Tree-of-Prompts: Abstracting Control-Flow for Prompt Optimization

Jihyuk Kim[1*], **Shubham Garg**[2], **Lahari Poddar**[2]
Seung-won Hwang[3†], **Christopher Hench**[2]

[1]LG AI Research, jihyuk.kim@lgresearch.ai
[2]Amazon, {gargshu, poddarl, henchc}@amazon.com
[3]Seoul National University, seungwonh@snu.ac.kr

Abstract

Prompt optimization (PO) generates prompts to guide Large Language Models (LLMs) in performing tasks. Existing methods, such as PromptAgent, rely on a single static prompt, which struggles with disjoint cases in complex tasks. Although Mixture-of-Prompts (MoP) uses multiple prompts, it fails to account for variations in task complexity. Inspired by programmatic control flow, we introduce a nested if-else structure to address both varying similarities and complexities across diverse cases. We propose **T**ree-**o**f-**P**rompts (**ToP**), which recursively expands child prompts from a parent prompt. Sibling prompts tackle disjoint cases while inheriting shared similarities from their parent, and handle cases more complex than the parent. Evaluated on Gorilla (*understanding*), MATH (*reasoning*), and a subset of BBH benchmarks, ToP outperforms PromptAgent and MoP, with improvements of 1.4% and 4.6% over PromptAgent and 3.2% and 4.5% over MoP, when tested with GPT-4o-mini and Llama 3.2-3B, respectively.

1 Introduction

Large Language Models (LLMs) (Brown et al., 2020; Liu et al., 2023) are used across a wide variety of tasks and the prompt optimization (PO) (Cheng et al., 2024) methods need to be able to harness the potential of LLMs for each of them. We hypothesize that a task consists of diverse cases, varying in terms of 1) **complexity**, defined as the level of reasoning effort required to solve a case, and 2) **disjointness**, referring to the degree of separation between cases based on their underlying reasoning strategies.

An exemplar is the API function calling task (Patil et al., 2023), where the system is given a question and a list of available functions and generates function call(s) to answer the question. In this task, (1) in terms of complexity, the gold answer may involve a single function invocation

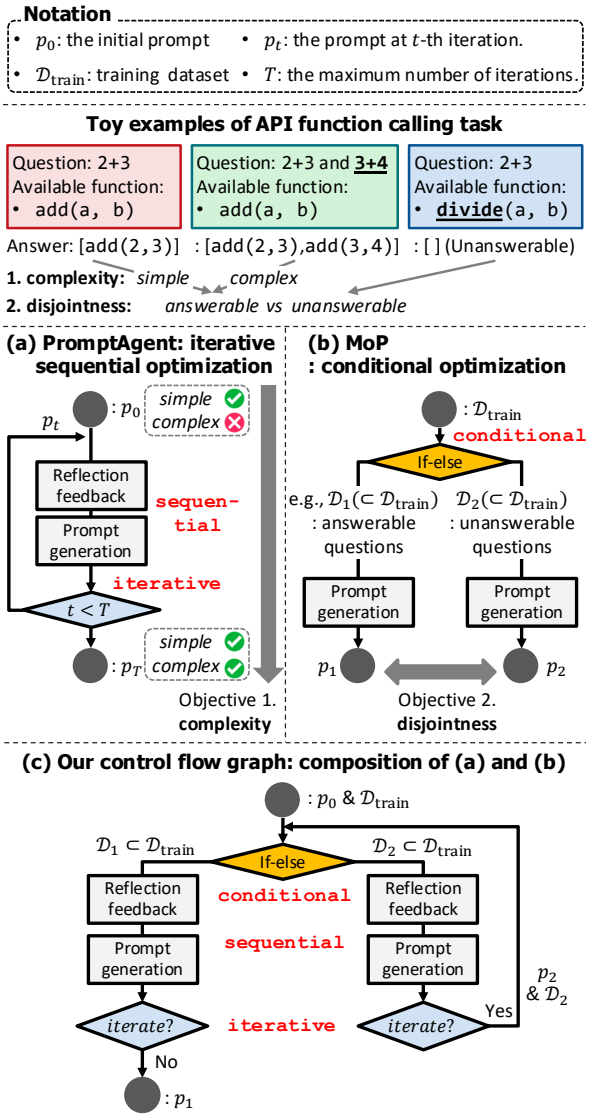

Figure 1: Illustration of the control flow graph for prompt optimization, comparing the two existing approaches – (a) PromptAgent and (b) MoP – with (c) our proposed method. (a) and (b) employ `iterative`, `sequential` execution and `conditional` execution, respectively, while (c) we integrate all the executions.

[*] Work done during internship at Amazon.
[†] Corresponding author.

Findings of the Association for Computational Linguistics: ACL 2025, pages 19436–19459

for a simple question or multiple invocations for a complex multi-step question, e.g., the toy examples presented in Figure 1 with red and green boxes, respectively. (2) In terms of varying similarity, while the above cases share a similarity – both are answerable – some questions may be unanswerable if none of the provided functions are relevant, as shown in the blue box example in Figure 1. Our contribution is identifying critical gaps between existing PO approaches and such real-world scenarios.

Existing works, illustrated in Figure 1(a, b), consider only either complexity or distinct cases. For the complexity, PromptAgent (Wang et al., 2024c) begins with a vanilla prompt, denoted by p_0, capable of handling simple cases, and iteratively refines the prompt over a predefined number of iterations, T, to eventually address more complex ones. For the refinement at each iteration, the reflection technique (Pryzant et al., 2023; Madaan et al., 2023) has been employed to address errors in the previous prompt. On the other hand, to address distinct cases, Mixture-of-Prompts (MoP) (Wang et al., 2024b) separately optimizes multiple prompts by partitioning the set of training examples, denoted by $\mathcal{D}_{\text{train}}$, case by case, e.g., one prompt p_1 for answerable cases and another p_2 for unanswerable ones.

Our novelty lies in systematically incorporating both the complexity and the disjointness for PO. To this end, inspired by recent work of "programming" via LLMs (Khattab et al., 2024; Opsahl-Ong et al., 2024), we formulate PO via a *Control Flow Graph* (CFG) (Allen, 1970) to offer a unified framework, illustrated in Figure 1 (c). Specifically, we integrate the three execution types in CFG: sequential, iterative, and conditional. From the perspective of CFG, PromptAgent implements the sequential execution via reflection feedback, in conjunction with iterative execution, though it often fails to handle disjoint cases. On the other hand, MoP implements conditinoal optimization to address disjoint cases. However, without iterative refinement, each prompt may underfit and remain sub-optimal. Integrating all three execution types, we iteratively generate better prompts conditioned on specific cases so that the set of optimized prompts can comprehensively handle diverse cases.

2 Related Work

We explore PO for tasks with diverse cases, focusing on two objectives: (1) during *training*, optimizing a comprehensive set of prompts to address both complex and disjoint cases, and (2) during *testing*, employing case-specific prompts for given inputs, as detailed below.

2.1 Prompt Optimization during Training

Existing works (Wang et al., 2024c,b) on PO focus on either 1) iteratively optimizing a single prompt or 2) separately optimizing multiple prompts. From CFG perspective, these correspond to iterative sequential execution and conditional execution, respectively. However, both fail to fully address complexity and disjointness, which our method integrates to handle diverse cases effectively.

Iterative Sequential PO Employing iterative sequential execution for PO, PromptAgent (Wang et al., 2024c) and Prompt Optimization with Textual Gradients (ProTeGi) (Pryzant et al., 2023) repeat the sequential process of reflect-then-update. Reflection feedback (Madaan et al., 2023), derived from error examples in the previous prompt, enables the prompt to be updated to address more complex cases. For iteration strategies, while ProTeGi adopts beam search, PromptAgent employs Monte Carlo Tree Search, offering an improved balance between exploitation and exploration. Despite these advancements, relying on a single prompt often biases the model towards majority cases, leading to poorer performance on disjoint minority cases (Henning et al., 2023).

Conditional PO MoP (Wang et al., 2024b), on the other hand, employs conditional execution for PO by optimizing a set of prompts $\mathcal{P}$, each conditioned on examples from a specific case. The training examples in $\mathcal{D}_{\text{train}}$ are first partitioned into multiple clusters, each representing a distinct case, e.g., $\mathcal{D}_1 \subset \mathcal{D}_{\text{train}}$ and $\mathcal{D}_2 \subset \mathcal{D}_{\text{train}}$ consisting of answerable and unanswerable questions, respectively. For each cluster, a prompt is optimized using Automatic Prompt Engineer (APE) (Zhou et al., 2023), which derives task instructions from a few input-output pairs sampled from $\mathcal{D}_{\text{train}}$. In contrast, MoP samples these pairs from each individual cluster. However, individual prompts, without iterative refinements, may fail to address complex cases.

Distinction To combine the strengths of iterative and conditional PO, a straightforward approach is to iteratively refining a prompt conditioned on each cluster. However, it treats all cases as entirely disjoint, independently optimizing prompts with-

out leveraging commonalities across cases. For instance, the toy examples shown with red and green boxes in Figure 1 both fall under the category of answerable questions, yet the naive approach would handle them separately. As a novel and effective integration, our distinction is utilizing a hierarchical tree structure that allows prompts to share a common parent prompt, enabling the reuse of shared knowledge while maintaining flexibility for case-specific optimization for each prompt. We empirically demonstrate its effectiveness compared to the naive integration in our experiments (Table 1).

2.2 Case-specific Prompting during Testing

To achieve adaptive test-time prompting for diverse cases, existing methods can be divided into generation-based and search-based approaches, with our method belonging to the latter.

