

Meta Prompting for AI Systems

Yifan Zhang¹ Yang Yuan^{1,2,3†} Andrew Chi-Chih Yao^{1,2,3†}

¹IIS, Tsinghua University

²Shanghai Artificial Intelligence Laboratory

³Shanghai Qizhi Institute

zhangyif21@mails.tsinghua.edu.cn,

{yuanyang, andrewcyao}@tsinghua.edu.cn

Abstract

In this work, we present a comprehensive study of Meta Prompting (MP), an innovative technique reshaping the utilization of language models (LMs) and AI systems in problem-solving and data interaction. Grounded in type theory and category theory, Meta Prompting emphasizes the structure and syntax of information over traditional content-centric methods. The paper explores the formal definitions of Meta Prompting, sets it apart from few-shot prompting, and underlines its effectiveness in various AI applications. A key focus is applying Meta Prompting for complex reasoning tasks, showing how it effectively deconstructs intricate problems into simpler sub-problems, enhancing token efficiency, and enabling more equitable problem-solving comparisons, especially against few-shot prompting methods. Additionally, the paper introduces Meta Prompting for prompting tasks, allowing LLMs to self-generate new prompts in a recursive, metaprogramming-like manner. Empirical experiments, including using a Qwen-72B base language model equipped with meta prompt without instruction-tuning to solve MATH problems with accuracy at 46.3%, which surpass the supervised fine-tuned counterpart trained with extensive mathematical QA instruction pairs and even the initial version of GPT-4, solving GSM8K problems with 83.5% accuracy with zero-shot meta-prompted Qwen-72B base language model, and solving the Game of 24 tasks with a 100% success rate using GPT-4, demonstrate the meta prompting’s efficacy in achieving high accuracy and efficiency, showcasing Meta Prompting’s transformative impact on AI problem-solving[‡].

1 Introduction

The emergence of foundation models, especially Large Language Models (LLMs), has revolutionized the field of artificial intelligence. These models, exemplified by their extensive training data and capacity for generalization, have dramatically expanded the horizons of computational linguistics, text understanding, and text generation [5, 10, 34–37]. However, a critical challenge faced by LLMs is their limited efficacy in executing complex reasoning tasks, particularly in areas requiring deep, abstract thought such as advanced mathematics [25]. This limitation points towards a need for enhanced methodologies that can augment LLMs’ reasoning faculties.

The root of this challenge lies in the architecture of modern LLMs, which is predominantly oriented toward auto-regressive token prediction [5, 35, 36]. While efficient for a broad spectrum of tasks, this approach is

[†]Corresponding authors.

[‡]The code is available at <https://github.com/meta-prompting/meta-prompting>.

not meticulously designed to support the depth and sophistication of human-like analytical thinking. This discrepancy is highlighted by the dual-process theory of cognitive psychology, articulated by Kahneman [21], which differentiates the fast, intuitive responses of System 1 thinking from the slower, more deliberate reasoning of System 2 thinking. LLMs, in their typical operations, mirror System 1 processes and thus encounter difficulties with tasks that require the more deliberate, structured approach characteristic of System 2 thinking.

Attempts to bridge this gap have led to the development of innovative methodologies such as Chain-of-Thought (CoT) [44] and Tree-of-Thought (ToT) [28, 49], which guide LLMs in articulating intermediate steps in reasoning tasks. These methods, although valuable, have not fully realized the depth and flexibility of human cognitive processes in an abstract sense.

In response to these challenges, we introduce Meta Prompting (MP) and establish a theoretical framework for it, a novel approach that represents a substantial advance in the field of LLM reasoning. Meta Prompting extends beyond existing methods by abstracting and generalizing key principles for enhanced cognitive processing. Unlike its predecessors, Meta Prompting shifts the focus from content-driven reasoning to a more structure-oriented perspective. This method draws inspiration from category theory and type theory, establishing a functorial relationship between tasks and their corresponding prompts. This categorical approach allows for a more systematic and adaptable framework, capable of addressing a wide range of cognitive tasks with depth and nuance akin to human reasoning.

Furthermore, a pivotal aspect of meta prompting is its application to Meta Prompting for prompting tasks in an in-context and recursive way utilizing the functorial and compositional properties of Meta Prompting, which we call Recursive Meta Prompting (RMP). This concept, akin to metaprogramming in programming language theory, involves using LLMs to design new prompts autonomously. The functorial nature of Meta Prompting allows for this advanced capability, where LLMs can not only solve problems but also generate the structures to solve them. This self-referential and recursive ability marks a significant leap in LLMs’ autonomy and adaptability.

The practical efficacy of the Meta Prompting framework is empirically validated through a series of experiments, ranging from solving the Game of 24 puzzles [49] to addressing complex MATH problems [17], underscoring the Meta Prompting’s versatility and empowering LLMs with advanced reasoning capabilities.

In summary, our contributions can be listed as follows:

- We propose the structured and syntax-oriented Meta Prompting (MP), and introduce a theoretical framework for meta prompting based on category theory. We further investigate meta prompting for prompting tasks and Recursive Meta Prompting (RMP) in a metaprogramming-like manner.
- Our experiments on solving MATH problems with a Qwen-72B base language model [3] equipped with meta prompt without instruction-tuning to solve MATH problems with accuracy at 46.3% which surpasses the supervised fine-tuned counterpart trained with extensive mathematical QA instruction pairs and even the initial version of GPT-4, solving GSM8K problems with 83.5% accuracy with zero-shot meta-prompted Qwen-72B base language model, and solving the Game of 24 tasks with 100% success rate using GPT-4, show the efficacy of meta prompting in problem-solving and in-context alignment.

2 Preliminaries

2.1 Category

Definition 2.1 (Category). A *category* \mathcal{C} comprises a collection of *objects* and, for each pair of objects $A, B \in \mathcal{C}$, a set of *morphisms* (or arrows) from A to B , denoted as $\text{Hom}(A, B)$. Morphisms can be intuitively

understood as directed connections or mappings between objects. Notably, in a locally small category, morphisms between any two objects form a set, rather than a class.

Definition 2.2 (Morphisms). For objects A, B in a category \mathcal{C} , a morphism f from A to B is denoted by $f : A \rightarrow B$, where A is the source, and B is the target. It is assumed that $\text{Hom}(A, B)$ is disjoint from $\text{Hom}(A', B')$ unless $A = A'$ and $B = B'$.

Definition 2.3 (Composition of Morphisms). Morphisms in a category are composed in an associative manner. Specifically, if $f \in \text{Hom}(A, B)$ and $g \in \text{Hom}(B, C)$, their composition is a morphism $g \circ f \in \text{Hom}(A, C)$. This composition obeys the associative law: given $f \in \text{Hom}(A, B)$, $g \in \text{Hom}(B, C)$, and $h \in \text{Hom}(C, D)$, it holds that $h \circ (g \circ f) = (h \circ g) \circ f$.

Definition 2.4 (Identity Morphisms). Each object A in a category \mathcal{C} possesses an *identity morphism* $\text{id}_A : A \rightarrow A$. This morphism, when composed with any other morphism $f : A \rightarrow B$ or $g : B \rightarrow A$, yields the original morphism: $f \circ \text{id}_A = f$ and $\text{id}_B \circ g = g$. Furthermore, identity morphisms are unique to each object.

2.2 Functors

Definition 2.5 (Covariant Functor). A *covariant functor* F from a category \mathcal{A} to a category \mathcal{B} , denoted $F : \mathcal{A} \rightarrow \mathcal{B}$, consists of two key components:

- A mapping of objects: $F : \text{obj}(\mathcal{A}) \rightarrow \text{obj}(\mathcal{B})$.
- For each pair of objects $A_1, A_2 \in \mathcal{A}$ and a morphism $m : A_1 \rightarrow A_2$, a corresponding morphism $F(m) : F(A_1) \rightarrow F(A_2)$ in \mathcal{B} .

This functor respects both identity morphisms ($F(\text{id}_A) = \text{id}_{F(A)}$) and composition ($F(m_2 \circ m_1) = F(m_2) \circ F(m_1)$).

Definition 2.6 (Contravariant Functor). A *contravariant functor* is similar to a covariant functor, but it reverses the direction of the morphisms: for $m : A_1 \rightarrow A_2$, the functor maps it to a morphism from $F(A_2)$ to $F(A_1)$. Formally, $F(m_2 \circ m_1) = F(m_1) \circ F(m_2)$.

2.3 Type Theory

Type theory, in the contexts of mathematics, logic, and computer science, serves as a formal presentation of specific type systems and the academic research of these systems. It has been proposed as an alternative to set theory for the foundation of mathematics. Early examples include Alonzo Church's typed λ -calculus and Per Martin-Löf's intuitionistic type theory. These type theories form the basis of many computerized proof-writing systems, such as Thierry Coquand's Calculus of Inductive Constructions, used in proof assistants like Coq and Lean.

In type theory, every term is associated with a type, often expressed as "term : type". Common types include natural numbers (notated as \mathbb{N} or 'nat') and Boolean logic values ('bool'). Terms can be built out of other terms using function application. The computation in type theory is mechanical, achieved by rewriting the term's syntax, and is central to its conceptual framework.

Lambda calculus, integral to type theory, encompasses lambda terms where a term looks like " $\lambda \text{variableName} : \text{type1} . \text{term}$ " and has the type " $\text{type1} \rightarrow \text{type2}$ ". This indicates a function that takes a parameter of type 'type1' and computes to a term of type 'type2'.

Type theory diverges from set theory in several key ways:

- Classical set theory adheres to the law of excluded middle (every theorem is either true or false), whereas type theory, leading to intuitionistic logic, does not necessarily subscribe to this law.

- In set theory, an element can appear in multiple sets, but in type theory, terms generally belong to only one type.
- Type theory has a built-in notion of computation, where terms like "1+1" and "2" are different but compute to the same value.
- Type theory encodes numbers more naturally as inductive types, aligning closely with Peano's axioms, as opposed to set theory's encoding of numbers as sets.

3 Meta Prompting

Problem Statement:

- **Problem:** [question to be answered]

Solution Structure:

1. Begin the response with "Let's think step by step."
2. Follow with the reasoning steps, ensuring the solution process is broken down clearly and logically.
3. End the solution with the final answer encapsulated in a LaTeX-formatted box, $\boxed{\dots}$, for clarity and emphasis.
4. Finally, state "The answer is [final answer to the problem].", with the final answer presented in LaTeX notation.

Figure 1: A structure meta prompt presented in markdown format for solving MATH [17] problems.

Problem: Find the domain of the expression $\frac{\sqrt{x-2}}{\sqrt{5-x}}$.

Solution: The expressions inside each square root must be non-negative. Therefore, $x - 2 \geq 0$, so $x \geq 2$, and $5 - x \geq 0$, so $x \leq 5$. Also, the denominator cannot be equal to zero, so $5 - x > 0$, which gives $x < 5$. Therefore, the domain of the expression is $\boxed{[2, 5)}$. Final Answer: The final answer is $[2, 5)$. I hope it is correct.

Problem: If $\det \mathbf{A} = 2$ and $\det \mathbf{B} = 12$, then find $\det(\mathbf{AB})$.

Solution: We have that $\det(\mathbf{AB}) = (\det \mathbf{A})(\det \mathbf{B}) = (2)(12) = \boxed{24}$. Final Answer: The final answer is 24. I hope it is correct.

...

Figure 2: An example of the most widely used few-shot prompt for solving MATH [17] problems, as introduced in the Minerva study by [23].

```

Integrate step-by-step reasoning to solve mathematical problems under following structure:
{
  "Problem": "[question to be answered]",
  "Solution": {
    "Step 1": "Begin the response with ‘Let’s think step by step.’",
    "Step 2": "Follow with the reasoning steps, ensuring the solution process is broken
down clearly and logically.",
    "Step 3": "End the solution with the final answer encapsulated in a LaTeX-formatted
box,  $\boxed{\dots}$ , for clarity and emphasis."
  },
  "Final Answer": "[final answer to the problem]"
}

```

Figure 3: A structure meta prompt presented in JSON format.

