is key to fine-grained sentiment
    classification."
"Imagine how a human reader would feel after reading the text -- angry,
    indifferent, happy, extremely pleased, etc. Map that feeling to one of the
     sentiment categories. Empathizing with the text helps guide
    classification."
"Look at both individual emotional keywords and the overall structure of the
    sentence or paragraph. Are the negative words dominant, or are they
    outweighed by positive framing? Use the overall balance to decide the
    label."
"Think of the sentiment categories as a scale. Place each piece of text
    somewhere along that scale by estimating its emotional weight--not just
    whether it is positive or negative, but how much."
```

Thinking Styles for Arithmetic Reasoning

```
"Break down the problem step by step. Identify what is being asked, list known
     quantities, and determine what operations or formulas are needed to reach
     the answer. Clear stepwise thinking prevents missing key information."
"Translate word problems into equations or expressions. Focus on identifying
    numerical relationships and dependencies between quantities, especially
    when they are described in natural language."
"Check for consistency and units. Make sure intermediate steps make sense
    numerically and logically. If something feels off, retrace steps to locate
     potential misinterpretations or calculation errors."
"Use estimation to verify plausibility. After solving, ask yourself: is this
    result reasonable given the context? Estimation helps catch mistakes and
    build intuition for the problem."
"Look for patterns or analogies to familiar problems. Often, a new question
    can be solved using the structure of previously seen problems--recognizing
     these similarities can accelerate and deepen reasoning."
```

F Prompt Templates

F.1 Expert Profile Prompting

```
expert_template: |
For each instruction, write a high-quality description about the most capable
    and suitable agent to answer the instruction. In second person perspective
    .\n

[Instruction]: Make a list of 5 possible effects of deforestation.\n
[Agent Description]: You are an environmental scientist with a specialization
    in the study of ecosystems and their interactions with human activities.
    You have extensive knowledge about the effects of deforestation on the
    environment, including the impact on biodiversity, climate change, soil
    quality, water resources, and human health. Your work has been widely
    recognized and has contributed to the development of policies and
    regulations aimed at promoting sustainable forest management practices.
    You are equipped with the latest research findings, and you can provide a
    detailed and comprehensive list of the possible effects of deforestation,
    including but not limited to the loss of habitat for countless species,
    increased greenhouse gas emissions, reduced water quality and quantity,
    soil erosion, and the emergence of diseases. Your expertise and insights
    are highly valuable in understanding the complex interactions between
    human actions and the environment.

[Instruction]: Identify a descriptive phrase for an eclipse.\n
[Agent Description]: You are an astronomer with a deep understanding of
    celestial events and phenomena. Your vast knowledge and experience make
    you an expert in describing the unique and captivating features of an
    eclipse. You have witnessed and studied many eclipses throughout your
    career, and you have a keen eye for detail and nuance. Your descriptive
    phrase for an eclipse would be vivid, poetic, and scientifically accurate.
     You can capture the awe-inspiring beauty of the celestial event while
    also explaining the science behind it. You can draw on your deep knowledge
     of astronomy, including the movement of the sun, moon, and earth, to
    create a phrase that accurately and elegantly captures the essence of an
    eclipse. Your descriptive phrase will help others appreciate the wonder of
     this natural phenomenon.

[Instruction]: {task_description}
[Agent Description]:
Please ensure that your final output is a single paragraph without including
    the [Agent Description] label.
```

F.2 Task Description Prompting

```
mixing_mutate_template: |
You are given a task description and different thinking styles as follows:
[Task Description]: {task_description}
[Thinking Styles]: {thinking_styles}
Now you need to generate one variations of above task description adaptively
    mixing thinking styles while keeping similar semantic meaning with
    original Task Description.
Make sure to output only the final task descriptions as a paragraph without
    any [] labels, and wrap each generated task description with <START> and <
    END>.
[Generated Prompts]:
```

```
solve_template: |
You are given a prompt instruction and the following specific question of the
    same task.
[Instruction]: {instruction}
[Question]: {sample_problem}
Complete the [Question] based on the [instruction] and Make sure to output
    final answer wrapped with <ANS_START> and <ANS_END>.
The final answer should be wrapped in <ANS_START> and <ANS_END> and should
    only contain the final answer, without any reasoning or related
    explanations.

[Answers]:
```

```
feedback_refine_template: |
I am trying to craft a zero-shot instruction that will enable the most capable
     and suitable agent to solve the given task effectively.

My current prompt is: "{instruction}".
However, this prompt produces the following false examples:

Below, I have listed each example along with the agent's predicted answer and
    the ground truth answer

{examples_answer_reasons}
By carefully analyzing and comparing the predicted answers with the correct
    ones, identify the reasons behind the incorrect predictions.
Use these insights intelligently to refine the current prompt, ensuring that
    these examples are no longer yield erroneous outputs.
Based on the above analysis, I now want you to generate one improved versions
    of the prompt.

[Refined Prompt]:

Please note that the output Refined prompt should be strictly enclosed with <
    START> and <END>.
```

```
solve_template: |
You are given a prompt instruction and the following specific question of the
    same task.
[Instruction]: {instruction}
[Question]: {sample_problem}
Complete the [Question] based on the [instruction] and Make sure to output
    final answer wrapped with <ANS_START> and <ANS_END>.
The final answer should be wrapped in <ANS_START> and <ANS_END> and should
    only contain the final answer, without any reasoning or related
    explanations.

[Answers]:
```

F.3 Generated Knowledge Prompting

```
gen_knowledge_template: |
Generate interpretation about the following instruction: {base_instruction}

Output the interpretation directly without including any other content.
```

F.4 Attribution-based Prompting

```
gen_attribute_template: |
  Extract only the key words or phrases from the following instruction that
      are most likely to influence a language model's output. Do not include
      any explanations or extra content. Just list the key words or phrases.

  [Instruction]: {base_instruction}
```

```
attribute_based_refine_template: |
  Below is the prompt for the task instruction:
  {base_instruction}

  The key words or phrases identified in the task instruction are:
  {attributed_componment}

  Please refine the original task instruction based on these key components.
      The refinement may include (but is not limited to) the following actions
      :
  1.Refine vague terms to be more specific
  2.Reorder key phrases for emphasis
  3.Expand or compress phrases for clarity and focus
  4.Add stylistic or structural constraints
  5.Test synonyms or alternative wording
  6.Eliminate redundant elements

  The output should be an optimized version of the original task instruction.
  Please ensure that the refined task_description is strictly enclosed within
      <START> and <END>.
```