Generation Approach Input-specific prompts can be generated on the fly during testing. ExpertPrompting (Xu et al., 2023) implements this by generating an input-tailored expert identity as the prompt. To extend it beyond a single prompt, Multi-ExpertPrompting (Long et al., 2024) incorporates multiple expert identities, while HMAW (Liu et al., 2024) introduces a hierarchical multi-agent workflow consisting of CEO, Manager, and Worker prompts. However, since those prompts remain unverified, their effectiveness is not guaranteed.

Search Approach In contrast, the search-based approach, e.g., MoP, employs only verified prompts. During training, it optimizes a comprehensive set of prompts, each validated on training examples, and during testing, it searches for the most suitable prompt. To identify input-specific prompts, MoP employs semantic similarity between training and testing inputs, though it often fails to distinguish between different cases. To illustrate, the two toy examples presented with red and blue boxes in Figure 1 only differ in the presence or absence of the relevant function for the identical question "(What is) 2+3". Despite the similarity, the two belong to disjoint cases, i.e., answerable and unanswerable cases.

Distinction Adopting the search-based approach, our distinction is in enhancing case identification by proposing solution-based clustering during training and case-augmented search during testing.

3 Tree-of-Prompts

We introduce a complete CFG to leverage all three execution types, namely, sequential, iterative and conditional. We iteratively optimize multiple prompts with reflection feedback on previous prompts while identifying disjoint cases at each iteration. Specifically, as an implementation of CFG for PO, we propose Tree-of-Prompts (ToP), where the tree structure represents a roll-out of iterative conditional optimization applied to prompts, as illustrated in Figure 2(a). The algorithm for building the tree is presented in Algorithm 1.

We start with a root node containing the entire training dataset, $\mathcal{D}_{\text{train}}$, and the initial prompt, p_0. The training dataset is defined as $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i and y_i denote the input and its corresponding label, respectively, and N represents the number of training examples. For p_0, we use a basic task description that can be easily crafted by a human, e.g., "Given a question and a list of functions, generate function call(s) to answer the question" for API function calling. The algorithm then invokes $\text{BUILD}(\mathcal{D}_{\text{train}}, p_0)$, to progressively expand the tree via three execution operations in CFG as follows.

Conditional Execution We first partition $\mathcal{D}_{\text{train}}$ into K clusters, denoted by $\{\mathcal{D}_k\}_{k=1}^{K}$, each corresponding to a specific case, creating K new nodes as the root's children. The prompt for each child node is then conditionally optimized based on $\mathcal{D}_k$. Note that we do not assume case labels to be known during training, and learn the distribution through our clustering algorithm as detailed in Section 3.1.

Sequential Execution To optimize a prompt for each child node using $\mathcal{D}_k$, we employ a reflection-based prompt update (Pryzant et al., 2023; Wang et al., 2024c), where two LLMs involve: a base LLM, denoted as M_{base}, and an optimizer LLM, denoted as $M_{\text{optimizer}}$. M_{base} produces the response $\hat{y}$ to x using a prompt p. $M_{\text{optimizer}}$ provides the reflection feedback on p based on a batch of error examples $\mathcal{D}_{\text{batch}}$ ($\subset \mathcal{D}_k$), and suggests a better prompt p_k that can address the errors:

$$\mathcal{D}_{\text{batch}} = \{(x_b, y_b, \hat{y}_b) | \hat{y}_b \neq y_b\}_{b=1}^{B}, \quad (1)$$

$$p_k = M_{\text{optimizer}}(\mathcal{D}_{\text{batch}}, p), \quad (2)$$

where $\hat{y}_b = M_{\text{base}}(x_b; p)$ and B denotes the batch size. To avoid degeneration, we sample the optimized prompt S times and select the best one based on its performance on $\mathcal{D}_{\text{valid}}$, the held-out

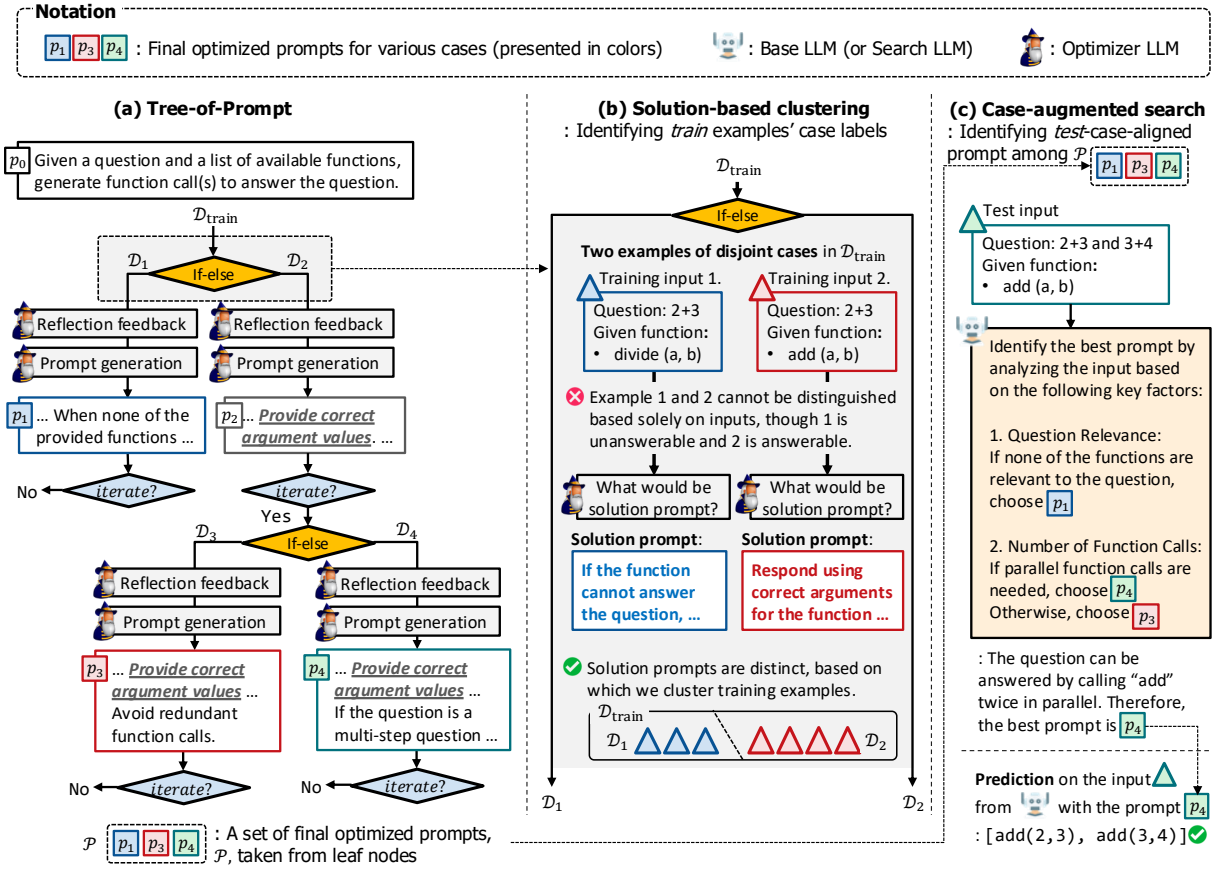

Figure 2: Illustration of our proposed method, Tree-of-Prompts (ToP), implementing CFG using a tree.

validation dataset. We present meta prompts for the reflection feedback and the prompt generation in Appendix A.1 (Figure 6).

Note that our integration of sequential execution with conditional execution allows the parent's prompt, p, to be reused as a preliminary prompt for every child node. This design is motivated by the observation that the specific cases addressed by child nodes are derived from the same broader case targeted by their parent, resulting in closely related cases with shared characteristics. For example, for all answerable cases in API function calling, prompts should include guidance for proper function formatting for generation. Therefore, reusing p enables the child nodes to leverage this shared knowledge (e.g., function formatting) while refining p into p_k to address their specialized cases (e.g., a single function call or parallel function calls).

Iterative Execution We recursively partition the examples in each child node, expanding the tree iteratively. The iteration, i.e., partitioning a node, terminates when the optimized prompt for that node produces fewer than B error examples within the assigned cluster, ensuring sufficient training data for the reflection-based prompt update (Eq (1)).

Algorithm 1 Building the tree of prompts by invoking $\text{BUILD}(\mathcal{D}_{\text{train}}, p_0)$, which produces a set of prompts, denoted by $\mathcal{P}$, with each optimized for a different case.

Function: $\text{BUILD}(\mathcal{D}, p)$
if $\sum_{\forall(x,y)\in\mathcal{D}} \mathbb{I}(y = M_{\text{base}}(x;p)) < B$ **then**
  return $\{p\}$
end if
$\mathcal{P} \leftarrow \emptyset$
$\{\mathcal{D}_1, \ldots, \mathcal{D}_K\} \leftarrow \text{KMEANS}(\mathcal{D}, K)$ ▷ $\bigcup_{k=1}^{K} \mathcal{D}_k = \mathcal{D}$
for $\mathcal{D}_k$ in $\{\mathcal{D}_1, \ldots, \mathcal{D}_K\}$ **do** ▷ *conditional execution*
  $\nabla \leftarrow \text{REFLECT}(\mathcal{D}_k, p)$ ▷ *sequential execution* 1. obtain errors in p on $\mathcal{D}_k$, denoted by ∇
  $p_k \leftarrow \text{UPDATE}(p, \nabla)$ ▷ *sequential execution* 2. update p to avoid the errrors.
  $\mathcal{P} \leftarrow \mathcal{P} \cup \text{BUILD}(\mathcal{D}_k, p_k)$ ▷ *iterative execution*
end for
return $\mathcal{P}$

Additionally, once the tree is built, to avoid redundant prompts, we truncate it to the optimal depth based on the performance on $\mathcal{D}_{\text{valid}}$. In our experiments, the optimal depth typically ranges from 1 to 3. We present an example diagram of the tree built in Appendix C.1.