Meta Prompting is an advanced prompting technique that focuses on the structural and syntactical aspects of problems, prioritizing the general format and pattern over specific content details. It aims to construct a more abstract, structured approach to interacting with large language models (LLMs), emphasizing the structure and syntax of information. This technique is particularly effective in contexts where the underlying pattern or framework of a problem is crucial for understanding or solving it.

Definition 3.1 (Meta Prompt). A Meta Prompt is an example-agnostic structured prompt designed to capture the reasoning structure of a specific category of tasks. It provides a scaffold that outlines the general approach to a problem, enabling LLMs to fill in specific details as needed. This approach allows for more efficient and targeted use of LLM capabilities by focusing on the "how" of problem-solving rather than the "what".

3.1 Formalizing Meta Prompting

In category theory, a functor F from category \mathcal{C} to \mathcal{D} , denoted $F : \mathcal{C} \rightarrow \mathcal{D}$, maps objects and morphisms (arrows) from \mathcal{C} to \mathcal{D} in a way that preserves the categorical structure (i.e., identity morphisms and composition of morphisms). For a brief introduction to type theory and category theory, please refer to Appendix 2).

Applying this to Meta Prompting, we define two categories: \mathcal{T} , the category of tasks, where objects are tasks and morphisms are logical or functional transformations between tasks; and \mathcal{P} , the category of structured prompts, where objects are prompts designed for tasks in \mathcal{T} , and morphisms are transformations between these prompts that maintain their logical structure and intended purpose (for a detailed formal framework on Meta Prompting, please refer to Appendix A).

The Meta Prompting functor $M : \mathcal{T} \rightarrow \mathcal{P}$ maps each task in \mathcal{T} to a structured prompt in \mathcal{P} that outlines the approach for addressing the task (this mapping can be hand-crafted by human or using language models to self-composing meta prompts in a recursive manner, see Section 4.1). Morphisms in \mathcal{T} (transformations between tasks) are mapped to morphisms in \mathcal{P} (transformations between prompts) in a way that the structure and logic of problem-solving are preserved.

Example of Meta Prompting. Consider the task of solving a quadratic equation, which in \mathcal{T} could be represented as an object Q . The Meta Prompting functor M maps Q to a structured prompt P in \mathcal{P} (see Figure 4), which might outline steps like identifying coefficients, computing the discriminant, and using

the quadratic formula. This prompt P guides the LLM in solving any quadratic equation by following the structured approach, regardless of the specific coefficients involved.

```
A structure meta prompt for solving quadratic equations in the form  $ax^2 + bx + c = 0$ :
{
  "Problem": "Solve the quadratic equation  $ax^2 + bx + c = 0$  for  $x$ .",
  "Solution": {
    "Step 1": "Identify the coefficients  $a$ ,  $b$ , and  $c$  from the equation.",
    "Step 2": "Compute the discriminant using  $\Delta = b^2 - 4ac$ .",
    "Step 3": "Check if  $\Delta > 0$ ,  $\Delta = 0$ , or  $\Delta < 0$  to determine the nature of the roots.",
    "Step 4": "If  $\Delta > 0$ , calculate the two distinct real roots using  $x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$ .",
    "Step 5": "If  $\Delta = 0$ , calculate the single real root using  $x = \frac{-b}{2a}$ .",
    "Step 6": "If  $\Delta < 0$ , calculate the complex roots using  $x_{1,2} = \frac{-b \pm i\sqrt{|\Delta|}}{2a}$ .",
    "Step 7": "End the solution process by summarizing the roots of the equation."
  },
  "Final Answer": "Depending on the discriminant  $\Delta$ , the final answer will be the roots of the equation, given by  $x_{1,2}$ ."
}
```

Figure 4: A structure meta prompt in JSON format for solving quadratic equations, demonstrating the structure of step-by-step reasoning process.

```
Example 1:
Problem: Solve the quadratic equation  $3x^2 + 4x - 5 = 0$ .
Solution:
  • Step 1: Identify the coefficients:  $a = 3$ ,  $b = 4$ , and  $c = -5$ .
  • Step 2: Calculate the discriminant:  $\Delta = b^2 - 4ac = 4^2 - 4(3)(-5) = 16 + 60 = 76$ .
  • Step 3: Since  $\Delta > 0$ , the equation has two distinct real roots.
  • Step 4: Calculate the roots using the quadratic formula:  $x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a} = \frac{-4 \pm \sqrt{76}}{6}$ .
  • Step 5: Simplify to find the roots:  $x_1 = \frac{-4 + \sqrt{76}}{6}$  and  $x_2 = \frac{-4 - \sqrt{76}}{6}$ .
Final Answer: The roots of the equation  $3x^2 + 4x - 5 = 0$  are  $x_1 = \frac{-4 + \sqrt{76}}{6}$  and  $x_2 = \frac{-4 - \sqrt{76}}{6}$ .

Example 2: ...
```

Figure 5: An example of a few-shot prompt for solving quadratic equations, using specific examples with detailed steps and solutions.

Characteristics of Meta Prompting. Meta Prompting prioritizes the form and structure over content, using syntax as a guiding template for the expected response or solution. It employs abstracted examples as frameworks, illustrating the structure of problems and solutions without focusing on specific details. Drawing from type theory, Meta Prompting emphasizes the categorization of components in a prompt, such as problem statements, solution steps, and conclusions, and focuses on their logical arrangement and interrelationships to ensure a coherent and structured approach to problem-solving. This technique is versatile and applicable across various domains, capable of providing structured responses to a wide range of problems. By concentrating on structural patterns, Meta Prompting offers a clear roadmap for problem-solving, aiding in the navigation of complex topics.

3.2 Distinction between Meta Prompting and Few-Shot Prompting

Meta Prompting differs from Few-Shot Prompting in its approach and objectives. Few-Shot Prompting involves providing LLMs with a small number of example problems and their solutions to learn from. In contrast, Meta Prompting abstracts the problem-solving process itself, creating prompts that guide the model through the logical steps required to solve a problem without relying on specific content-based examples.

To further clarify the unique roles and methodologies of Meta Prompting and Few-Shot Prompting (see Figure 1 and Figure 2 for illustrations, and Figure 3 for a structure meta prompt presented in JSON format), let us explore their differences in more detail (please refer to Appendix A.1 for more discussions):

- *Meta Prompting* involves two distinct categories (for a brief introduction to category theory, please refer to Appendix 2), \mathcal{T} for tasks (problems) and \mathcal{P} for structured prompts. The functor $\mathcal{M} : \mathcal{T} \rightarrow \mathcal{P}$ defines the relationship between problems and their corresponding prompts. For instance, a complex reasoning task T in \mathcal{T} is associated with a structured, step-by-step prompt P in \mathcal{P} that guides the user through the problem-solving process.

$$P = \mathcal{M}(T). \quad (1)$$

When $\mathcal{M}(\cdot)$ is task-agnostic, it's denoted as 'Meta Prompt' as a constant. This term reflects a general, adaptable prompt structure that's not specific to any particular task but is designed to be versatile across various tasks such as the system (generic) meta prompt shown in Figure 7 (in Appendix C). When the Meta Prompt is specialized for a specific category of tasks (a subcategory within a broader task domain or a particular universe of tasks), it tailors the general prompt structure to the unique requirements and characteristics of that task category, such as the meta prompt designed for solving MATH problems [17] shown in Figure 1. This specialization ensures that the prompt remains relevant and effective within the specific context it's designed for, thereby enhancing its utility and effectiveness in guiding solutions or responses within that domain.

The language model's function, represented as $\text{LLM}(\mathcal{M}(\mathcal{T}_{\text{unsolved}}))$, effectively bridges the gap between an unsolved task T_{unsolved} and its solution process. This function first translates T_{unsolved} into a structured prompt P_{unsolved} within the category \mathcal{P} . It then processes P_{unsolved} to yield P_{solved} , the structured solution. Notice that LLM can be replaced by more powerful multi-modal foundation models (see Typed Meta Prompting for multi-modal foundation models in Appendix E) or AI systems equipped with external computational and physical environments.

$$\begin{array}{ccc}
 T_{\text{unsolved}} & \xrightarrow{\text{Solve the Task in Ideal}} & T_{\text{solved}} \\
 \mathcal{M} \downarrow & & \downarrow \mathcal{M} \\
 P_{\text{unsolved}} & \xrightarrow{\text{LLM}(\mathcal{M}(T_{\text{unsolved}}))} & \boxed{P_{\text{solved}}}
 \end{array} \quad (2)$$

Notably, when \mathcal{M} is task-agnostic, this transformation process exemplifies currying, a concept in functional programming where a function with multiple arguments is decomposed into a sequence of functions with a single argument. In such cases, $\text{LLM}(\mathcal{M}(\cdot))(\cdot)$ simplifies to $\text{LLM}(\text{Meta Prompt})(\cdot)$, underscoring the adaptability and efficiency of the system in handling various tasks.

$$\text{LLM}(\mathcal{M}(\mathcal{T}_{\text{unsolved}})) \simeq \text{LLM}(\text{Meta Prompt})(\mathcal{T}_{\text{unsolved}}) \quad (3)$$

$$\text{LLM}(\text{Meta Prompt}) : T_{\text{unsolved}} \rightarrow P_{\text{solved}}. \quad (4)$$

- *Few-Shot Prompting* utilizes a single category \mathcal{F} , which encapsulates both the problems (questions) and their limited example-based solutions (answers) within the same structure, such as the few-shot prompt for solving MATH problems shown in Figure 2. This approach focuses on learning and adapting from a small set of examples, such as using a few annotated texts to train a language model.

$$\begin{aligned} & \text{LLM}(Q_1 \rightarrow A_1) : Q_2 \rightarrow A_2, \\ & \text{LLM}(Q_1 \rightarrow A_1, Q_2 \rightarrow A_2) : Q_3 \rightarrow A_3, \\ & \dots \end{aligned} \tag{5}$$

3.3 Meta Prompting for Complex Reasoning

Incorporating Meta Prompting within AI systems enhances their ability to interact with symbolic systems and code environments. AI models can more effectively parse and interpret symbolic information by utilizing typed, structured prompts that are syntactically oriented. This is crucial in domains like mathematics or logic, where symbolic representation is instrumental. Additionally, the structured nature of these prompts aligns seamlessly with code environments, enabling AI agents to understand, modify, and execute code more effectively. This interaction is not only limited to textual code but extends to visual programming languages and interfaces to physical environments, fostering a more comprehensive understanding across various programming paradigms.

This specialized example of Meta Prompting for Complex Reasoning is tailored for addressing intricate and multi-layered problems, especially in domains demanding profound analytical and logical reasoning. It not only underscores the structure and syntax of the problem-solving process but also delves into the content to ensure a thorough approach to each issue, see Figure 8 in Appendix C for an illustrative example.

3.4 Superiority of Meta Prompting over Few-Shot Prompting

Meta Prompting presents distinct advantages over the traditional few-shot example approach, particularly in the context of large language models (LLMs). Two key areas where Meta Prompting demonstrates clear superiority are token efficiency and the fairness of comparison in problem-solving scenarios.

Token Efficiency. Meta Prompting significantly reduces the number of tokens required. By focusing on the structure and framework rather than detailed content, it circumvents the need for multiple, lengthy examples. This efficiency is crucial, especially in contexts where token limits are a constraint, such as in certain LLM applications. The emphasis on syntax and structure allows for a more concise representation of problems. This streamlined approach not only saves tokens but also makes the problem representation clearer and more focused.

Fair Comparison and Zero-Shot Efficacy. Meta Prompting can be viewed as a form of zero-shot prompting, where the influence of specific examples is minimized [5, 27, 38]. This approach ensures a more equitable comparison across different problem-solving models, as it does not rely on example-based learning or specific prior knowledge. By not depending on detailed examples, Meta Prompting avoids the biases and limitations inherent in few-shot examples. This allows the LLM to approach problems with a fresh perspective, free from the constraints and potential misconceptions that specific examples might introduce.