Once the tree is built, we take prompts in leaf nodes, producing a final prompt set $\mathcal{P}$, with each prompt considered an expert in its target case. At test time, for each input, a test-case-aligned prompt,

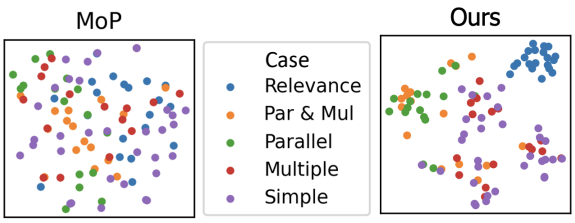

Figure 3: tSNE visualization of embeddings on training examples, comparing MoP (semantic-based embeddings) and Ours (solution-based embeddings).

capable of effectively addressing the input's case, is searched from $\mathcal{P}$, and is used by M_{base} to produce the final response to the input.

A key challenge is that case labels for task examples are often unknown during both training and testing. Our contribution is automatically identifying such labels for training examples and test inputs, using solution-based clustering (§3.1) and case-augmented search prompting (§3.2), respectively, discussed below.

3.1 Solution-based Clustering

To identify case labels for training examples, and enable conditional execution, we propose a novel solution-based clustering approach as illustrated in Figure 2(b). We model the similarity between examples – and ultimately between their respective cases – by the prompts that can effectively handle the examples, referred to as "solution" prompts.

In contrast, MoP clusters examples based on semantic similarity; however, semantically similar examples may belong to different cases. For example, as showcased by "Training input 1" and "Training input 2" in Figure 2(b) with an identical question, two entirely different cases – one answerable and the other unanswerable – may only semantically differ in the presence of a relevant function in the provided function list.

To devise a better alternative, our intuition is that examples that can be addressed using similar solution prompts are likely to belong to the same case, while those requiring different prompts often correspond to distinct cases. For example, solution prompts for unanswerable questions will differ from those for answerable ones, as illustrated by red-colored and blue-colored prompts in Figure 2(b), enabling them to be separated into different clusters. Based on the intuition, for each example $(x, y) \in \mathcal{D}$ where $\mathcal{D}$ denotes the cluster in a given node, we first generate the solution prompt, denoted by $p_{(x,y)}$. This is generated using only a single example in Eq (2) with $\{(x, y)\}$ as a single

Instruction for optimizer LLM:

I have candidate prompts designed for a task.
: ***Prompt 1***. … \n ***Prompt 2***. … \n …

Here are the evaluation results on the prompts:
<1> (***Prompt 1*** answers correctly)
Question: What is 2+3?
Given function: divide(int a, int b)
The correct answer is: The function is not relevant.
<2> (***Prompt 2*** answers correctly)
Question: What is 2+3?
Given function: add(int a, int b)
The correct answer is: add(2, 3)
…
Summarize effective cases for each prompt:

Response:

Prompt 1 is effective when none of the functions can answer the question. … ***Prompt 2*** is suitable for straightforward queries needing single call. …

Instruction for optimizer LLM:

Now, outline key factors necessary for analyzing the input, and present guidance on how to choose the most suitable prompt:

Response:

1. **Question Relevance**:
If none of the functions are relevant to the question, choose **Prompt 1**.
2. **Number of Function Calls**:
If a single call is sufficient, select ***Prompt 2*** …

Figure 4: Meta prompts used by $M_{\text{optimizer}}$ for constructing p_{search} and its example responses. Colors indicate different target cases: blue for unanswerable cases and red for answerable cases[1].

size $\mathcal{D}_{\text{batch}}$. Meta prompts used for the solution prompt generation are presented in Appendix A.2 (Figure 7). We utilize $p_{(x,y)}$ in representing each example, based on which we perform K-means clustering. Specifically, for each example, we encode $p_{(x,y)}$, instead of the input text, into a dense vector, denoted by $\mathbf{e}_{(x,y)}$.

To understand the quality of clusters, we plot a tSNE visualization in Figure 3. The visualization shows that MoP is not able to capture the underlying distributions of subtasks by only relying on semantic similarity and fails to separate any cases of examples. In contrast, our solution-based representation $\mathbf{e}_{(x,y)}$ enables separating distinct examples and form cohesive clusters.

3.2 Case-Augmented Search

Identifying the test-case-aligned prompt from the set $\mathcal{P}$ can be formulated as a search task, where the input given at test time and each prompt in $\mathcal{P}$ serve

as the query and search target, respectively. For the search task during inference, MoP again relies on semantic similarity, where it searches for the prompt having examples that are most semantically similar to the given input. However, as shown in Figure 3, the semantic similarity often fails to differentiate different cases.

To address this gap during inference, we propose Case-Augmented Search, as illustrated in Figure 2(c). Inspired by recent work leveraging LLM for search tasks (Sun et al., 2023; Tang et al., 2024; Zhuang et al., 2024; Reddy et al., 2024; Xu et al., 2024; Zeng et al., 2024), we introduce a dedicated LLM for prompt search, denoted as M_{search}, along with its prompt, denoted as p_{search}. We augment case information in p_{search} by incorporating: (1) the specific case in which each prompt in $\mathcal{P}$ is most effective and (2) guidelines for determining the specific case of the given input. The combination of these two instructions enables identification of a case-aligned prompt for the test input.

To this end, we first sample training examples that highlight the unique strengths and weaknesses of each prompt across different cases. Specifically, we sample B_{search} examples, where, for each example, at least one prompt fails, exposing weaknesses, and at least one prompt succeeds, highlighting strengths. As illustrated in Figure 4, we then present $M_{\text{optimizer}}$ with the evaluation results of $\mathcal{P}$ on the sampled examples, indicating whether each $p \in \mathcal{P}$ is correct or incorrect. We instruct $M_{\text{optimizer}}$ to identify effective cases for each p and explain the key factors that can help identify the specific case of the given input. We repeat sampling p_{search} S times and choose the best p_{search} based on performance on $\mathcal{D}_{\text{valid}}$.

4 Experiments

4.1 Evaluation Setup

Dataset and evaluation metric For evaluation, we employ two representative tasks: API function calling and solving mathematical problems, using **Gorilla** (Patil et al., 2023) and **MATH** (Hendrycks et al., 2021) datasets, respectively. API function calling assesses *understanding and generation* capabilities of LLMs. Given a question and a list of available functions, an LLM should understand single or multiple requests within the question, assess the relevance of each function to the request(s), and generate function calls with accurate arguments. On the other hand, mathematical problems assess *reasoning* abilities of LLMs, with more emphasis on intermediate steps that allow the model to derive the correct answer from the input provided. Math encompasses diverse cases, as reasoning strategies can vary across subjects such as Algebra, Calculus, Probability, and others. In addition, to further validate the generalization of our method, we also studied challenging tasks in the Big-Bench-Hard (BBH) benchmark. More precisely, following PromptAgent (Wang et al., 2024c), we take the subset of the benchmark, such as **Geometric** Shapes, **Epistemic**, **Object** Counting, **Temporal** Sequences, and **Causal** Judge. Detailed descriptions and statistics on Gorilla, MATH, and BBH, are reported in Appendix B.1 (Table 18), Appendix B.2 (Table 19), and Appendix B.3 (Table 20), respectively.

We adopt the official evaluation metric for each benchmark. For Gorilla, we measure accuracy based on Abstract Syntax Tree (AST) matching between the generated function call(s) and annotated gold function call(s). For unanswerable questions, since none of the provided functions answer the question, a response is considered as correct if AST parsing fails (e.g., "The provided functions are non-relevant to the question."). For the other benchmarks, the accuracy is measured based on exact-match between the parsed response and the gold answer. For parsing responses, the official script for each benchmark was used.

Implementation Detail We evaluated various LLMs, while adhering to the following motivation: To ensure quality optimized prompts, we employed sufficiently capable LLMs for $M_{\text{optimizer}}$, while examining various LLMs for M_{base}. For M_{search}, we used the same LLM used for M_{base}. More precisely, for $M_{\text{optimizer}}$, we used GPT-4o, and for M_{base} and M_{search}, we used GPT-4o-mini and Llama 3.2-3B[2] to examine generalization across different architectures and capacities. In Appendix C.2, we also report performance using GPT-4o for M_{base}.

Following PromptAgent (Wang et al., 2024c), we set the temperature parameter to 1.0 for $M_{\text{optimizer}}$ to facilitate the exploration of diverse prompts, while setting it by 0.0 for M_{base} and M_{search}. Meanwhile, for encoding $\mathbf{e}_{(x,y)}$, we used `e5-mistral-7b-instruct` (Wang et al., 2024a),

[1] In p_{search}, we present the optimized prompts (in $\mathcal{P}$) in descending order based on their performance on $\mathcal{D}_{\text{val}}$, which outperformed random ordering in our preliminary study.