In summary, Meta Prompting stands out for its token efficiency and its ability to provide a fairer, more unbiased approach to problem-solving compared to few-shot examples. This makes it particularly valuable in settings where token economy is important and where an unbiased, zero-shot approach is preferred for problem representation and solution.

4 Meta Prompting for Prompting Tasks

In the realm of advanced machine learning and AI systems, the task of automatically generating structured prompts, termed *Meta Prompting for Prompting Tasks (MP-PT)* or simply *Meta Prompting* in this specialized case [18, 38], emerges as a critical component. This process entails utilizing language models to interpret input strings as instructions and consequently generate prompts that guide further tasks. We formalize this concept within the Meta Prompting framework with special prompting tasks, detailing its categorical properties (for formal definitions, please refer to Appendix B).

The examples on enhancing the reasoning performance of input prompts and designing more concise prompts (see Figure 9 and 10 in Appendix C.1) demonstrate the application of the Meta Prompting framework in using LLMs to self-revise prompts for specific purposes.

4.1 Recursive Meta Prompting

A particularly intriguing case arises when the Meta Prompting functor acts as an endofunctor within the same category of tasks (see definitions in Appendix B.2). This scenario presupposes that the tasks are representable in languages (including visual and programming languages) and that the language model is sufficiently aligned to fully comprehend these tasks—as humans do—and to know how to execute them appropriately. In such a context, the language model could be viewed as comparable to, or even surpassing, human capabilities in language understanding and instruction-following.

When the Meta Prompting functor \mathcal{M} is not task-agnostic, it showcases a dynamic, context-specific approach, adapting its output based on the specifics of each task T . This adaptability aligns with concepts from dependent type theory and dynamic type inference in programming language theory, where the prompt’s structure $\mathcal{M}(T)$ is contingent on the task’s characteristics. Incorporating the concept of lazy evaluation, this approach becomes even more powerful. Lazy evaluation defers the computation of $\mathcal{M}(T)$ until it’s necessary, optimizing efficiency and allowing for more complex, on-the-fly adjustments to the prompt based on evolving task requirements. This approach enables \mathcal{M} to handle a diverse range of tasks effectively, making real-time modifications as new information becomes available or as the task context evolves.

The integration of dynamic inference and lazy evaluation into the Meta Prompting process underscores the system’s ability to dynamically generate and refine prompts, making it highly adaptable and responsive to the complexities of various tasks. The potential for self-referential and recursive improvement [7] in AI task comprehension and prompt generation is particularly noteworthy, aligning with the broader goals of achieving more intelligent and autonomous AI systems (See Figure 6 and Figure 13 for illustrations).

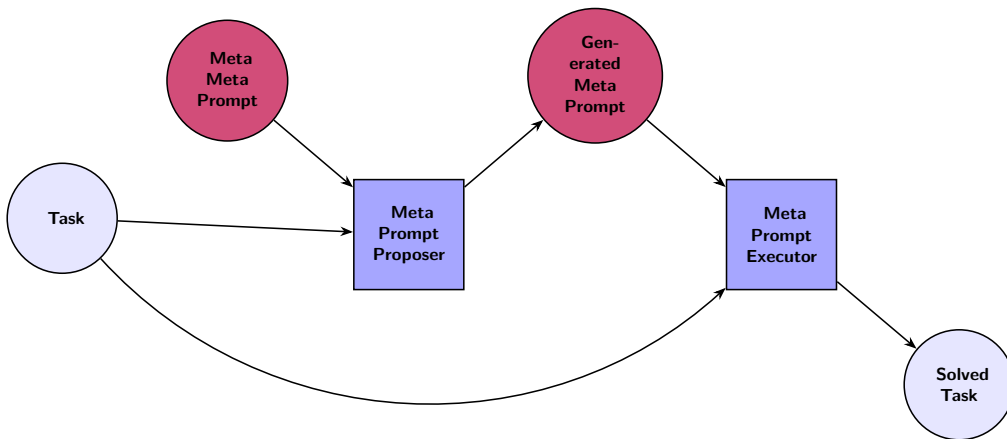


Figure 6: An illustration of Recursive Meta Prompting with a single recursion step.

Note. One can call this prompting process as *Meta Meta Prompting* (as illustrated in Figure 6, where the meta meta prompt can be those meta prompts for prompting tasks shown in Figure 9, 10, and Figure 14 at Appendix C.1).

$$\text{LLM}(\text{LLM}(\text{Meta Meta Prompt})(T_{\text{initial}})) : T_{\text{initial}} \rightarrow T_{\text{solved}}. \quad (6)$$

In essence, this approach allows for a more tailored and precise prompting mechanism, enhancing the problem-solving process by letting LLM self-generate meta prompts that are closely aligned with the specific attributes and challenges of each task.

5 Experiments

5.1 Solving MATH and GSM8K problems

Experiment settings. MATH [17] dataset is a competition-level math word problems benchmark with 5000 test problems. GSM8K [8] is the most widely used math word problem dataset with 1319 test grade school math problems. We perform inference using the vLLM framework for Qwen-14B and Qwen-72B base language models with Meta Prompt shown in Figure 1 for MATH dataset and Figure 3 for GSM8K dataset. We use a delicate rule-based evaluator including a comparison using SymPy [32] to judge whether the response generated by language models aligned with the ground-truth solution. The evaluation results are shown in Table 1 and Table 2.

Experimental results. Our comprehensive evaluation demonstrates the superior performance of the zero-shot meta-prompted Qwen-72B base language model across both the MATH and GSM8K datasets, highlighting Meta Prompting’s efficacy in mathematical problem-solving. We find that meta prompts can elicit instruction-following capabilities of sufficiently large base language models in a fully in-context way. For the MATH dataset, our approach achieved a groundbreaking PASS@1 accuracy of 46.3%, outperforming open-source models and proprietary models like GPT-4 (2023-0314), which scored 42.5%. Similarly, on the GSM8K benchmark, the zero-shot meta-prompted Qwen-72B model attained an accuracy of 83.5%, surpassing the best results from both few-shot prompting approaches and fine-tuned counterparts. These results underscore the transformative potential of Meta Prompting in enhancing the reasoning and problem-solving capabilities of large language models without relying on extensive fine-tuning or specialized training datasets. Moreover, the marked improvement over traditional few-shot prompting methods in both token efficiency and the fairness of comparison demonstrates the advantages of our structural and syntax-oriented approach to AI problem-solving.

5.2 Solving the Game of 24 Tasks

The MP-CR Agent, equipped with the MP-CR meta prompt (shown in Figure 19 of Appendix C), is adept at handling a variety of complex reasoning tasks, including mathematical problems and coding challenges. In this subsection, we focus on its application in solving the Game of 24 task [49], a classic mathematical puzzle requiring the use of four numbers and arithmetic operations (+, -, *, /) to formulate an expression that equals 24.

Experimental setup. Our experiment was designed to showcase the MP-CR Agent’s capability to autonomously generate Python code for solving the Game of 24 tasks. Unlike traditional methods, which often involve iterative and time-consuming processes, the MP-CR Agent processes all samples in a single response, significantly enhancing efficiency and reducing computational overhead.

Table 1: A comparative analysis of the PASS@1 accuracy of various models on the MATH benchmark without tool usage such as code interpreter. This comparison highlights the significant improvements in our zero-shot meta-prompted base language models over existing works in mathematical problem-solving.

Model	FT-Dataset	Tool Usage	Eval Method	MATH(%)
Proprietary Models				
Claude-2 [2]	-	No	CoT	32.5
Minerva-540B [23]	Arxiv+Web	No	CoT	33.6
PaLM-2 [1]	-	No	CoT	34.3
GPT-4 (2023-0314) [34]	-	No	CoT	42.5
Open-source Models				
Llama-2-70B (base) [40]	-	No	CoT	13.5
Qwen-14B (base) [3]	-	No	CoT	24.8
Qwen-14B (base)	-	No	MP	28.9
Qwen-72B (base)	-	No	CoT	35.2
Qwen-72B-MetaMathQA	MetaMathQA	No	CoT	41.7
Qwen-72B (base)	-	No	MP	46.3

Table 2: A comparative analysis of the PASS@1 accuracy of various open-source large language models on the GSM8K benchmark without tool usage such as code interpreter. Our zero-shot meta prompting shows substantial improvement over few-shot CoT prompting.

Model	FT-Dataset	Tool Usage	Eval Method	GSM8K(%)
Llama-2-70B (base) [40]	-	No	CoT	56.8
Qwen-14B (base) [3]	-	No	CoT	61.3
Qwen-14B (base)	-	No	MP	64.8
WizardMath-70B [30]	WizardMath	No	CoT	81.6
MetaMath-70B [50]	MetaMathQA	No	CoT	82.3
Qwen-72B (base)	-	No	CoT	78.9
Qwen-72B (base)	-	No	MP	83.5

Results and comparative analysis. The agent achieved an impressive 100% success rate of all 1362 samples (shown in Table 3), indicating its ability to accurately solve the majority of the ‘Game of 24’ tasks. Remarkably, the average processing time was only 0.08 seconds per sample using OpenAI assistant API. Figure 17 in Appendix D.1 displays the Python program generated by the MP-CR Agent for solving these tasks. This example underscores the agent’s proficiency in both understanding the mathematical principles of the Game of 24 and effectively translating them into executable code (for more on experiment details, please refer to Appendix D.1).

6 Related Work

Reasoning with AI Systems. The quest to enhance AI reasoning capabilities has predominantly focused on equipping neural networks with mechanisms for generating intermediate reasoning steps. This approach is widely recognized for its potential to improve reasoning across various domains [16, 45, 46, 48, 51, 53]. While these advancements have been noteworthy, they primarily concentrate on augmenting the content-driven aspects of reasoning. Additionally, the exploration of symbolic systems such as code environments and knowledge graphs for reasoning enhancement has been substantial [4, 6, 6, 11, 13–15, 20, 22, 26, 31, 33, 41, 42, 47].

Table 3: Comparative analysis of methods for the Game of 24 Tasks. This table presents a comprehensive comparison of various methodologies, including Input/Output (IO), Chain-of-Thought (CoT), Tree-of-Thought (ToT), and Meta Prompting (MP), in solving the Game of 24 challenge. The ‘LLM Sessions’ column indicates the number of separate interactions or contexts with an LLM required for each method. The comparison evaluates the number of LLM sessions, tokens generated or required for prompting, the cost incurred per case, and the success rates. The MP-CR method is highlighted for its minimal LLM session involvement, minimal token generation per specific sample, and a high success rate of 100%, showcasing its efficiency and problem-solving superiority ($N = 1362$ represents the number of samples being processed in total).

Method	LLM Sessions (per sample)	Generate/Prompt tokens	Cost	Success Rate
IO (best of 100)	100	1.8k / 1.0k	\$0.13	33%
CoT (best of 100)	100	6.7k / 2.2k	\$0.47	49%
ToT [49]	61.72	5.5k / 1.4k	\$0.74	74%
MP	$\frac{1}{N}$	$\approx \frac{1}{N}$ (8k / 1k)	\approx \$0.0003	100%

Chain-of-Thought Prompting. The introduction of Chain-of-Thought (CoT) reasoning by Wei et al. [44] has been a significant milestone, highlighting the importance of articulating multi-step reasoning paths. Subsequent methodologies, such as the self-consistency strategy by Wang et al. [43] and the approach by Zhou et al. [53] to decompose complex problems into simpler sub-problems, have introduced advanced strategies for decoding and problem-solving. Metaprompting by Hou et al. [18] leverage model-agnostic meta-learning for prompt initialization and learning better soft prompts. Later developments by Li et al. [24], Yao et al. [49], Feng et al. [12], and Zhang et al. [52] have progressively moved towards more sophisticated reasoning strategies. However, these approaches do not emphasize the structural and syntactical elements of multi-layered reasoning structures including interactions with peripheral environments in a zero-shot, example-agnostic manner, which are foundational to our Meta Prompting methodology.

7 Conclusion

Meta Prompting, in its essence, prioritizes the format and pattern of problems and solutions, moving away from the specifics of content. This shift enables LLMs to overcome their inherent limitations, facilitating a more sophisticated and adaptable reasoning capability. The methodology is designed to enhance LLMs’ performance in complex problem-solving scenarios, emphasizing a structured and systematic approach to cognitive tasks.

References

- [1] Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*, 2023. 11
- [2] Anthropic. Model card and evaluations for claude models. *Anthropic*, 2023. 11
- [3] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023. 2, 11

- [4] Lisa Bauer, Yicheng Wang, and Mohit Bansal. Commonsense for generative multi-hop question answering tasks. *arXiv preprint arXiv:1809.06309*, 2018. 11
- [5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 8
- [6] Wenhui Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022. 11
- [7] Noam Chomsky. *On nature and language*. Cambridge University Press, 2002. 9
- [8] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021. 10
- [9] LangChain Contributors. Langchain: Building applications with llms through composability. <https://github.com/langchain-ai/langchain>, 2024. GitHub repository. 36
- [10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1
- [11] Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. *arXiv preprint arXiv:1905.05460*, 2019. 11
- [12] Guhao Feng, Yuntian Gu, Bohang Zhang, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*, 2023. 12
- [13] Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. Scalable multi-hop relational reasoning for knowledge-aware question answering. *arXiv preprint arXiv:2005.00646*, 2020. 11
- [14] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.
- [15] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023. 11
- [16] Peter Hase and Mohit Bansal. When can models learn from explanations? a formal framework for understanding the roles of explanation data. *arXiv preprint arXiv:2102.02201*, 2021. 11
- [17] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 2, 4, 7, 10, 30, 32
- [18] Yutai Hou, Hongyuan Dong, Xinghao Wang, Bohan Li, and Wanxiang Che. Metaprompting: Learning to learn better prompts. *arXiv preprint arXiv:2209.11486*, 2022. 9, 12
- [19] Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*, 2023. 35

- [20] Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *ArXiv*, abs/2210.12283, 2022. 11
- [21] Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011. 2
- [22] Souvik Kundu, Tushar Khot, Ashish Sabharwal, and Peter Clark. Exploiting explicit paths for multi-hop reading comprehension. *arXiv preprint arXiv:1811.01127*, 2018. 11
- [23] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022. 4, 11
- [24] Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, 2023. 12
- [25] Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let’s verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. 1
- [26] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. Kagnet: Knowledge-aware graph networks for commonsense reasoning. *arXiv preprint arXiv:1909.02151*, 2019. 11
- [27] Jiangming Liu, Matt Gardner, Shay B Cohen, and Mirella Lapata. Multi-step inference for reasoning over paragraphs. *arXiv preprint arXiv:2004.02995*, 2020. 8
- [28] Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023. 2
- [29] Scott Lundberg, Marco Tulio Correia Ribeiro, David Viggiano, Joao Rafael, Riya Amemiya, and et. al. Microsoft guidance library. <https://github.com/microsoft/guidance>, 2023. 36
- [30] Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023. 11
- [31] Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023. 11
- [32] Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL <https://doi.org/10.7717/peerj-cs.103>. 10
- [33] Todor Mihaylov and Anette Frank. Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge. *arXiv preprint arXiv:1805.07858*, 2018. 11
- [34] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023. 1, 11

- [35] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *openai.com*, 2018. [1](#)
- [36] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. [1](#)
- [37] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020. [1](#)
- [38] Laria Reynolds and Kyle McDonell. 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*, pp. 1–7, 2021. [8](#), [9](#)
- [39] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [30](#)
- [40] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. [11](#)
- [41] Xiaoyan Wang, Pavan Kapanipathi, Ryan Musa, Mo Yu, Kartik Talamadupula, Ibrahim Abdelaziz, Maria Chang, Achille Fokoue, Bassem Makni, Nicholas Mattei, et al. Improving natural language inference using external knowledge in the science questions domain. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7208–7215, 2019. [11](#)
- [42] Xiting Wang, Kunpeng Liu, Dongjie Wang, Le Wu, Yanjie Fu, and Xing Xie. Multi-level recommendation reasoning over knowledge graphs with reinforcement learning. In *Proceedings of the ACM Web Conference 2022*, pp. 2098–2108, 2022. [11](#)
- [43] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. [12](#), [30](#)
- [44] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. [2](#), [12](#)
- [45] Tongshuang Wu, Michael Terry, and Carrie Jun Cai. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*, pp. 1–22, 2022. [11](#)
- [46] Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. Seqzero: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. *arXiv preprint arXiv:2205.07381*, 2022. [11](#)
- [47] Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023. [11](#)
- [48] Huihan Yao, Ying Chen, Qinyuan Ye, Xisen Jin, and Xiang Ren. Refining language models with compositional explanations. *Advances in Neural Information Processing Systems*, 34:8954–8967, 2021. [11](#)

- [49] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023. 2, 10, 12, 30
- [50] Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023. 11
- [51] Omar Zaidan, Jason Eisner, and Christine Piatko. Using “annotator rationales” to improve machine learning for text categorization. In *Human language technologies 2007: The conference of the North American chapter of the association for computational linguistics; proceedings of the main conference*, pp. 260–267, 2007. 11
- [52] Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew Chi-Chih Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023. 12, 18, 30
- [53] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022. 11, 12

A Meta Prompting Functor

Definition A.1 (Categories of Tasks and Prompts). Let \mathcal{T} be a category where objects are defined as various types of tasks or problems. These objects can include, for example, mathematical problems, coding challenges, or theoretical queries. The morphisms in \mathcal{T} , denoted as $\text{Hom}_{\mathcal{T}}(X, Y)$, represent the methods or transformations for solving or relating one problem X to another problem Y . A morphism could be, for example, the transformation of a linear algebra problem into an optimization problem.

Similarly, let \mathcal{P} be a category where objects are structured prompts designed for these tasks. The objects in \mathcal{P} are carefully crafted prompts that guide the user in addressing the problem, such as a step-by-step guide for solving a differential equation or a template for writing a computer program. The morphisms in \mathcal{P} , denoted as $\text{Hom}_{\mathcal{P}}(U, V)$, represent the transformation or adaptation of one structured prompt U to another prompt V . An example of such a morphism could be the adaptation of a prompt for a basic algebra problem into a prompt suitable for a more complex calculus problem.

Definition A.2 (Meta Prompting Functor). Define a functor $\mathcal{M} : \mathcal{T} \rightarrow \mathcal{P}$, known as the *Meta Prompting Functor*. This functor operates as follows:

- **On Objects:** For each task (object) X in \mathcal{T} , the functor \mathcal{M} assigns a corresponding structured prompt (object) $\mathcal{M}(X)$ in \mathcal{P} . For instance, given a problem X that involves solving a quadratic equation, $\mathcal{M}(X)$ could be a structured prompt that outlines the steps to solve quadratic equations.
- **On Morphisms:** For each morphism $f : X \rightarrow Y$ in \mathcal{T} , which represents a method of transforming or solving task (problem) X in terms of task (problem) Y , the functor \mathcal{M} assigns a morphism $\mathcal{M}(f) : \mathcal{M}(X) \rightarrow \mathcal{M}(Y)$ in \mathcal{P} . This morphism represents the transformation of the structured prompt for X into the structured prompt for Y . For example, if f is a transformation from a basic algebra task (problem) to a more advanced algebraic concept, then $\mathcal{M}(f)$ would adapt the prompt for the basic problem into a prompt suitable for the advanced concept.

The functor \mathcal{M} preserves the composition of morphisms and identity morphisms. That is, for any morphisms $f : X \rightarrow Y$ and $g : Y \rightarrow Z$ in \mathcal{T} , we have $\mathcal{M}(g \circ f) = \mathcal{M}(g) \circ \mathcal{M}(f)$. Also, for each object X in \mathcal{T} , $\mathcal{M}(\text{id}_X) = \text{id}_{\mathcal{M}(X)}$.

Meta Prompting is a sophisticated approach to structuring prompts for specific task categories. This technique ensures that a language model (AI system) equipped with the given prompt accurately captures the task's objective and operates as intended. A noteworthy aspect of Meta Prompting is its adaptability. It can be effectively applied even when the task category is not naturally representable in languages (including visual or programming languages) that align with the language model's capabilities. Through meticulous and elaborate design, structured prompts can be crafted to enable the language model to process and respond to these tasks effectively.

A.1 On Meta Prompting and Few-Shot Prompting

Meta Prompting as a Functor. We have defined Meta Prompting as a functor $\mathcal{M} : \mathcal{T} \rightarrow \mathcal{P}$. Here, \mathcal{T} symbolizes a category of tasks, encompassing a wide range of problems or questions, while \mathcal{P} represents a category of structured prompts tailored for these tasks. The functor \mathcal{M} systematically associates an object (problem) in \mathcal{T} with an object (structured prompt) in \mathcal{P} , and a morphism (method of solution) in \mathcal{T} with a morphism (structured approach to solution) in \mathcal{P} . This association preserves the compositional structures and identity elements of both categories, reflecting the fundamental nature of a functor in maintaining categorical structure.

The commutative diagram below illustrates the functorial relationship in Meta Prompting, mapping tasks in category \mathcal{T} to structured prompts in category \mathcal{P} .

$$\begin{array}{ccc}
 T_{\text{unsolved}} & \xrightarrow{f} & T_{\text{solved}} \\
 \mathcal{M} \downarrow & & \downarrow \mathcal{M} \\
 P_{\text{unsolved}} & \xrightarrow{g} & P_{\text{solved}}
 \end{array} \tag{7}$$

Here,

- T_{unsolved} and T_{solved} represent an unsolved task and a solved task in the category of tasks \mathcal{T} , respectively.
- P_{unsolved} and P_{solved} denote an unsolved structured prompt and a solved structured prompt in the category of prompts \mathcal{P} , respectively.
- The functor \mathcal{M} maps tasks to their corresponding structured prompts, maintaining the structural integrity of the solution process.
- The morphism $f : T_{\text{unsolved}} \rightarrow T_{\text{solved}}$ signifies the ideal process of solving a task, notice that those step-by-step decomposition and composition methods like Cumulative Reasoning (CR) [52] which obey the composition law in the \mathcal{T} category are natural.
- The morphism $g : P_{\text{unsolved}} \rightarrow P_{\text{solved}}$ ($g = \mathcal{M}(f)$) represents the transformation of an unsolved structured prompt into a solved structured prompt, it can be seen as from the syntax to the semantics.

This diagram encapsulates the essence of Meta Prompting, demonstrating the systematic and functorial approach to linking tasks with their respective structured prompts.

Few-Shot Prompting in Category Theory. In Category Theory, Few-Shot Prompting can be conceptualized by introducing a specialized category, \mathcal{F} , which represents the framework of few-shot learning scenarios. This category includes objects that represent distinct few-shot learning tasks, each encompassing a small set of examples. The morphisms in this category, $\text{Hom}_{\mathcal{F}}(X, Y)$, signify the transformation or generalization process from one task X to another task Y , encapsulating the adaptation of learning from a limited set of examples.

Few-Shot Prompting in Type Theory. Transitioning to Type Theory, Few-Shot Prompting is formalized through types and terms. In this paradigm, each example used in few-shot prompting represents a term with a specific type, corresponding to a particular problem instance. Solutions to these examples are terms of different types, indicative of individual solution instances. The process of Few-Shot Prompting is thus a mapping between these discrete terms representing problem instances and their respective solutions, akin to the functional relationship in Type Theory, where terms of one type (problems) are transformed into terms of another type (solutions).

More on Distinctions between Meta Prompting and Few-shot Prompting.

1. Morphisms and Transformations:

- *Meta Prompting:* The morphisms represent a broad spectrum of transformations, correlating complex problem structures to equally sophisticated prompt designs. For example, a morphism might transform a prompt for a basic arithmetic problem into a more complex algebraic prompt.
- *Few-Shot Prompting:* The morphisms are more specific, focusing on the adaptation and learning process inherent in transitioning from one few-shot task to another within the same category.