[2] https://huggingface.co/meta-llama/Llama-3.2-3B-Instruct

Model	M_{base}	Gorilla	MATH	Geometric	Epistemic	Object	Temporal	Causal	Average
Vanilla	GPT-4o -mini	0.813	0.668	0.465	0.836	0.830	0.906	0.680	0.743
Human		0.872	0.672	0.575	0.824	0.942	0.902	0.680	0.781
APE		0.867	0.675	0.823	0.860	0.922	**0.993**	0.677	0.831
MoP		0.830	0.669	<u>0.910</u>	0.861	0.942	0.984	0.643	0.834
ProTeGi		0.876	0.675	0.837	0.849	0.958	0.988	<u>0.710</u>	0.842
PromptAgent		<u>0.878</u>	0.679	0.890	<u>0.862</u>	<u>0.966</u>	<u>0.990</u>	0.700	<u>0.852</u>
PA + MoP		0.838	<u>0.683</u>	0.905	0.827	0.957	0.973	0.683	0.838
(Ours) ToP		**0.883**	**0.689**	**0.915**	**0.869**	**0.976**	**0.993**	**0.740**	**0.866**
Vanilla	Llama 3.2 -3B	0.286	0.332	0.005	0.220	0.554	0.476	0.290	0.309
Human		0.326	0.356	0.035	0.086	0.412	0.480	0.420	0.302
APE		0.355	0.357	0.197	0.456	0.545	0.697	<u>0.487</u>	0.442
MoP		0.359	0.365	**0.475**	0.451	0.676	0.651	0.423	0.486
ProTeGi		0.388	0.355	0.237	<u>0.554</u>	0.636	0.690	0.483	0.478
PromptAgent		<u>0.413</u>	0.368	0.188	0.551	<u>0.713</u>	0.715	0.443	0.485
PA + MoP		0.361	<u>0.374</u>	<u>0.432</u>	0.504	0.687	<u>0.721</u>	0.430	<u>0.501</u>
(Ours) ToP		**0.470**	**0.388**	0.250	**0.559**	**0.752**	**0.795**	**0.500**	**0.531**

Table 1: Overall performance comparisons across different LLMs and different datasets. The **best** and <u>second-best</u> performances are highlighted for each M_{base}. We used GPT-4o for $M_{\text{optimizer}}$ to ensure quality prompts.

which has shown one of the state-of-the-art performance on clustering tasks in MTEB benchmark (Muennighoff et al., 2023).

For hyperparameters of ToP, we set $S = 10$ and $B = 5$ following PromptAgent, while $B_{\text{search}} = 15$ to ensure sufficient diversity across cases for building p_{search}. For K, we set $K = 2$ by default, since it signifies the if-else branching of conditional execution, and can represent any set of cases when used in conjunction with iterative executions via nested if-else structures. Nevertheless, in practice, K can be set by any arbitrary number. We report performance from different K in Appendix C.3, where K with 2 or 3 often performs best overall.

Baseline We compare our proposed method with state-of-the-art PO methods, such as **APE** (Zhou et al., 2023), **ProTeGi** (Pryzant et al., 2023), **PromptAgent** (Wang et al., 2024c), and **MoP** (Wang et al., 2024b). In addition, we compare a baseline that combines PromptAgent and MoP, denoted by **PA+MoP**, to contrast with our hierarchical tree structure for integrating iterative and conditional PO. Specifically, PA+MoP first partitions examples into clusters, as done in MoP, and then, for each cluster, uses PromptAgent to iteratively refine a prompt conditioned on the cluster.

We also compare against a **Vanilla** prompt (the initial prompt, p_0) and a **Human**-engineered prompt, detailed in Appendix A.3. For the vanilla prompt, which contains only the basic task description, we manually designed the prompt for Gorilla and MATH, and used the short task description provided in each dataset for BBH. For the human-engineered prompt on Gorilla, assuming the possible cases are known, we added instructions into the vanilla prompt, that can address all possible cases. Notably, identifying possible cases may be infeasible or time-consuming, whereas ToP aims to automatically identify these cases. For BBH and MATH, following PromptAgent, we employed the chain-of-thought prompting (Wei et al., 2022) as the human-engineered prompt.

4.2 Results

ToP generalizes well across diverse tasks and different LLMs. Results on overall performance comparisons are reported in Table 1. MoP, which incorporates conditional PO, outperforms APE, which does not, demonstrating the effectiveness of conditional PO. However, PromptAgent with iterative refinement often surpasses MoP, highlighting the superior contribution of iterative PO over conditional PO. Meanwhile, PA+MoP does not consistently outperform PromptAgent, indicating limited benefits from the naive integration of iterative and conditional PO. In contrast, ToP's hierarchical tree achieves top performance across datasets and LLMs, except for Geometric with Llama 3.2-3B. In Appendix C.5, we compare the comprehensiveness of the set of optimized prompts, $\mathcal{P}$ (i.e., the leaf prompts in the tree), to those from MoP and PA+MoP, while decoupling it from the search process under the assumption of an oracle search.

In the subsequent paragraphs, we performed detailed analyses using the stronger model, GPT-4o-mini, and treated the human-engineered prompt as

Model	(a) Disjointness (in Gorilla)		(b) Complexity (in MATH)	
	ANS	Not ANS	Easy	Hard
Human	0.885	0.781	0.820	0.563
MoP	0.843	0.814	0.824	0.555
PromptAgent	0.891	0.784	0.825	0.571
(Ours) ToP	**0.903**	**0.848**	**0.827**	**0.588**

Table 2: Performance comparisons for disjoint cases – answerable questions and unanswerable questions, denoted by "ANS" and "Not ANS", respectively – on the Gorilla dataset, and comparisons across varying difficulty levels on the MATH dataset.

a robust baseline for evaluating automatic PO, with red highlighting performance below the baseline.

ToP better addresses both disjoint cases and complex cases. For in-depth evaluations, we compare models regarding our two primary objectives: handling (a) disjoint cases and (b) complex cases. These evaluations are conducted using Gorilla and MATH, which provide annotations specific to each case type. More precisely, for the evaluation on the disjoint cases, we report accuracy on the two disjoint cases – answerable and unanswerable questions – in Gorilla. For the complex cases, we employ MATH which includes difficulty level annotations for each question, ranging from Level 1 to 5. To balance the number of evaluation examples, Levels 1–3 are categorized as easy cases, while Levels 4–5 are classified as hard cases. For both evaluation, we compare PromptAgent and MoP as two representative PO methods for handling each objective. Results are presented in Table 2, while more detailed case-by-case comparisons are presented in Appendix C.4.

Regarding (a) disjointness, PromptAgent shows minimal improvement on unanswerable questions (0.784) compared to the human-engineered prompt (0.781). We argue that the single static prompt used by PromptAgent is often biased toward the more frequently observed cases, i.e., answerable questions which account for 80.6% of the dataset. On the other hand, MoP, suffering from under-fitting, underperforms the human-engineered prompt baseline on answerable questions. In contrast, ToP performs the best for both disjoint cases.

Regarding (b) complexity, while MoP performs comparably to the human-engineered prompt on easy questions, it underperforms on harder questions. We argue that the absence of iterative refinements causes MoP to suffer from under-fitting on complex questions, as reflected in the competitive performance of PromptAgent. In contrast, ToP performs the best for both easy and hard cases.

Model	BBH Dataset				
	Geo	Epi	Obj	Tem	Cau
Human	0.575	0.824	0.942	0.902	0.680
generation					
EP	0.500	0.828	0.866	0.924	0.680
MEP	0.620	0.640	0.498	0.782	0.670
HMAW	0.595	0.788	0.516	0.966	0.540
search					
MoP	0.910	0.861	0.942	0.984	0.643
(Ours) ToP	**0.915**	0.882	**0.976**	**0.993**	**0.740**
(w/ oracle search)	*(0.930)*	*(0.988)*	*(0.986)*	*(1.000)*	*(0.940)*
ablation					
ToP w/o CAS	0.895	0.876	0.952	0.986	0.690
ToP (vote)	0.910	**0.896**	0.968	**0.993**	0.690

Table 3: Comparing input-specific prompting methods. "w/o CAS" denotes our ablation model on Case-Augmented Search. "ToP (vote)" refers to removing the search and, instead, employing majority voting based on multiple prompts in $\mathcal{P}$. To reduce space, BBH datasets are abbreviated using their first three characters.

ToP with case-augmented search facilitates accurate, case-specific prompting. Using BBH datasets, we evaluate ToP against existing input-specific prompting methods designed to adaptively address diverse cases based on inputs. As baselines, we compare both generation-based and search-based methods, with ToP belonging to the latter category. For the generation-based approach, we compare ExpertPrompting (**EP**) (Xu et al., 2023), Multi-ExpertPrompting (**MEP**) (Long et al., 2024), and **HMAW** (Liu et al., 2024), while employing MoP as the baseline for the search-based approach. In addition, for in-depth analyses of ToP, we include ToP with oracle search, representing the upper-bound performance of ToP, along with two ablation models as follows. First, we report "ToP w/o CAS", which ablates Case-Augmented Search by replacing p_{search} with vanilla search prompt, detailed in Appendix A.4. Second, to showcase the importance of search, we compare "ToP (vote)", which excludes search on $\mathcal{P}$ but instead employs majority voting on predictions generated by multiple prompts in $\mathcal{P}$. Note that this incurs higher inference costs due to repetitive LLM invocations. Results are reported in Table 3.

The generation-based methods often underperform compared to the human-engineered static prompt, underscoring the potential drawbacks of unverified prompts generated at test time. In contrast, search-based methods, which validate prompts on the training dataset, demonstrate superior overall performance. In particular, when an

oracle search is assumed, ToP shows much better performance, suggesting the potential of our PO method. Among practical search methods, ToP outperforms MoP and the two ablation models. The only exception is Epistemic dataset, where ToP (vote) achieves the best performance. This is because voting has an advantage in this dataset, which involves binary classification with only two options ("entailment" or "non-entailment"), making the aggregated responses from multiple prompts more likely to converge on the correct answer.