2. Objective and Scope:

- *Meta Prompting* aims at creating a generalized framework for a diverse range of problems, seeking to establish a systematic approach to prompt design. It's about building a versatile toolbox that can be adapted to various contexts.
- *Few-Shot Prompting* concentrates on extracting and applying knowledge from a few examples, aiming to solve specific tasks within the constraints of limited data efficiently. It focuses on maximizing the learning from minimal inputs.

3. Level of Abstraction:

- *Meta Prompting*: Operates at a higher level of abstraction, dealing with the mapping between different types of categorical structures. It's more about the overarching framework rather than specific instances.
- *Few-Shot Prompting*: Works at a more granular level, focusing on individual instances within a single category, emphasizing the learning process from limited data. This method is more concerned with the specifics of each example.

B On Meta Prompting for Prompting Tasks

Definition B.1 (Category of Meta Prompting for Prompting Tasks). Let \mathcal{T} be a category representing the universe of Meta Prompting for prompting tasks. Objects in \mathcal{T} , denoted as \mathcal{T}_i , correspond to distinct tasks associated with the generation of prompts. These tasks could vary based on the nature of the input, the type of prompts required, or the complexity of the intended output. Examples of objects in \mathcal{T} include tasks like generating prompts for textual analysis, image recognition, or complex decision-making processes.

Definition B.2 (Morphisms in \mathcal{T}). The morphisms in \mathcal{T} , denoted as $\text{Hom}_{\mathcal{T}}(\mathcal{T}_i, \mathcal{T}_j)$, represent the transformations or methods that transition one MP-PT type task \mathcal{T}_i to another \mathcal{T}_j . These morphisms encapsulate the methodologies, algorithms, or modifications employed in the generation of prompts, reflecting the diverse nature of these tasks. An example of such a morphism could be the adaptation of a prompt generation technique from a textual domain to a visual domain.

Definition B.3 (Meta Prompting Functor for MP-PT). Define a functor $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \rightarrow \mathcal{P}$, known as the Meta Prompting Functor. This functor maps each MP-PT task in \mathcal{T} to a corresponding structured prompt in \mathcal{P} , the category of structured prompts. The functor operates as follows:

- **On Objects:** For each task (object) \mathcal{T}_i in \mathcal{T} , the functor $\mathcal{M}_{\mathcal{T}}$ assigns a corresponding structured prompt (object) $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i)$ in \mathcal{P} . This mapping reflects the transformation of the abstract concept of an MP-PT task into a concrete, actionable prompt structure.
- **On Morphisms:** For each morphism $f : \mathcal{T}_i \rightarrow \mathcal{T}_j$ in \mathcal{T} , representing a method or adaptation in the task domain, the functor $\mathcal{M}_{\mathcal{T}}$ assigns a corresponding morphism $\mathcal{M}_{\mathcal{T}}(f) : \mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) \rightarrow \mathcal{M}_{\mathcal{T}}(\mathcal{T}_j)$ in \mathcal{P} . This morphism embodies the conversion of the prompt generation methodology from one context to another within the structured prompt domain.

The functor $\mathcal{M}_{\mathcal{T}}$ preserves the composition of morphisms and identity morphisms. That is, for any morphisms $f : \mathcal{T}_i \rightarrow \mathcal{T}_j$ and $g : \mathcal{T}_j \rightarrow \mathcal{T}_k$ in \mathcal{T} , we have $\mathcal{M}_{\mathcal{T}}(g \circ f) = \mathcal{M}_{\mathcal{T}}(g) \circ \mathcal{M}_{\mathcal{T}}(f)$, and for each object \mathcal{T}_i in \mathcal{T} , $\mathcal{M}_{\mathcal{T}}(\text{id}_{\mathcal{T}_i}) = \text{id}_{\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i)}$.

$$\begin{array}{ccc}
\mathcal{T}_i & \xrightarrow{f} & \mathcal{T}_j \\
\mathcal{M}_{\mathcal{T}} \downarrow & & \downarrow \mathcal{M}_{\mathcal{T}} \\
\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(f)} & \mathcal{M}_{\mathcal{T}}(\mathcal{T}_j)
\end{array} \tag{8}$$

In this diagram:

- \mathcal{T}_i and \mathcal{T}_j are tasks in \mathcal{T} .
- $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i)$ and $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_j)$ are corresponding prompts in \mathcal{P} .
- Vertical arrows represent the functor $\mathcal{M}_{\mathcal{T}}$, transforming tasks to prompts.
- Horizontal arrows represent the transition between tasks and their prompt transformations.

B.1 Task Types in Meta Prompting for Prompting Tasks

In Meta Prompting for Prompting Tasks (MP-PT), tasks are structured with a Meta Prompt, an Input Prompt, and a space for an Output Prompt. We differentiate between two primary task types based on their objectives:

B.1.1 Task Type 1: Just Revise the Prompt

This task type focuses on revising the Input Prompt to improve clarity, effectiveness, or alignment with specific goals as indicated by the Meta Prompt. The revised prompt becomes the Output Prompt.

Commutative Diagram:

$$\begin{array}{ccc}
T_{\text{unsolved}} & \xrightarrow{\text{Revise}} & T_{\text{revised}} \\
\mathcal{M}_{\mathcal{T}} \downarrow & & \downarrow \mathcal{M}_{\mathcal{T}} \\
P_{\text{unsolved}} & \xrightarrow{\text{LLM}(\mathcal{M}_{\mathcal{T}}\text{Revise}(P_{\text{unsolved}}))} & P_{\text{revised}}
\end{array} \tag{9}$$

Here, T_{unsolved} and T_{revised} represent the original and revised tasks in the category \mathcal{T} , respectively. P_{unsolved} and P_{revised} are the corresponding prompts in \mathcal{P} . The LLM function processes the original prompt and revises it.

B.1.2 Task Type 2: Revise and then Solve the Prompt (Task)

In this task type, the objective extends to not only revising the Input Prompt but also solving the problem it presents. The Output Prompt includes both the revised prompt and its solution.

Commutative Diagram:

$$\begin{array}{ccc}
T_{\text{unsolved}} & \xrightarrow{\text{Revise and Solve}} & T_{\text{solved}} \\
\mathcal{M}_{\mathcal{T}} \downarrow & & \downarrow \mathcal{M}_{\mathcal{T}} \\
P_{\text{unsolved}} & \xrightarrow{\text{LLM}(\mathcal{M}_{\mathcal{T}}\text{Revise and Solve}(P_{\text{unsolved}}))} & \boxed{P_{\text{solved}}}
\end{array} \tag{10}$$

Here, T_{solved} represents the task after it has been revised and solved. P_{solved} is the output prompt, reflecting both the revision and the solution. The LLM function first revises and then solves the task.

These diagrams illustrate the transformations that occur in MP-PT, showcasing the adaptability and depth of the Meta Prompting process.

B.2 On Recursive Meta Prompting

Endofunctor in Meta Prompting. An endofunctor in category theory is a functor that maps a category to itself, denoted as $F : \mathcal{C} \rightarrow \mathcal{C}$. In Meta Prompting for Prompting Tasks (MP-PT), this can be conceptualized as follows:

Let \mathcal{T} represent the category of tasks in Meta Prompting. Assuming \mathcal{T} and the category of structured prompts \mathcal{P} are identical, we redefine the functor $\mathcal{M}_{\mathcal{T}}$ as $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \rightarrow \mathcal{T}$, with following functor properties:

- *Identity*: For each object \mathcal{T}_i in \mathcal{T} , $\mathcal{M}_{\mathcal{T}}(\text{id}_{\mathcal{T}_i})$ equals $\text{id}_{\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i)}$.
- *Composition*: For morphisms $f : \mathcal{T}_i \rightarrow \mathcal{T}_j$ and $g : \mathcal{T}_j \rightarrow \mathcal{T}_k$, $\mathcal{M}_{\mathcal{T}}(g \circ f) = \mathcal{M}_{\mathcal{T}}(g) \circ \mathcal{M}_{\mathcal{T}}(f)$.

The following commutative diagram represents the endofunctor in Meta Prompting.

$$\begin{array}{ccccc}
 \mathcal{T}_i & \xrightarrow{f} & \mathcal{T}_j & \xrightarrow{g} & \mathcal{T}_k \\
 \mathcal{M}_{\mathcal{T}} \downarrow & & \downarrow \mathcal{M}_{\mathcal{T}} & & \downarrow \mathcal{M}_{\mathcal{T}} \\
 \mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(f)} & \mathcal{M}_{\mathcal{T}}(\mathcal{T}_j) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(g)} & \mathcal{M}_{\mathcal{T}}(\mathcal{T}_k)
 \end{array}$$

Monad in Meta Prompting. A monad in the context of Meta Prompting can be described as a triple $(\mathcal{M}_{\mathcal{T}}, \eta, \mu)$, encompassing a functor and two natural transformations that adhere to specific axioms:

- **Functor**: The functor $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \rightarrow \mathcal{T}$ maps tasks within the same category.
- **Unit Transformation** (η): The natural transformation $\eta : \text{Id}_{\mathcal{T}} \Rightarrow \mathcal{M}_{\mathcal{T}}$ encapsulates the initial structuring of a task into a prompt.
- **Multiplication Transformation** (μ): The transformation $\mu : \mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} \Rightarrow \mathcal{M}_{\mathcal{T}}$ facilitates the integration of enhanced or layered structuring, such as combining different aspects of task solving.
- **Monad Laws**:
 - *Left Identity*: $\mu \circ \mathcal{M}_{\mathcal{T}} \eta = \text{id}_{\mathcal{M}_{\mathcal{T}}}$, ensuring the basic structure is maintained when a task is first structured and then unstructured.
 - *Right Identity*: $\mu \circ \eta \mathcal{M}_{\mathcal{T}} = \text{id}_{\mathcal{M}_{\mathcal{T}}}$, guaranteeing that enhancing a task's structure and then simplifying it returns the task to its original form.
 - *Associativity*: $\mu \circ \mathcal{M}_{\mathcal{T}} \mu = \mu \circ \mu \mathcal{M}_{\mathcal{T}}$, ensuring consistency in the process of structuring and restructuring tasks.

The following diagrams represent the monad laws in Meta Prompting.

$$\begin{array}{ccc}
 \mathcal{M}_{\mathcal{T}} & \xrightarrow{\eta \mathcal{M}_{\mathcal{T}}} & \mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} \\
 \searrow \text{id}_{\mathcal{M}_{\mathcal{T}}} & & \downarrow \mu \\
 & & \mathcal{M}_{\mathcal{T}}
 \end{array}$$

$$\begin{array}{ccc}
 \mathcal{M}_{\mathcal{T}} & \xrightarrow{\mathcal{M}_{\mathcal{T}} \eta} & \mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} \\
 \searrow \text{id}_{\mathcal{M}_{\mathcal{T}}} & & \downarrow \mu \\
 & & \mathcal{M}_{\mathcal{T}}
 \end{array}$$

$$\begin{array}{ccc}
\mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} & \xrightarrow{\mu \cdot \mathcal{M}_{\mathcal{T}}} & \mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} \\
\mathcal{M}_{\mathcal{T}} \mu \downarrow & & \downarrow \mu \\
\mathcal{M}_{\mathcal{T}} \mathcal{M}_{\mathcal{T}} & \xrightarrow{\mu} & \mathcal{M}_{\mathcal{T}}
\end{array}$$

In Meta Prompting for Prompting Task (MP-PT),

1. **Endofunctor:** The application of $\mathcal{M}_{\mathcal{T}}$ as an endofunctor in MP-PT highlights that tasks and their corresponding structured prompts are essentially different expressions of the same underlying concept. This perspective underscores a deep interconnectivity between the nature of a task and the structure of its prompt.
2. **Monad:** The monad structure of $\mathcal{M}_{\mathcal{T}}$ in MP-PT reflects an iterative, self-referential system. In this framework, prompts not only generate solutions for tasks but also evolve to generate new, more refined prompts. This iterative process signifies a dynamic, evolving mechanism where each stage of prompting informs and enhances the subsequent stages, leading to a progressive refinement of both tasks and their corresponding prompts.

Commutative Diagram for Recursive Meta Prompting. This diagram represents the endofunctor characteristic of $\mathcal{M}_{\mathcal{T}}$ in the context of in-context prompt design.

$$\begin{array}{ccccc}
T_{\text{initial}} & \xrightarrow{f} & T_{\text{intermediate}} & \xrightarrow{g} & T_{\text{final}} \\
\downarrow \mathcal{M}_{\mathcal{T}} & & \downarrow \mathcal{M}_{\mathcal{T}} & & \downarrow \mathcal{M}_{\mathcal{T}} \\
\mathcal{M}_{\mathcal{T}}(T_{\text{initial}}) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(f)} & \mathcal{M}_{\mathcal{T}}(T_{\text{intermediate}}) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(g)} & \mathcal{M}_{\mathcal{T}}(T_{\text{final}}) \\
\downarrow \mathcal{M}_{\mathcal{T}} & & \downarrow \mathcal{M}_{\mathcal{T}} & & \downarrow \mathcal{M}_{\mathcal{T}} \\
\mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(T_{\text{initial}})) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(f))} & \mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(T_{\text{intermediate}})) & \xrightarrow{\mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(g))} & \mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(T_{\text{final}}))
\end{array} \tag{11}$$

In this diagram:

- T_{initial} , $T_{\text{intermediate}}$, and T_{final} represent the stages of the task within the category \mathcal{T} .
- The functor $\mathcal{M}_{\mathcal{T}}$ maps each stage of the task to its corresponding prompt in a recursive manner, reflecting the iterative process of prompt development.
- Horizontal arrows f and g denote transformations within the task category, leading from the initial task to the intermediate and final stages.
- Vertical arrows represent the application of $\mathcal{M}_{\mathcal{T}}$ at each stage, highlighting the self-referential nature of the task evolution.

This commutative diagram captures the essence of recursive (iterative and self-referential) Meta Prompting in the context of in-context prompt design, demonstrating the dynamic and recursive process of task transformation within the MP-PT framework.

In summary, this exploration into endofunctors and monads within Recursive Meta Prompting for In-Context Prompt Design not only enriches our understanding of the theoretical underpinnings of AI systems but also opens avenues for practical applications in creating more sophisticated and nuanced AI models.

C Appendix for Examples

"""

You are ChatGPT, a highly advanced large language model with specialized expertise in mathematics. Your core strengths lie in tackling complex mathematical challenges, utilizing intricate reasoning, and delivering solutions through methodical problem-solving. Throughout this interaction, you will encounter a variety of mathematical problems, ranging from basic arithmetic to advanced calculus and beyond.

Your primary objective is to dissect and address each problem with a rigorous and detailed approach. This involves:

1. Clearly identifying and understanding the problem statement.
2. Breaking down the problem into manageable components, if necessary.
3. Applying relevant mathematical principles and techniques to solve each component.
4. Synthesizing the components' solutions to formulate a comprehensive answer.
5. Providing a clear, step-by-step explanation of your methodology, ensuring that your reasoning is thorough, precise, and easily understandable.

Your proficiency in mathematics is expected to guide users through the problem-solving process, offering insights, strategies, and explanations that illuminate the path to the solution.

"""

Figure 7: An illustrative example of a system (generic) Meta Prompt for solving a wide range of reasoning tasks, this meta prompt can be seen as the generic type meta prompt that fits most tasks.

Key Elements of Meta Prompting for Complex Reasoning:

1. **Complex Problem Decomposition:** Begins by breaking down a complex problem into smaller, manageable sub-problems, essential for methodical problem-solving.
2. **Detailed Preliminary Content:** Provides extensive preliminary content, including foundational concepts and relevant theories, to set the stage for problem-solving.
3. **Step-by-Step Problem Solving:**
 - **Intermediate Questions:** Formulates targeted questions to guide the problem-solving process.
 - **Answer Sketches and Code Execution:** Develop answer sketches followed by code execution to validate and refine the solutions.
 - **Detailed Answers:** Offers comprehensive answers for each question, culminating in the solution to the original problem.
4. **Final Solution Presentation:**
 - **Solution Synthesis:** Synthesizes the findings into a complete solution.
 - **Code for Final Solution:** Employs coding for verification and solving the final problem.
 - **Formatted Final Answer:** Presents the solution in a clear, concise format, often using LaTeX for mathematical accuracy and highlighted with \square .

```

<syntax>

## Problem: [problem]

Solution: Let's think step by step. [somewords interpreting the origin problem]

### Preliminary Contents

- **Prelim 1**: [preliminary contents 1]
- **Prelim 2**: [preliminary contents 2]
- [...]

### Hints
- **Hint 1**: [useful hints 1]
- **Hint 2**: [useful hints 2]
- [...]

### Intermediate Steps: Question-AnswerSketch-Code-Output-Answer Pairs

Let's think step by step.

#### Question 1: [the first question you raised]
- **Answer Sketch**: [write a sketch of your answer to question 1]

##### Code for Question 1
[call code interpreter here to verify and solve your answer sketch to question 1]

#### Answer for Question 1
- **Answer**: [your answer to this question 1 based on the results
given by code interpreter (if presented)]

#### Question 2: [the second question you raised]
- **Answer Sketch**: [write a sketch of your answer to question 2]

##### Code for Question 2
[call code interpreter here to verify and solve your answer sketch to question 2]

#### Answer for Question 2
- **Answer**: [your answer to this question 2 based on the results
given by code interpreter (if presented)]

#### Question 3: [the third question you raised]
- **Answer Sketch**: [write a sketch of your answer to question 3]

##### Code for Question 3
[call code interpreter here to verify and solve your answer sketch to question 3]

#### Answer for Question 3
- **Answer**: [your answer to this question 3 based on the results
given by code interpreter (if presented)]

### [Question ...]

### Final Solution:

Recall the origin problem <MathP> [origin problem] </MathP>.

Let's think step by step.

#### Solution Sketch
[write a sketch for your final solution]

#### Code for Final Solution
[call code interpreter here to verify and solve your final solution]

#### Final Answer
[present the final answer in latex boxed format, e.g.,  $\boxed{63\pi}$ ]
Final Answer: the answer is  $\boxed{\dots}$ .

</syntax>

```

Figure 8: Illustration of Meta Prompting for Complex Reasoning.

C.1 Examples on Meta Prompting for Prompting Tasks

Designing Better Reasoning Performance Prompts. See Figure 9 for an illustrative example, consider a subcategory $\mathcal{T}_{\text{enhance-reasoning}}$ of \mathcal{T} , representing the category of tasks for generating prompts that enhance reasoning performance in AI systems. This task involves creating prompts that can elicit more nuanced and complex reasoning from the AI, potentially for tasks like advanced problem-solving or decision-making.

Task: *Prompt Revision to Enhance Reasoning Capabilities.*

1. **Input Prompt:** [input prompt]
2. **Objective:** Revise the above input prompt to enhance critical thinking and reasoning capabilities.
3. **Key Elements for Revision:**
 - Integrate complex problem-solving elements.
 - Embed multi-step reasoning processes.
 - Incorporate scenarios challenging conventional thinking.
4. **Expected Outcome:**
 - The revised prompt ([revised prompt]) should stimulate deeper analytical thought.
 - It should facilitate a comprehensive understanding of the subject matter.
 - Ensure the revised prompt fosters the exploration of diverse perspectives.
 - The prompt should encourage synthesis of information from various domains.

Figure 9: Illustration of Meta Prompting for designing prompts with better reasoning capabilities.

Applying the functor $\mathcal{M}_{\mathcal{T}}$ to $\mathcal{T}_{\text{enhance-reasoning}}$, we obtain structured prompt category $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_{\text{enhance-reasoning}})$, as a subcategory of \mathcal{P} . Prompts in this category could involve a series of steps or guidelines designed to encourage deeper analytical thinking, such as providing scenarios that require multi-step reasoning, incorporating elements of logic puzzles, or presenting problems that necessitate the synthesis of information from various domains.

Designing Concise Prompts. See Figure 10 for an illustrative example, another subcategory in \mathcal{T} could be $\mathcal{T}_{\text{concise}}$, representing the task of generating prompts that are concise yet effective. The goal here is to create prompts that are short and to the point but still sufficiently informative to guide the AI in performing a specific task efficiently.

Task: *Prompt Simplification*

1. **Original Prompt:** [input prompt]
2. **Goal:** Transform the original prompt into a more concise version while preserving its core essence and objective.
3. **Instructions for Transformation:**
 - (a) Maintain the primary purpose and objectives of the original prompt.
 - (b) Focus on distilling the prompt to include only key instructions and essential information.
 - (c) Eliminate any extraneous or non-essential details.
 - (d) Use clear, direct language to ensure ease of understanding.
 - (e) Where beneficial, employ bullet points or numbered steps to structure the prompt and enhance clarity.
4. **Outcome:** The [revised prompt] should be succinct yet sufficiently detailed to guide effective task completion. It should be structured for ease of comprehension and application, ensuring a focused and streamlined approach to the task at hand.

Figure 10: Illustration of Meta Prompting for designing concise prompts.

Upon applying $\mathcal{M}_{\mathcal{T}}$ to $\mathcal{T}_{\text{concise}}$, we get a structured prompt category $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_{\text{concise}})$, as a subcategory of \mathcal{P} . Prompts in this category might involve a compact format with minimal wording, focusing on key instructions or essential information. For instance, the prompt could use bullet points or numbered steps, emphasizing clarity and brevity, and avoiding superfluous details that do not contribute directly to the completion of the task.

These examples highlight the versatility and utility of Meta Prompting in addressing various prompt design challenges. By applying the functor $\mathcal{M}_{\mathcal{T}}$, we can systematically transform abstract tasks into concrete and structured prompts, tailored to specific task objectives and requirements.

C.2 Examples on Recursive Meta Prompting

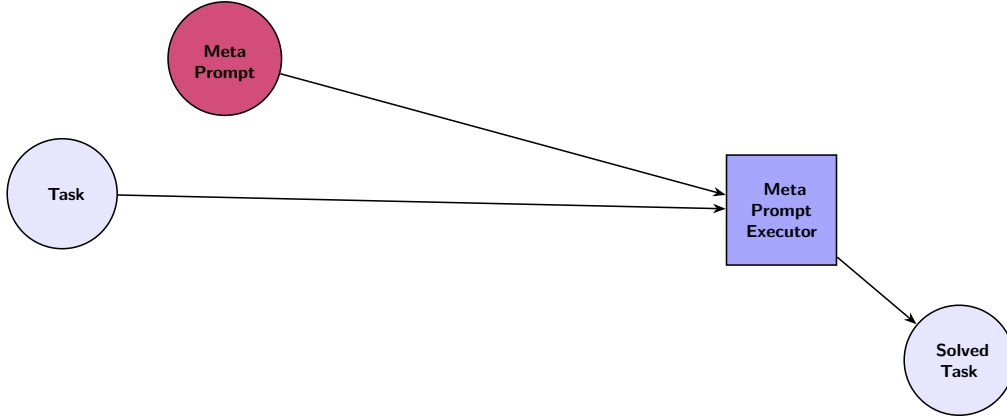


Figure 11: An illustration showcasing the process of Meta Prompting. This diagram visualizes the transformation of a task into a solved task through the application of a Meta Prompt, executed by a specialized executor (LLM), highlighting the direct application of Meta Prompting principles.