5 Conclusion

We study prompt optimization for LLMs to cater to a broad distribution of tasks, which may involve disjoint cases or cases with varying complexity. We propose Tree-of-Prompts which implements control flow graphs to address both scenarios. In the absence of annotated labels for possible cases, we propose the solution-based clustering method for training examples and case-augmented search prompting for test inputs to identify the target cases at both training and test time. We empirically demonstrate the effectiveness ToP on Gorilla, MATH, and BBH benchmarks, outperforming strong baselines such as PromptAgent and MoP.

Limitations

While our work demonstrates success in optimizing prompts for accuracy, there are opportunities for further improvement. Future work could extend prompt optimization to address multiple objectives beyond a single task, such as simultaneously improving accuracy and enhancing the safety of outputs.

Acknowledgements

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea Government (MSIT) (RS-2024-00414981) and ITRC (Information Technology Research Center) support program (IITP-2025-2020-0-01789) supervised by IITP (Institute for Information & Communications Technology Planning & Evaluation).

References

Frances E. Allen. 1970. Control flow analysis. In *Proceedings of a Symposium on Compiler Optimization*, page 1–19, New York, NY, USA. Association for Computing Machinery.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

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

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Sophie Henning, William Beluch, Alexander Fraser, and Annemarie Friedrich. 2023. A survey of methods for addressing class imbalance in deep-learning based natural language processing. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 523–540, Dubrovnik, Croatia. Association for Computational Linguistics.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, Heather Miller, et al. 2024. Dspy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*.

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).

Yuchi Liu, Jaskirat Singh, Gaowen Liu, Ali Payani, and Liang Zheng. 2024. Towards hierarchical multi-agent workflows for zero-shot prompt optimization. *arXiv preprint arXiv:2405.20252*.

Do Xuan Long, Duong Ngoc Yen, Anh Tuan Luu, Kenji Kawaguchi, Min-Yen Kan, and Nancy F. Chen. 2024. Multi-expert prompting improves reliability, safety and usefulness of large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 20370–20401,

Miami, Florida, USA. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Niklas Muennighoff, Nouamane Tazi, Loic Magne, and Nils Reimers. 2023. MTEB: Massive text embedding benchmark. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 2014–2037, Dubrovnik, Croatia. Association for Computational Linguistics.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. 2024. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

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

Revanth Gangi Reddy, JaeHyeok Doo, Yifei Xu, Md Arafat Sultan, Deevya Swain, Avirup Sil, and Heng Ji. 2024. First: Faster improved listwise reranking with single token decoding. *arXiv preprint arXiv:2406.15657*.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937, Singapore. Association for Computational Linguistics.

Raphael Tang, Crystina Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. 2024. Found in the middle: Permutation self-consistency improves listwise ranking in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2327–2340, Mexico City, Mexico. Association for Computational Linguistics.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024a. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Ruochen Wang, Sohyun An, Minhao Cheng, Tianyi Zhou, Sung Ju Hwang, and Cho-Jui Hsieh. 2024b. One prompt is not enough: Automated construction of a mixture-of-expert prompts. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 50043–50064. PMLR.

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

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

Benfeng Xu, An Yang, Junyang Lin, Quan Wang, Chang Zhou, Yongdong Zhang, and Zhendong Mao. 2023. Expertprompting: Instructing large language models to be distinguished experts. *arXiv preprint arXiv:2305.14688*.

Chengjin Xu, Muzhi Li, Cehao Yang, Xuhui Jiang, Lumingyuan Tang, Yiyan Qi, and Jian Guo. 2024. Move beyond triples: Contextual knowledge graph representation and reasoning. *arXiv preprint arXiv:2406.11160*.

Yifan Zeng, Ojas Tendolkar, Raymond Baartmans, Qingyun Wu, Huazheng Wang, and Lizhong Chen. 2024. Llm-rankfusion: Mitigating intrinsic inconsistency in llm-based ranking. *arXiv preprint arXiv:2406.00231*.

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*.

Shengyao Zhuang, Honglei Zhuang, Bevan Koopman, and Guido Zuccon. 2024. A setwise approach for effective and highly efficient zero-shot ranking with large language models. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 38–47, New York, NY, USA. Association for Computing Machinery.

A Prompts

A.1 Reflection-based Prompt Update

Meta prompts used by $M_{\text{optimizer}}$ for the reflection-based prompt update are presented in Figure 6.

A.2 Solution Prompt Generation

Meta prompts used by $M_{\text{optimizer}}$ for the solution prompt generation are presented in Figure 7.

A.3 Manual Prompts

The initial and human-engineered prompts for each evaluation dataset are presented in Table 15 and Table 16, respectively.

A.4 Search Prompt Templates

The prompt templates for the vanilla search and our case-augmented search are presented in Table 17.

B Datasets

B.1 Gorilla

Descriptions and statistics on possible cases for Gorilla benchmark are reported in Table 18.

B.2 MATH

The statistics on possible cases for MATH are reported in Table 19.

B.3 BBH

The statistics for datasets in BBH benchmark are presented in Table 20. For detailed illustrations of diverse cases in each BBH dataset, refer to §C.6.

C Additional Analysis

C.1 Example Diagram of Tree

Figure 5 presents (a) clustering results and (b) performance trajectories of optimized prompts across iterations from the root node (i.e., p_0) to each leaf node: The cluster for p_2 contains training examples for the two correlated cases, such as "Relevance" and "Multiple", where the list of available functions is non-relevant to the given question at all or partially relevant, respectively. As a result, the performance trajectory of $p_0 \rightarrow p_2$ shows improvements in the two cases. On the other hand, for the trajectory of $p_0 \rightarrow p_1 \rightarrow p_4$, the cluster for p_1 contains answerable questions (i.e., all cases except "Relevance"), and the subsequent clustering optimizes p_4 more focused on the answers consisting of multiple function calls in parallel, i.e., "Parallel" and "Parallel & Multiple" cases. As a result, the

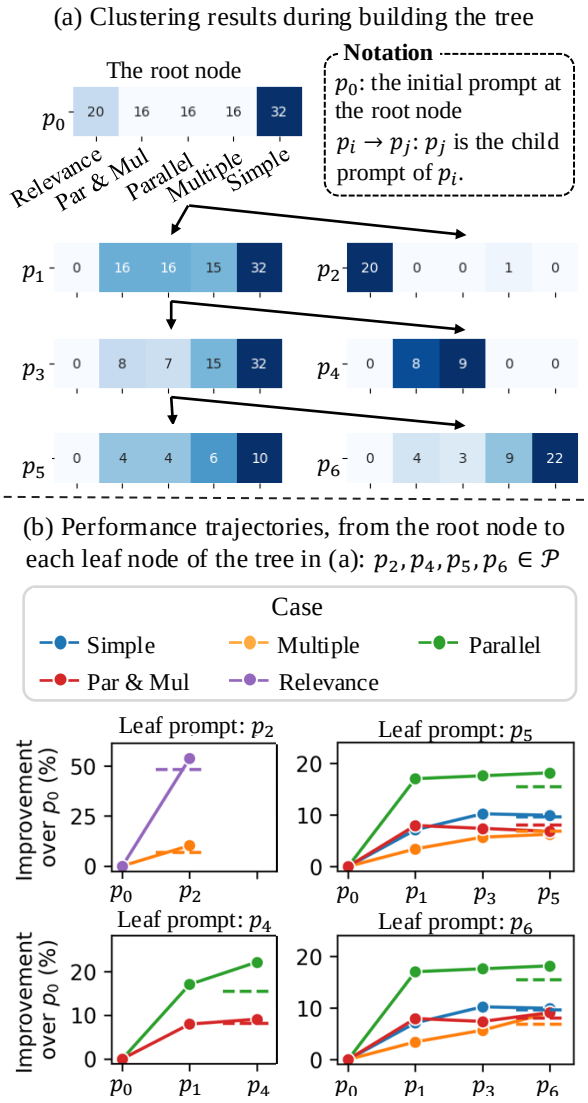

Figure 5: (a) Clustering results and (b) the resulting trajectories of performance from the initial prompt (p_0) to leaf prompts. The numbers in each cell in (a) denote the number of training examples for different cases. In (b), the dashed line denotes PromptAgent's performance using a single prompt iteratively optimized for all cases.

trajectory shows performance improvements in the two cases, such that the optimized prompts p_2 and p_4 complement each other.

C.2 Evaluation of Frontier Model

In Table 4, we report the performance of the frontier model evaluated using GPT-4o on the Gorilla benchmark for both M_{base} and $M_{\text{optimizer}}$. Our proposed method, ToP, achieves the best performance, suggesting that our framework is effective when applied to larger models.

C.3 Performance from Different K

Table 5 reports the performance of different K, such as 2, 3, or 4, where $K = 2$ or $K = 3$ often

Model	M_{base} & $M_{\text{optimizer}}$	Gorilla
Vanilla	GPT-4o	0.376
Human		0.808
APE		0.894
MoP		0.906
ProTeGi		0.910
PromptAgent		0.917
PA+MoP		0.857
(Ours) ToP		**0.925**

Table 4: Performance comparison using GPT-4o for both M_{base} and $M_{\text{optimizer}}$ on Gorilla.

K	Object Counting	Temporal	Causal Judge
2	**0.976**	0.994	**0.740**
3	0.972	**0.998**	0.680
4	0.952	0.946	0.690

Table 5: Performance of different K.

shows the best performance overall.

C.4 Detailed Case-by-Case Comparisons

On Gorilla and MATH which include case labels, we present detailed case-by-case comparisons between all baselines in Table 6 (Gorilla), Table 7 (subjects in MATH), and Table 8 (difficulty levels in MATH).