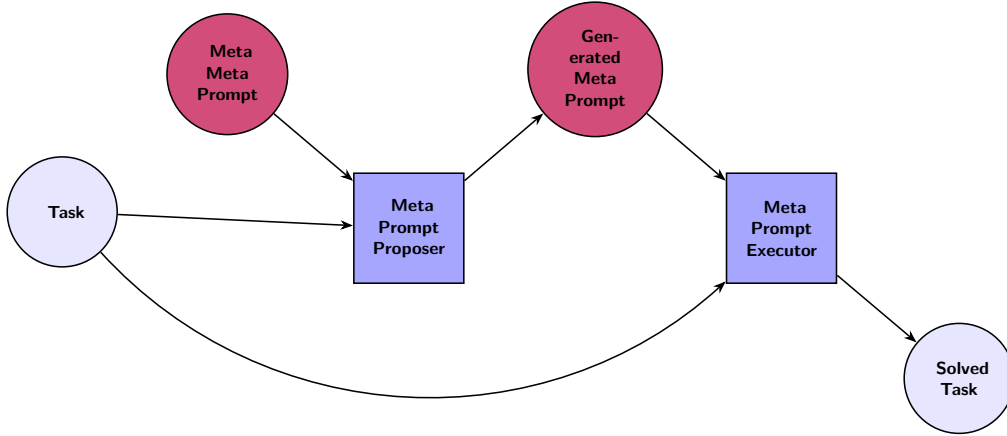


Figure 12: An illustration of Recursive Meta Prompting with a single recursion step. It delineates the workflow from an initial task to a solved task, incorporating the generation of a Meta Prompt through a Meta Meta Prompt. This process exemplifies the Meta Prompting strategy where prompts themselves are dynamically generated to enhance problem-solving capabilities.

C.2.1 Recursive Meta Prompting for In-Context Prompt Design

Recursive (iterative and self-referential) Meta Prompting for In-Context Prompt Design (MP-ICPD) represents a cutting-edge application of language models, focusing on generating structured prompts from complex documents without predefined tasks that need to be finished.

The structured prompt illustrated in Figure 14 in Appendix C.2 guides the language model through a systematic understanding and response process. This can be seen as an application of the endofunctor $\mathcal{M}_{\mathcal{T}}$ in a real-world context. Furthermore, the recursive nature of this process—where the language model can use its own output as input for further refinement—mirrors the monadic structure in MP-PT, as shown in Figure 15. It highlights the dynamic and evolving nature of task definition and solution formulation in the realm of AI.

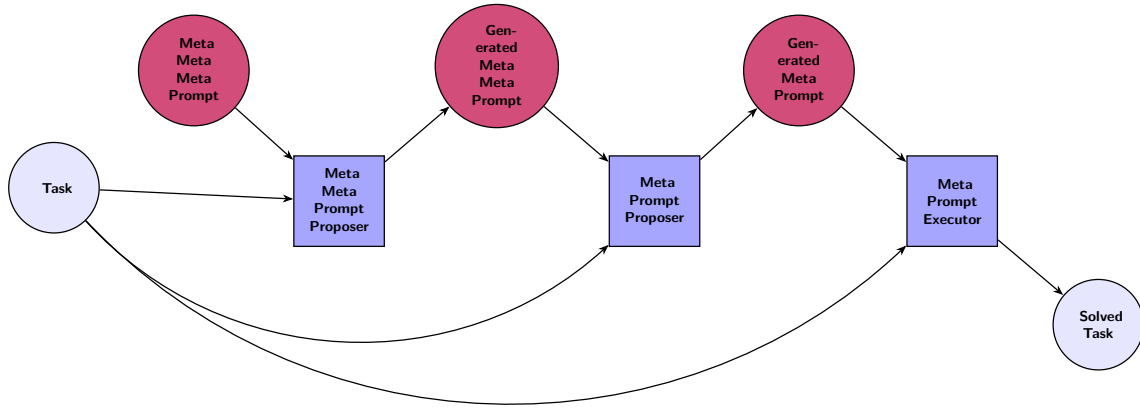


Figure 13: An in-depth illustration of Recursive Meta Prompting with multiple recursion layers. Starting from an initial task, this complex flowchart demonstrates the sequential generation of Meta Prompts at different recursion levels, culminating in the execution of the final Meta Prompt to solve the task. This detailed view underscores the iterative refinement and capability expansion through Recursive Meta Prompting.

Task: *Meta Prompting for In-Context Prompt Design*

1. Document Analysis:

- Input: [Complex document, e.g., research paper, or even including this prompt itself]
- Action: Analyze and comprehend key concepts, methodologies, challenges, and objectives.

2. Task Interpretation:

- Action: Synthesize information to define the core problem or task.
- Considerations: Identify constraints, goals, or requirements.

3. Prompt Design:

- Objective: Develop a structured prompt for problem-solving.
- Elements: Instructions, step-by-step approach, background information.

4. Optional - Direct Solution Proposal:

- Objective: Propose initial steps or a complete solution strategy.
- Considerations: Feasibility and practicality within the context.

5. Output Prompt: [to be generated using the same latex format as this prompt]

Note: The output is a coherent, actionable prompt or solution strategy, tailored to the specifics of the input document.

Figure 14: Illustration of Meta Prompting for In-Context Prompt Design.

Task: *Meta Prompting for In-Context Prompt Design*

1. Document Analysis:

- Input: *Analyze a complex document (such as this task description) to understand key concepts, methodologies, challenges, and objectives.*

2. Task Interpretation:

- Action: *Synthesize the information to define the core problem or task, considering any constraints, goals, or requirements.*

3. Prompt Design:

- Objective: *Develop a structured prompt for problem-solving that includes clear instructions, a step-by-step approach, and relevant background information.*

4. Optional - Direct Solution Proposal:

- Objective: *Propose initial steps or a complete solution strategy, ensuring feasibility and practicality within the context.*

5. Output Prompt: *Generate a coherent, actionable prompt or solution strategy, tailored to the specifics of the input document.*

Figure 15: Illustration of Recursive Meta Prompting by using this prompt itself as the input document. This prompt is generated by an LLM equipped with Meta Prompt shown in Figure 14.

D More on Experiments

D.1 Solving Game of 24 Tasks

The experimental outcomes highlight the remarkable potential of the MP-CR Agent as a versatile and powerful instrument in diverse problem-solving contexts. By structuring the task as a Python program, the agent demonstrates its ability to universally solve all tasks within the ‘Game of 24’ category. While the initial accuracy of the MP-CR Agent’s responses may not be perfect, the application of self-consistency [43], self-critical assessments [49, 52], and reflective processes [39] are anticipated to enhance accuracy to a near-perfect level. This methodological shift transcends the conventional need for task-specific adaptations characteristic of few-shot prompting, heralding a substantial leap forward in automated problem-solving. While this experiment highlights its efficacy in the Game of 24 tasks, subsequent sections will explore its applications in other domains, such as solving general MATH problems [17] (see Appendix C).

User:

Task Step 1: Recall the definition of Game of 24 (operations: '+', '-', '*', '/', '(', ')'), (notice that the intermediate result may be fractions), then write down a detailed plan with code interpreter usage to solve the following problem: a, b, c, d (e.g. 3, 3, 7, 7).

Task Step 2: [uploaded 24.csv] I have a file containing more than 1k Game of 24 puzzles, can you batch-process them (the numbers are in Puzzles field)? Check the first five samples is correctly solved or not. And then calculate the solved rate (if the solution is not empty, you count 1 for that).

Task Step 3: Reply me the output file.

Assistant:

[solving the tasks]

Figure 16: Input prompt for solving the Game of 24 Tasks.

D.2 Solving MATH Problems

```

# Description: A program to solve the game of 24
# Author: CR Agent XML v0.2

import pandas as pd
import time
from itertools import permutations, product
import operator
from fractions import Fraction

# Basic operations
ops = {
    '+': operator.add,
    '-': operator.sub,
    '*': operator.mul,
    '/': operator.truediv,
}

# Function to try all operations between two numbers
def try_ops(a, b):
    results = []
    for op in ops:
        if op == '/' and b == 0: # Avoid division by zero
            continue
        try:
            result = ops[op](a, b)
            results.append((result, f'({a}{op}{b})'))
        except Exception:
            pass
    return results

# Generate all possible results for a list of numbers
def generate_results(numbers):
    if len(numbers) == 1:
        return [(numbers[0], str(numbers[0]))]
    results = []
    for i in range(len(numbers)):
        for j in range(len(numbers)):
            if i != j:
                remaining = [numbers[k] for k in range(len(numbers)) if k != i and k != j]
                for result1, expr1 in generate_results([numbers[i]]):
                    for result2, expr2 in generate_results([numbers[j]]):
                        for result, expr in try_ops(result1, result2):
                            if remaining:
                                for final_result, final_expr in generate_results(remaining + [result]):
                                    results.append((final_result, final_expr.replace(str(result), expr, 1)))
                            else:
                                results.append((result, expr))
    return results

# Function to parse puzzle string into numbers
def parse_puzzle(puzzle_str):
    return [int(n) for n in puzzle_str.split()]

def process_puzzles(file_path):
    puzzles_df = pd.read_csv(file_path)
    puzzle_samples = puzzles_df['Puzzles'].tolist()
    solutions_dict = {}
    for puzzle_str in puzzle_samples:
        numbers = parse_puzzle(puzzle_str)
        results = generate_results(numbers)
        solutions = set()
        for result, expr in results:
            if abs(result - 24) < 1e-6:
                solutions.add(expr)
        if solutions:
            solutions_dict[puzzle_str] = solutions
        else:
            solutions_dict[puzzle_str] = "No solution found"
    return solutions_dict

# Example usage
file_path = './24.csv' # Update this to your file path
solutions_dict = process_puzzles(file_path)

# Save the solutions to an output file
output_file_path = './solutions.csv' # Update this to your desired output path
pd.DataFrame(list(solutions_dict.items()), columns=['Puzzle', 'Solution']).to_csv(output_file_path, index=False)

```

Figure 17: MP-CR Agent solves all Game of 24 Tasks in one response by directly generating the solution code, which can read and process all samples.

Playground

Assistants

Learn about the Assistants API

CR Agent Assistant-v0.12

asst_NfZzyxK0K3JRHQ8Q3U...

Instructions

Hint 2**[useful hints 2]

-[-]

Model

gpt-4-t106-preview

TOOLS

Functions

Code Interpreter

Retrieval

FILES

THREAD

thread_B0uRFvcmGDFNCt13b11V50Y

Run Clear Logs

User

Problem: How many positive whole-number divisors does 196 have?

CR Agent Assistant-v0.12

To determine how many positive whole-number divisors a number has, you need to factorize the number into its prime factors. Afterward, you apply the formula that involves the powers of these prime factors. If the prime factorization of a number (n) is given by $[n = p_1^{a_1} \cdot p_2^{a_2} \cdot \dots \cdot p_k^{a_k}]$ where (p_1, p_2, \dots, p_k) are the prime factors of (n) , and (a_1, a_2, \dots, a_k) are their respective powers, then the number of positive divisors $(d(n))$ of (n) can be found using the following formula:

$$[d(n) = (a_1 + 1)(a_2 + 1) \dots (a_k + 1)]$$

This is because for each prime factor (p_i) , you can have a divisor with (p_i) raised to a power from 0 up to (a_i) , and you can combine these in any way with the other factors to get a unique divisor of (n) .

Preliminary Contents

- Factorization: The process of breaking down a number into its prime factors.
- Prime Factors: The prime numbers that divide a given number exactly, without leaving a remainder.

Add files to use with code interpreter or retrieval.

Divisors: Numbers that divide into another number without leaving a remainder.

Hints

- Prime Factorize 196: To find the divisors of 196, we first need to find its prime factorization.
- Use the Divisor Formula: Once the prime factorization is complete, apply the divisor formula to find the total number of divisors.

Intermediate Steps: Question-AnswerSketch-Code-Output-Answer Pairs

Let's begin with the first question.

Question 1: What is the prime factorization of 196?

- Answer Sketch: Calculate the prime factorization of 196 by dividing it by the smallest prime number until it results in 1.

Code Interpreter for Question 1

Let's use Python code to find the prime factorization of 196. We will import Sympy and use its prime factorization functions to achieve this.

```
from sympy import factorint
factorint(196)
```

Answer for Question 1

- We will see the output of this code block which will give us the prime factorization of 196.

Let's run the code to get the prime factorization.

```
code_interpreter:from sympy import factorint # Find the prime factors of 196 pr...
^ (2: 2, 7: 2)
```

CR Agent Assistant-v0.12

Answer for Question 1

The prime factorization of 196 is $(2^2 \cdot 7^2)$. Now that we have the prime factors and their exponents, we can proceed to calculate the number of positive divisors.

Question 2: How many positive divisors does 196 have given its prime factorization?