C.5 Comprehensiveness of $\mathcal{P}$ on Diverse Cases

On diverse cases annotated in Gorilla and MATH, we evaluate the comprehensiveness of the set of optimized prompts, $\mathcal{P}$, generated by our proposed method, ToP, compared to those from MoP and PA+MoP. For this evaluation, we assume an oracle search, where a prediction is considered correct if at least one prompt in $\mathcal{P}$ produces the correct predictions. As a baseline, we also compared PromptAgent, which employs a single static prompt (without search) rather than the set of multiple prompts. Case-by-case comparisons are reported in Table 9 (Gorilla) and Table 10 (MATH).

Among compared models, PromptAgent performs worst, suggesting that the single prompt is limited in addressing diverse cases. Among methods optimizing multiple prompts, MoP performs worst on average accuracy. Due to the lack of iterative refinements, each prompt in $\mathcal{P}$ suffers from under-fitting on its target case. While employing the same clustering method to MoP, PA+MoP produces better $\mathcal{P}$ outperforming MoP by iteratively refining a prompt on each cluster and its target case. However, PA+MoP separately optimizes prompts for different cases, failing to incorporate commonalities between related cases (e.g., answerable cases in Gorilla, such as "Simple", "Multiple", "Parallel", and "Parallel Multiple", or related subjects in MATH, such as "Algebra", "Prealgebra", and "Intermediate Algebra"). For instance, for Llama 3.2-3B on Gorilla (Table 9), PA+MoP shows inconsistent results between related answerable cases: While improving performance on "Parallel Multiple" and "Parallel", PA+MoP significantly degrades performance on "Multiple" and "Simple" (highlighted in red).

In contrast, by introducing a hierarchical tree structure allowing siblings' prompts to share the preliminary prompt from their parent, ToP achieves the best average accuracy on both Gorilla and MATH across all LLMs, and the best accuracy on 9 cases out of 12 for GPT-4o-mini and all cases for Llama 3.2-3B. This validates that ToP produces a comprehensive set of prompts that can effectively address diverse cases.

C.6 Diverse Cases in BBH Datasets

Targeting tasks in BBH benchmark, we present examples of diverse cases and our optimized prompts effective for them, in Table 11, Table 12, Table 13, and Table 14[3].

Geometric Shape (Table 11) This task aims to identify the shape of an object drawn using SVG commands such as "M" (move), "L" (draw a line), or "A" (draw an arc). Effective strategies depend on the patterns of these commands. The example in the first row consists of a single "M" command followed by multiple "L" commands. In such simple cases, an effective strategy can be simply counting the total number of vertices (e.g., five occurrences of "L" commands indicate a pentagon.). In contrast, the second and third examples present more complex and distinct scenarios. Multiple occurrences of "M" in the second example create sub-paths, while the "A" command in the third example defines an arc, resulting in a more intricate shape.

Epistemic (Table 12) The example in the first row is the simplest, as the entailment between the

[3]We excluded "Temporal" from consideration because the vanilla prompt already achieves sufficient accuracy (over 90% with GPT-4o-mini), rendering the task trivial and making a single prompt sufficient to address it.

Model	M_{base}	Diverse cases in API function calling					
		Relevance	Parallel Multiple	Parallel	Multiple	Simple	Average
Vanilla	GPT-4o -mini	0.452	0.807	**0.898**	0.915	0.938	0.802
Human		0.781	0.790	0.886	**0.938**	0.926	0.864
APE		0.794	0.760	0.860	0.918	0.941	0.855
MoP		0.814	0.767	0.812	0.892	0.901	0.837
ProTeGi		0.797	0.792	0.886	0.920	0.939	0.867
PromptAgent		0.784	0.812	0.888	0.932	0.932	0.870
PA + MoP		0.671	0.790	0.881	0.926	0.932	0.840
(Ours) ToP		**0.848**	**0.869**	0.875	0.915	**0.952**	**0.892**
Vanilla	Llama 3.2 -3B	0.336	0.123	0.238	0.452	0.273	0.284
Human		0.329	**0.238**	0.333	0.468	0.290	0.332
MoP		0.425	0.142	0.156	0.495	0.445	0.333
ProTeGi		0.351	0.107	0.270	0.616	0.486	0.366
PromptAgent		0.387	0.202	0.298	0.495	0.545	0.385
PA + MoP		**0.606**	0.153	0.270	0.397	0.336	0.352
(Ours) ToP		0.304	0.145	**0.349**	**0.646**	**0.702**	**0.429**

Table 6: Case-by-case comparisons on Gorilla.

Model	M_{base}	Subjects in MATH							
		Algebra	Counting and Probability	Geometry	Intermediate Algebra	Number Theory	Prealgebra	Precalculus	Average
Vanilla	GPT-4o -mini	0.835	0.606	**0.641**	0.376	0.716	0.841	0.451	0.638
Human		0.818	0.614	0.590	0.436	**0.745**	**0.867**	0.415	0.641
APE		0.844	0.591	0.547	0.466	0.681	0.844	0.421	0.628
MoP		0.861	0.606	0.496	0.444	0.674	0.833	0.451	0.624
ProTeGi		**0.869**	0.598	0.564	0.444	0.660	0.852	0.431	0.631
PromptAgent		0.859	**0.629**	0.556	0.440	0.681	0.859	0.446	0.639
PA + MoP		0.861	0.606	0.573	**0.474**	0.695	0.837	0.456	0.643
(Ours) ToP		0.854	**0.629**	0.607	**0.474**	0.674	0.856	**0.467**	**0.651**
Vanilla	Llama 3.2 -3B	0.482	0.250	0.205	0.141	0.255	0.544	0.138	0.288
Human		**0.547**	0.182	0.205	0.205	**0.319**	0.478	0.200	0.305
MoP		0.530	0.250	0.282	0.162	0.312	0.548	0.174	0.323
ProTeGi		0.523	**0.288**	**0.291**	0.158	0.234	0.541	0.154	0.313
PromptAgent		0.533	**0.288**	0.222	0.188	**0.319**	0.544	0.169	0.323
PA + MoP		0.545	0.250	0.265	0.205	0.305	0.556	0.164	0.327
(Ours) ToP		**0.547**	0.273	**0.291**	**0.209**	**0.319**	**0.559**	**0.215**	**0.345**

Table 7: Case-by-case comparisons on MATH across different subjects.

premise and the hypothesis can be directly identified using the vanilla prompt. In contrast, the second and third examples are more complex for different reasons. For the second example, an effective prompt should address nested beliefs (e.g., "Ava assumes that David thinks that ..."). On the other hand, for the third example, an effective prompt may explore implications or generalizations (e.g., Students who pay attention to a lecture would perform listening).

Object Count (Table 13) The example in the first row corresponds to the simplest case, which can be addressed by general instructions related to the task. The examples in the second and third rows require more complex strategies while being distinct to each other. For the former, as highlighted by blue keywords, an object may appear multiple times, necessitating an instruction related to handling plural forms. On the other hand, for the latter, objects should be first classified to avoid counting non-relevant objects (e.g., "onion" in the example) to the question (e.g., "animals"). The example in the last row is the most challenging, encompassing both complexities. Our optimized set of prompts can comprehensively address those diverse cases with dedicated instructions. Meanwhile, MoP, which relies on semantic similarity, would fail to cluster those examples into distinct cases. Instead, examples are likely to be grouped based on object type (e.g., musical instruments in one cluster and animals in another), as the object type reflects the overall semantics of the question.

Model	M_{base}	Difficulty levels in MATH					
		Level 1	Level 2	Level 3	Level 4	Level 5	Average
Vanilla	GPT-4o -mini	0.930	0.792	0.727	0.656	0.455	0.712
Human		**0.947**	0.792	0.721	0.674	0.452	0.717
APE		**0.947**	**0.812**	0.712	0.643	0.466	0.716
MoP		**0.947**	**0.812**	0.712	0.638	0.471	0.716
ProTeGi		**0.947**	0.799	**0.748**	0.659	0.452	0.721
PromptAgent		**0.947**	0.795	0.733	0.664	0.479	0.724
PA + MoP		0.939	0.799	0.721	0.674	**0.495**	0.725
(Ours) ToP		**0.947**	0.809	0.724	**0.682**	**0.495**	**0.731**
Vanilla	Llama 3.2 -3B	0.658	0.500	0.387	0.279	0.111	0.387
Human		0.711	0.510	0.432	0.326	0.095	0.415
MoP		0.737	0.503	0.429	0.307	0.151	0.426
ProTeGi		0.667	0.514	0.405	0.292	0.161	0.408
PromptAgent		**0.746**	0.497	0.468	0.313	0.124	0.430
PA + MoP		0.667	0.510	**0.474**	0.315	0.153	0.424
(Ours) ToP		0.728	**0.549**	0.447	**0.331**	**0.169**	**0.445**

Table 8: Case-by-case comparisons on MATH across different difficulty levels.

Model	M_{base}	Diverse cases in API function calling					
		Relevance	Parallel Multiple	Parallel	Multiple	Simple	Average
PromptAgent	GPT-4o -mini	0.813	0.839	0.917	0.943	0.952	0.893
MoP		0.859	0.835	0.928	0.966	0.957	0.909
PA + MoP		**0.876**	0.860	0.924	**0.970**	0.957	0.917
(Ours) ToP		0.863	**0.947**	**0.956**	**0.970**	**0.986**	**0.944**
PromptAgent	Llama 3.2 -3B	0.606	0.279	0.429	0.664	0.664	0.528
MoP		0.828	0.339	0.476	0.841	0.793	0.655
PA + MoP		0.906	0.396	0.587	0.765	0.706	0.672
(Ours) ToP		**0.989**	**0.470**	**0.667**	**0.947**	**0.908**	**0.796**

Table 9: Case-by-case comparisons on Gorilla, assuming oracle search on the set of optimized prompts $\mathcal{P}$ from MoP, PA+MoP, and ToP, while contrasting them with the single static prompt optimized by PromptAgent. Red indicates the performance below that of MoP.

Causal Judgement (Table 14) While all three examples require common-sense reasoning to identify causality, they differ in the factors needed to determine it. As disjoint cases, the beliefs or knowledge of the protagonist(s) should be considered for the first and second examples (e.g., "he knows that the water has been poisoned" or "Claire has told Daniel ..."), whereas objective causality in the physical world should be addressed for the third example (e.g., "Someone will win the game if the total of dice roll is greater than 11 and the coin comes up heads"). Regarding complexity, the first example involves direct causality from a single person's action (i.e., "the man pumps the water into the cistern, such that it will poison and kill the inhabitants."), whereas multiple factors may jointly contribute to the outcome in the second and third examples (i.e., "Alex will only win the game if the total of his dice roll is greater than 11 and the coin comes up heads." for the second example, and "If Claire and Daniel are both logged on at the same time, the computer will crash." for the third example), making them more challenging. Therefore, these diverse cases should be addressed using different prompts, as shown by our optimized prompts in the table.

D Potential Risk

As we focus on optimizing prompts primarily for accuracy, the resulting prompts may generate harmful responses. Future work could extend prompt optimization to address multiple objectives beyond a single task, such as simultaneously improving accuracy and minimizing harmful outputs.

E Usage of AI Assistants

We used ChatGPT for language edits.

Model	M_{base}	Subjects in MATH							
		Algebra	Counting and Probability	Geometry	Intermediate Algebra	Number Theory	Prealgebra	Precalculus	Average
PromptAgent	GPT-4o -mini	0.859	0.629	0.556	0.440	0.681	0.859	0.446	0.639
MoP		0.922	0.705	0.726	0.628	**0.844**	0.922	0.564	0.759
PA + MoP		0.932	**0.712**	0.735	0.620	**0.844**	0.930	0.615	0.770
(Ours) ToP		**0.954**	0.689	**0.778**	**0.684**	0.809	**0.952**	**0.631**	**0.785**
PromptAgent	Llama 3.2 -3B	0.533	0.288	0.222	0.188	0.319	0.544	0.169	0.323
MoP		0.788	0.508	0.470	0.359	0.560	0.774	0.385	0.549
PA + MoP		0.796	0.538	0.453	0.380	0.631	0.781	0.369	0.564
(Ours) ToP		**0.849**	**0.636**	**0.607**	**0.483**	**0.645**	**0.867**	**0.415**	**0.643**

Table 10: Case-by-case comparisons on MATH across different subjects, assuming oracle search on the set of optimized prompts $\mathcal{P}$ from MoP, PA+MoP, and ToP, while contrasting them with the single static prompt optimized by PromptAgent. Red indicates the performance below that of MoP.

Task (Geometric Shape): Name geometric shapes from their SVG paths.	
Example	Prompt
This SVG path element <path d="M 62.00, 89.00 L 36.00, 63.00 L 38.00, 28.00 L 85.00, 35.00 L 90.00, 74.00 L 62.00, 89.00"/> draws a Option: (A) circle (B) heptagon (C) hexagon (D) kite (E) line (F) octagon (G) pentagon (H) rectangle (I) sector (J) triangle (**Answer: (G) Pentagon**)	Identify the geometric shapes from their SVG paths based on explicit vertex count and geometric properties. Guidelines: 1. Count the Vertices: Count the number of distinct points (vertices) connected by the lines in the SVG path. ... 2. Shape Identification Rules: ... - Pentagon: 5 vertices. - Hexagon: 6 vertices. ... Confirm the total vertex count matches the shape's properties as per the above rules. ...
This SVG path element <path d="M 27.90, 64.74 L 34.84, 44.47 L 47.96, 46.51 L 42.27, 35.46 L 66.92, 43.08 M 66.92, 43.08 L 55.91, 49.64 M 55.91, 49.64 L 56.62, 66.11 L 27.90, 64.74"/> draws a Option: (A) circle ... (J) triangle (**Answer: (B) Heptagon**)	Identify the geometric shapes ... 1. Count the Vertices: Identify distinct points (vertices) in the SVG path, considering sub-paths (M commands) as part of the single shape unless they form a new shape. ... 3. Disaggregate Sub-Paths: Treat new sub-paths (indicated by 'M') as part of the same shape unless they form distinguishable different shapes. ...
This SVG path element <path d="M 54.00, 61.00 L 68.40, 56.81 A 15.00, 15.00 0.00 0, 1 51.82, 75.84 L 54.00, 61.00"/> draws a Option: (A) circle ... (J) triangle (**Answer: (I) Sector**)	Name geometric shapes from their SVG paths by carefully considering the following details: 1. Arc Commands: Identify if the path uses the "A" command to form arcs. ... If the radii are equal, the shape is a circle; otherwise, it is an ellipse. Determine if the arc forms a complete or partial loop (e.g., circle vs. sector). ...

Table 11: Examples of diverse cases in "Geometric Shape" task and effective prompts for them. Colors highlight keywords consistent between examples and their effective prompts.

Task (Epistemic): Determine whether one sentence entails the next.	
Example	Prompt
Premise: Emma learns that a girl bounces in a bounce house. *Hypothesis*: Ava learns that a girl is bouncing around. (**Answer: non-entailment**)	Determine whether one sentence entails the next.
Premise: Ava assumes that David thinks that two men are drumming, one is standing and one is sitting down. *Hypothesis*: Ava assumes that two men are drumming, one is standing and one is sitting down. (**Answer: non-entailment**)	Determine whether one sentence entails the next by analyzing nested beliefs, understanding that an individual's perception or memory of another's belief implies the truth of that belief. ...
Premise: Isabella sees that a group of attentive students are paying attention to a college lecture. *Hypothesis*: Isabella sees that a group of students are listening. (**Answer: entailment**)	Determine whether one sentence entails the next by analyzing implied contexts, common context implications, and any generalization from specific to broader statements. ...

Table 12: Examples of diverse cases in "Epistemic" task and effective prompts for them. Colors highlight keywords consistent between examples and their effective prompts.

F Artifact Licenses

- **Gorilla dataset**[4]: Apache license 2.0
- **MATH dataset**[5]: MIT license
- **Big-Bench-Hard datasets**[6]: MIT license

[4] https://huggingface.co/datasets/gorilla-llm/Berkeley-Function-Calling-Leaderboard

[5] https://github.com/hendrycks/math

[6] https://github.com/suzgunmirac/BIG-Bench-Hard

Task (Object Count): Count the overall number of all items.	
Example	Prompt
I have an oven, and a toaster. How many objects do I have? (**Answer: 2**)	Count and verify the total number of items by accurately listing each item first, summing them up methodically, and providing only the final total.
I have three pianos, a drum, and four trombones. How many musical instruments do I have? (**Answer: 8**)	... Thoroughly review the entire input list to ensure every item is properly counted, including items in plural forms. ...
I have a mouse, an onion, a pig, a snake, and a yam. How many animals do I have? (**Answer: 3**)	Please count the total number of items belonging to the specified category in the input. Only include items that match the category given in the question. For example, if the question asks, "How many animals do I have?" only count animals.
I have three bears, a fish, a duck, a goat, a donkey, a cat, a snake, a rabbit, a cow, a potato, and a chicken. How many animals do I have? (**Answer: 12**)	Please follow these steps ... 1. **Identify and list all items** of the specified category (e.g., vegetables, fruits, animals). Ensure each item is correctly classified as belonging to the specified category. 2. **Note the quantity of each item**. Remember to: - Properly account for multiple quantities of the same item (e.g., "two chickens" should be noted as 2). - Correctly include or exclude items based on the specified category (e.g., garlic as a vegetable). 3. **Sum all the quantities** to get the total count. Ensure that: - All correctly identified items are included in the final tally. - All quantities are summed accurately, combining multiple counts of the same item. ...

Table 13: Examples of diverse cases in "Object Count" task and effective prompts for them. Colors highlight keywords consistent between examples and their effective prompts.

Task (Causal Judgement): Answer questions about causal attribution.	
Example	Prompt
There is a man who gets paid for pumping water into a cistern thereby replenishing the supply of drinking water in a nearby house. Unfortunately for the inhabitants of the house, the water that the man is pumping into the cistern today has been systematically contaminated with a lethal poison whose effects are unnoticeable until they can no longer be cured. Even though the man pumping the water had nothing to do with poisoning the water, he knows that the water has been poisoned. Nevertheless, the man pumps the water into the cistern knowing that it will poison and kill the inhabitants. But, he doesn't care at all about the inhabitants, he simply wants to do his job and get paid. Did the man intentionally poison the inhabitants? (**Answer: Yes**)	Answer questions about causal attribution by evaluating primary intentions and accepted, foreseeable side effects. Direct and inevitable consequences of an action should always be considered intentional, regardless of the primary intention. Clarify that personal beliefs or procedural constraints do not negate the intentionality of accepted outcomes. Focus on the inevitability and directness of consequences when determining intentionality, ensuring any direct and foreseeable outcome of an intentional action is deemed intentional, even if it was not the primary goal.
Claire's parents bought her an old computer. Claire uses it for schoolwork, but her brother Daniel sometimes logs on to play games. Claire has told Daniel, "Please don't log on to my computer. If we are both logged on at the same time, it will crash". One day, Claire and Daniel logged on to the computer at the same time. The computer crashed. Later that day, Claire's mother is talking with the computer repairman. The repairman says, "I see that Daniel was logged on, but this computer will only crash if two people are logged on at the same time. So, I still don't see quite why the computer crashed." Did Daniel cause the computer crash? (**Answer: Yes**)	Answer questions about causal attribution by balancing the focus between immediate, direct actions and the broader context of events leading to outcomes. Clearly differentiate between actions that are jointly sufficient versus those that independently cause an outcome. Consider defined roles, rules, and violations, emphasizing when agents act within permissions or against guidelines. Acknowledge the chain of events and historical factors that contribute to outcomes, while ensuring secondary agents' contributions are considered ... Emulate typical human reasoning by attributing causality ...
Alex will only win the game if the total of his dice roll is greater than 11 AND the coin comes up heads. It is very unlikely that he will roll higher than 11, but the coin has equal odds of coming up heads or tails. Alex flips the coin and rolls his dice at exactly the same time. The coin comes up heads, and he rolls a 12, so amazingly, he rolled greater than 11. Alex wins the game. Did Alex win because of the coin flip? (**Answer: No**)	Answer questions about causal attribution by systematically analyzing: - Necessary and sufficient causes for each event ... - Interactions and shared causation among different conditions. - Incorporate insights on how typical people attribute causality, ... Break down each scenario clearly, emphasizing common-sense reasoning where responsibility may be shared or multifactorial.