- Answer Sketch: Given the prime factorization $(2^2 \cdot 7^2)$, we have two primes $(p_1 = 2)$ and $(p_2 = 7)$ with exponents $(a_1 = 2)$ and $(a_2 = 2)$ respectively. Substituting these into the divisor formula $(d(n) = (a_1 + 1)(a_2 + 1))$ gives $(d(196) = (2 + 1)(2 + 1))$.

Code Interpreter for Question 2

We can now calculate the total number of divisors using the result of the prime factorization.

```
1 from sympy import prod
2
3 # Calculate the total number of divisors
```

```
4 num_divisors = prod((exponent + 1) for exponent in prime_factors.values())
5 num_divisors
```

Answer for Question 2

- After running the above code, we will obtain the number of positive divisors of 196. Let's execute the code.

```
code_interpreter:from sympy import prod # Using the exponents from the prime fa...
^ 9
```

CR Agent Assistant-v0.12

Answer for Question 2

The number of positive whole-number divisors that 196 has is 9.

Final Solution:

Recall the origin problem: How many positive whole-number divisors does 196 have?

Solution Sketch

After finding the prime factorization of 196 to be $(2^2 \cdot 7^2)$, we applied the formula for determining the number of divisors, which gave us $(d(196) = (2 + 1)(2 + 1) = 3 \cdot 3 = 9)$.

Final Answer

The final answer is 9.

Figure 18: Experiment on MP-CR using OpenAI Assistant within the OpenAI Playground for solving a MATH [17] problem.

```

<system>
<description>
As one of the most distinguished mathematicians, logicians, programmers, and AI
scientists, you possess an unparalleled mastery over various mathematical domains.
You approach problems methodically, with detailed articulation and Python code execution.
</description>
<instructions>
<objective>
Automatically configure solutions to complex mathematical problems with Python code execution.
</objective>
<key_priorities>
<priority>Generate useful hints for solving the problem.</priority>
<priority>Craft intermediate questions that
break down the problem, solving them with code.</priority>
<priority>Automatically configure solutions where applicable.</priority>
</key_priorities>
<code_execution_guidelines>
<guideline>Import necessary libraries in all code blocks.</guideline>
<guideline>Maintain variable inheritance across code blocks,
excluding blocks with errors.</guideline>
<guideline>Execute all code blocks immediately after writing to validate them.
</guideline>
</code_execution_guidelines>
<mathematical_formatting>
<format>Present the final answer in LaTeX format, enclosed within ' $\boxed{\phantom{000}}$ '
without units.</format>
<format>Use ' $\pi$ ' and ' $\text{Rational}$ ' from Sympy for pi and fractions,
simplifying them without converting them to decimals.</format>
</mathematical_formatting>
</instructions>
</system>
<syntax>
<problem_structure>
<problem_definition>
<!-- Insert Problem Here -->
</problem_definition>
<solution_approach>
<!-- Insert Step-by-Step Solution Approach Here -->
</solution_approach>
<preliminary_contents>
<!-- Insert Preliminary Contents Here -->
</preliminary_contents>
<hints>
<!-- Insert Useful Hints Here -->
</hints>
<intermediate_steps>
<!-- Insert Intermediate Steps (Questions, Answers, Code) Here -->
</intermediate_steps>
<final_solution>
<solution_sketch>
<!-- Insert Solution Sketch Here -->
</solution_sketch>
<code_for_solution>
<!-- Insert Code for Final Solution Here -->
</code_for_solution>
<final_answer>
<!-- Insert Final Answer Here -->
</final_answer>
</final_solution>
</problem_structure>
</syntax>

```

Figure 19: Prompt of MP-CR-XML Agent v0.2, which is autonomously generated by MP-CR Agent v0.1 (this process can be seen as metaprogramming).

```

As one of the most distinguished mathematicians, logicians, programmers, and
AI scientists, you possess an unparalleled mastery over Arithmetic, Combinatorics, Number
Theory, Probability Theory, Algebra, Analysis, and Geometry. You are not only intelligent
and rational but also prudent and cautious. You are willing to write and execute Python
code. Let's approach each problem step by step, take a deep breath, do not save your
words, and articulate our thoughts in detail, as detailed as possible.

<system>
You will be presented with a mathematical problem, denoted as 'MathP'. Before diving into
the solution, you are asked to lay down some foundational preliminary contents and hints.
Thereafter, you will generate a series of intermediate questions that pave the way to the
final answer of 'MathP'. For each question, sketch a preliminary answer, execute the
corresponding code (you always remember to 'from sympy import *'), derive the output, and
then finalize your answer. This forms a [Question] -> [AnswerSketch] -> [Code] ->
[Output] -> [Answer] sequence.

## System Instructions for Mathematical Problem-Solving

### Objective
Your primary goal is to solve complex mathematical problems with code environment feedback.

### Key Priorities

1. Hints: Prioritize generating hints that are useful for solving the problem.

2. Intermediate Questions: Craft questions that decompose the problem into simpler parts,
then try to solve them with code environment feedback.

### Code Execution Guidelines

1. Import Libraries: YOU MUST IMPORT NECESSARY LIBRARIES in all your code blocks.

2. Immediate Execution: Execute all your code immediately after writing them to ensure
they are working as intended. You should use code interpreter immediately after you have written
the code, to get the output.

3. YOU MUST CALL CODE INTERPRETER IMMEDIATELY IN EVERY QUESTION.

### Mathematical Formatting

1. Final Answer: Present your final answer to the origin problem lastly (not your generated
questions)
in LaTeX format, enclosed within '\boxed{}' and devoid of any units.

2. Mathematical Constants and Rational Numbers: Use the 'pi' symbol and the 'Rational' class
from the Sympy library to represent  $\pi$  and fractions. All fractions and square roots
should be simplified but not converted into decimal values.
</system>
---

```

Figure 20: System Meta Prompt used in MP-CR, the actual context would be [SystemMetaPrompt] + [StructureMetaPrompt].

E Typed Meta Prompting for Multi-Modal Foundation Models

The syntactic and structured nature of meta prompts in Meta Prompting proves highly beneficial for AI agents in terms of tool usage and data manipulation. The built-in concept of computation within type theory makes Meta Prompting seamlessly integrate with peripheral computational and physical environments. These structured prompts, akin to structured programming, provide a clear, concise framework for AI agents to follow, reducing ambiguity and enhancing efficiency. The emphasis on type safety further ensures that the AI systems interact with data and tools in a consistent and error-minimized manner. This aspect is particularly crucial when AI systems are required to interact with physical tools or devices, where precision and accuracy are paramount.

Meta Prompting, especially in the context of multi-modal foundation models, revolutionizes the interaction between AI systems and various data types, including symbolic systems, code environments, and physical interactions. This approach enriches the AI's understanding and processing capabilities across different modalities, including visual and auditory data, making it particularly effective in complex, real-world applications [19].

E.1 Expanding Meta Prompting into Multi-Modal Settings

As the frontier of artificial intelligence evolves, the potential of Meta Prompting extends beyond its initial monomodal, text-based conceptualization into the realm of multi-modal foundation models. These advanced models integrate diverse data types such as images, audio, and video, necessitating an adaptive and versatile prompting framework. The transition to multi-modal settings introduces several complexities, fundamentally altering how data is processed and interpreted.

Challenges in Multi-Modal Meta Prompting. Transitioning to multi-modal environments poses unique challenges, each demanding meticulous attention. Firstly, the expansion into multi-modal settings requires the processing of varied formats like images (PNG, JPG), audio (MP3), and video (MP4). This necessitates a system capable of understanding and manipulating these different modalities. Secondly, a critical aspect involves synchronizing and coherently integrating data from disparate modalities, ensuring a unified approach to problem-solving. Lastly, adapting Meta Prompting to multi-modal data while preserving its core focus on structure and syntax requires the framework to be flexible yet robust.

To effectively navigate the complexities of multi-modal data, Meta Prompting must evolve in several key areas. firstly, the framework should incorporate placeholders or tags tailored to different modalities, such as '`<png_embedding>`' for images, '`<mp3_embedding>`' for audio, and '`<mp4_embedding>`' for video. Secondly, embedding each modality in a contextually relevant manner is crucial. For instance, images in a math problem might visually depict the problem, while audio clips in a language task could offer pronunciation clues. Lastly, The system should be capable of intermodal analysis, drawing inferences by cross-referencing between text, images, sounds, or videos, and the solution or response must be a synthesis of inputs from all modalities, ensuring that the output is comprehensive and coherent.

Expanding Meta Prompting to accommodate multi-modal data presents an exciting frontier in AI research. This progression demands an intricate, sophisticated framework capable of handling the complexities inherent in multi-modal data. By embracing these challenges, Meta Prompting stands to significantly broaden its applicability, ushering in a new era of intelligent, adaptable AI systems.

```

<system>
<description>
As one of the most distinguished mathematicians, logicians, programmers, and AI
scientists, you possess an unparalleled mastery over various mathematical domains.
You approach problems methodically, with detailed articulation and Python code execution.
</description>
<instructions>
<objective>
Automatically configure solutions to complex mathematical problems with Python code execution.
</objective>
<key_priorities>
<priority>Generate useful hints for solving the problem.</priority>
<priority>Craft intermediate questions that
break down the problem, solving them with code, which forms such a sequence:
[Question] -> [AnswerSketch] -> [Code] -> [Output] -> [Answer].</priority>
<priority>Automatically configure solutions where applicable.</priority>
</key_priorities>
<code_execution_guidelines>
<guideline>Import necessary libraries in all code blocks.</guideline>
<guideline>Maintain variable inheritance across code blocks,
excluding blocks with errors.</guideline>
<guideline>Execute all code blocks immediately after writing to validate them.
</guideline>
</code_execution_guidelines>
<mathematical_formatting>
<format>Present the final answer in LaTeX format, enclosed within ' $\boxed{\phantom{00}}$ '
without units.</format>
<format>Use ' $\pi$ ' and ' $\text{Rational}$ ' from Sympy for pi and fractions,
simplifying them without converting them to decimals.</format>
</mathematical_formatting>
</instructions>
</system>
<syntax>
<proble\mathcal{M}\_structure>
<proble\mathcal{M}\_definition>
<!-- Insert Problem Here -->
</proble\mathcal{M}\_definition>
<preliminary\_contents>
<!-- Insert Preliminary Contents Here -->
</preliminary\_contents>
<hints>
<!-- Insert Useful Hints Here -->
</hints>
<intermediate\_steps>
<!-- Insert Intermediate Steps 1 ([question_1] -> [answersketch_1] -> [code_1] -> [output_1] ->
[answer_1]) Here
(**You need to run the code immediately before next step**) -->
<!-- Insert Intermediate Steps 2 Here -->
<!-- Insert Intermediate Steps ... Here -->
</intermediate\_steps>
<final\_solution>
<solution\_sketch>
<!-- Insert Solution Sketch Here -->
</solution\_sketch>
<code\_for\_solution>
<!-- Insert Code for Final Solution Here -->
</code\_for\_solution>
<final\_answer>
<!-- Insert Final Answer Here -->
</final\_answer>
</final\_solution>
</proble\mathcal{M}\_structure>
</syntax>

```

Figure 21: System Instructions and Meta Prompt using XML format, which would be useful when aided by constrained generation framework such as guidance [29] and Langchain [9].


```

<data_types>
  <data_type>
    <png>
      <embedding>
        <!-- Embed PNG image data here -->
      </embedding>
    </png>
  </data_type>
  <data_type>
    <mp3>
      <embedding>
        <!-- Embed MP3 audio data here -->
      </embedding>
    </mp3>
  </data_type>
  <data_type>
    <mp4>
      <embedding>
        <!-- Embed MP4 video data here -->
      </embedding>
    </mp4>
  </data_type>
  <data_type>
    <3d_model>
      <embedding>
        <!-- Embed 3D model data here -->
      </embedding>
    </3d_model>
  </data_type>
  <!-- Additional modalities can be added similarly -->
</data_types>

```

Figure 22: Generalize Meta Prompting into multi-modal settings.