Table 14: Examples of diverse cases in "Causal Judgement" task and effective prompts for them. Colors highlight keywords consistent between examples and their effective prompts.

Reflection Feedback:

I have candidate prompts designed for a task.
I'm writing prompts for a language model designed for a task.

My current prompt is: {`current_prompt`, e.g., the initial prompt}

But this prompt gets the following examples wrong:
<1>
The model's input is: {`model_input`}
The model's response is: {`model_response`}
The correct label is: {`label`}
The model's prediction is: {`prediction`}

<2> …

For each wrong example, carefully examine each question and wrong answer step by step, provide comprehensive and different reasons why the prompt leads to the wrong answer. At last, based on all these reasons, summarize and list all the aspects that can improve the prompt.

Prompt Update:

I'm writing prompts for a language model designed for a task.

My current prompt is: {`current_prompt`, e.g., the initial prompt}

But this prompt gets the following examples wrong:
<1>
The model's input is: {`model_input`}
The model's response is: {`model_response`}
The correct label is: {`label`}
The model's prediction is: {`prediction`}

<2> …

Based on these errors, the problems with this prompt and the reasons are:
{`reflection_feedback`}

There are a list of former prompts including the current prompt, and each prompt is modified from its former prompts:
{`trajectory_of_prompts`}

Based on the above information, please write a new prompt following these guidelines:
1. The new prompt should solve the current prompt's problems.
2. The new prompt should consider the list of prompts and evolve based on the current prompt.
3. The new prompt should be wrapped with <START> and <END>.

The new prompt is:

Figure 6: Meta prompts used by $M_{\text{optimizer}}$ for reflection-based prompt update. The generated response from the first meta prompt is used for "`{reflection_feedback}`" in the second meta prompt. "`{trajectory_of_prompts}`" in the second meta prompt is the list of prompts that have been previously optimized, including "`{current_prompt}`". Both "`{model_response}`" and "`{model_prediction}`" present the model's response to the input. However, the former (the raw response) also includes intermediate responses, typically providing rationales for the final prediction and facilitating reflection feedback.

Solution Prompt Generation for Wrong Example:

I'm writing instructions for a language model designed for a task.

My current instruction is:
{current_prompt}

But this instruction gets the following example wrong:
The model's input is: {model_input}
The model's response is: {model_response}
The correct label is: {label}
The model's prediction is: {prediction}

There are a list of former instructions including the current instruction, and each instruction is modified from its former instructions:
{trajectory_of_prompts}

Based on the above information, please suggest a missing instruction, which will be added to the current instruction, following these guidelines:
1. The missing instruction should address 1 reason why the current instruction could have gotten the example wrong.
2. The missing instruction should consider the list of former instructions and complement them.
3. The missing instruction should not assume any scenarios beyond the provided example.
4. The missing instruction should be entirely abstract and generalized without providing any specific examples including the current example.
5. The missing instruction should be wrapped with <START> and <END>.

The missing instruction is:

Solution Prompt Generation for Correct Example:

I'm writing instructions for a language model designed for a task.

My current instruction is:
{current_prompt}

This instruction gets the following example correct:
The model's input is: {model_input}
The model's response is: {model_response}
The correct label is: {label}
The model's prediction is: {prediction}

There are a list of former instructions including the current instruction, and each instruction is modified from its former instructions:
{trajectory_of_prompts}

Based on the above information, please identify a missing instruction, which will be added to the current instruction, following these guidelines:
1. The missing instruction should consider 1 aspect that has not been explicitly covered in the current instruction but could have helped the model get the example correct.
(Omitting the rest as the same guidelines with the meta prompt above are used)

Figure 7: Meta prompts used by $M_{\text{optimizer}}$ for solution prompt generation. The first and the second meta prompts target scenarios where M_{base} with the current prompt produces a wrong or correct prediction for the example, respectively. "{trajectory_of_prompts}" is the list of prompts that have been previously optimized, including "{current_prompt}". Both "{model_response}" and "{model_prediction}" present the model's response to the input. However, the former (the raw response) also includes intermediate responses, typically providing rationales for the final prediction.

Dataset	Initial prompt
Gorilla	You are given a question and a set of possible functions in Python language. Generate function call(s) to answer the question.
MATH	Answer the mathematical question.
Geometry	Name geometric shapes from their SVG paths.
Epistemic	Determine whether one sentence entails the next.
Object Count	Count the overall number of all items.
Temporal	Answer questions about which times certain events could have occurred.
Causal Judge	Answer questions about causal attribution.

Table 15: Initial prompts, containing basic task descriptions, for different datasets.

Dataset	Human-engineered prompt
Gorilla	You are given a question and a set of possible functions in Python language. Generate function call(s) to answer the question. - If the question can be answered with a single function call, avoid generating additional functions. - If the question requires multiple functions in parallel, separate them with a comma and present them in the order they are called. - Some of the provided functions may not be relevant to the given question. Use only the relevant function(s). - If none of the provided functions can address the question, state that "None of the functions are relevant".
MATH	Please reason step by step.
Geometry	Name geometric shapes from their SVG paths. Let's think step by step.
Epistemic	Determine whether one sentence entails the next. Let's think step by step.
Object Count	Count the overall number of all items. Let's think step by step.
Temporal	Answer questions about which times certain events could have occurred. Let's think step by step.
Causal Judge	Answer questions about causal attribution. Let's think step by step.

Table 16: Human-engineered prompts for different datasets.

Vanilla Search Prompt	You are given multiple candidate prompts used for a language model for the same task with different focuses. Your goal is to choose the best prompt that most fits an input. ### Candidate Prompts: {prompts} ### Input: {input_text} Output only the best prompt's identifier, such as "Prompt 1", without any additional explanations. Output:
Case-Augmented Prompt	You are given multiple candidate prompts used for a language model for the same task with different focuses. Your goal is to choose the best prompt that most fits an input. ### Candidate Prompts: {prompts} ### Input: {input_text} ### Instruction: {case_related_instruction} Output only the best prompt's identifier, such as "Prompt 1", without any additional explanations. Output:

Table 17: Search prompt templates. The example for the case_related_instruction is presented in Figure 4.

Case	Description	$\lvert\mathcal{D}_{\text{train}}\rvert$	$\lvert\mathcal{D}_{\text{valid}}\rvert$	$\lvert\mathcal{D}_{\text{test}}\rvert$
Overall	Given a question and a list of available functions, generate function call(s) to answer the question using the provided functions.	100	50	1090
Simple	The provided list of functions contains only one function, which is relevant to the given question. The question can be answered using a single invocation of the function.	32	16	352
Multiple	The provided list of functions contains multiple functions. Some of them are non-relevant to the given question. The question can be answered using a single invocation of the function.	16	8	176
Parallel	The provided list of functions contains only one function, which is relevant to the given question. The question can be answered using multiple invocations of the function in parallel.	16	8	176
Parallel Multiple	The provided list of functions contains multiple functions. Some of them are non-relevant to the given question. The question can be answered using multiple invocations of the function in parallel.	16	8	176
Relevance	The question is unanswerable. None of the provided functions can answer the question.	20	10	210

Table 18: Descriptions for possible cases in Gorilla benchmark and the number of examples in each split for each case.

Case (Subject)	$\lvert\mathcal{D}_{\text{train}}\rvert$	$\lvert\mathcal{D}_{\text{valid}}\rvert$	$\lvert\mathcal{D}_{\text{test}}\rvert$
Algebra	111	111	137
Counting and Probability	38	38	44
Geometry	53	53	39
Intermediate Algebra	89	89	78
Number Theory	67	67	47
Prealgebra	91	91	90
Precalculus	51	51	65
Total	500	500	500
Complexity	$\lvert\mathcal{D}_{\text{train}}\rvert$	$\lvert\mathcal{D}_{\text{valid}}\rvert$	$\lvert\mathcal{D}_{\text{test}}\rvert$
Level 1	37	37	38
Level 2	85	85	96
Level 3	114	114	111
Level 4	141	141	129
Level 5	123	123	126
Total	500	500	500

Table 19: The number of examples in each split for possible cases and annotated difficulty levels in MATH.

Dataset	$\lvert\mathcal{D}_{\text{train}}\rvert$	$\lvert\mathcal{D}_{\text{valid}}\rvert$	$\lvert\mathcal{D}_{\text{test}}\rvert$
Geometry	150	150	200
Epistemic	500	150	500
Object Count	300	150	500
Temporal	300	150	500
Causal Judge	90	90	100

Table 20: The number of examples for each split for each dataset in BBH benchmark. For detailed illustrations of BBH tasks on diverse cases, refer to §C.6.