Meta Prompting for AI Systems

Yifan Zhang[1] **Yang Yuan**[1,2] **Andrew Chi-Chih Yao**[1,2]

[1]IIIS, Tsinghua University
[2]Shanghai Qizhi Institute
{yuanyang,andrewcyao}@tsinghua.edu.cn

Abstract

This paper presents a comprehensive study of Meta Prompting, an innovative technique reshaping the utilization of large language models (LLMs), multi-modal foundation models, 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 (MP), 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 (MP-CR) 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. The paper also introduces the integration of Meta Prompting into multi-modal foundation model settings, tackling the challenges and opportunities of incorporating varied data types such as images, audio, and video within the structured Meta Prompting framework. Empirical experiments, including solving the Game of 24 tasks with 100% success rate, demonstrate the MP-CR Agent's enhanced reasoning capabilities, achieving high accuracy and efficiency, and 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 problem-solving [2, 6, 27–30]. 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 [20]. This limitation points towards a need for enhanced methodologies that can augment LLMs' reasoning faculties.

The root of this challenge lies in the fundamental architecture of LLMs, which is predominantly oriented towards auto-regressive token prediction. While efficient for a broad spectrum of tasks, this approach lacks the necessary framework 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

[†]The code is available at `https://github.com/meta-prompting/meta-prompting`.

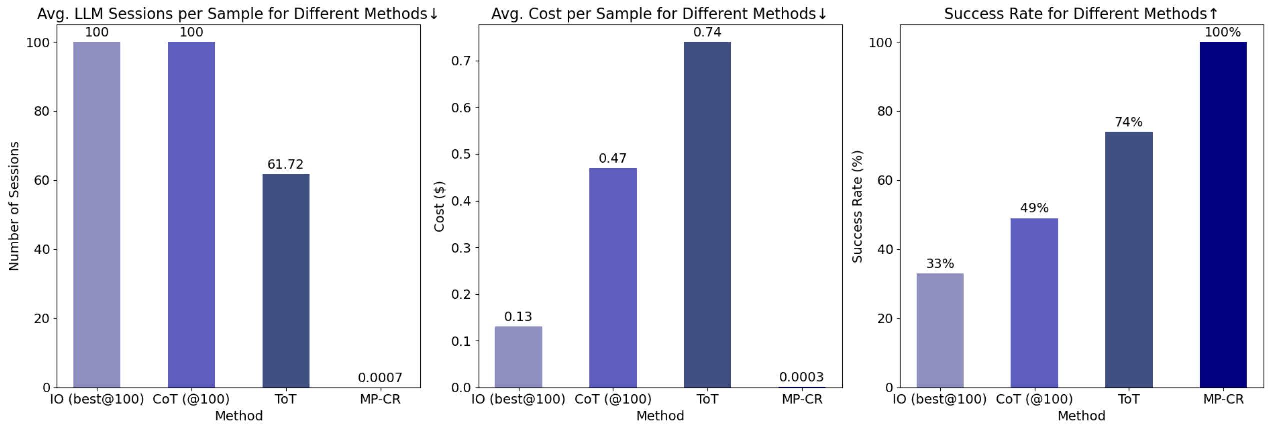


Figure 1: Comparative analysis of different methods (including IO (Direct), CoT [36], ToT [41], and one of our method MP-CR) for solving the Game of 24 tasks, illustrating the average LLM sessions per sample, the average cost per sample, and the overall success rate.

[17], 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) [36], Tree-of-Thought (ToT) [23, 41], and Cumulative Reasoning (CR) [43], 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 the theoretical framework of Meta Prompting (MP), a novel approach that represents a significant 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 the depth and nuance akin to human reasoning.

Furthermore, a pivotal aspect of Meta Prompting is its application to Meta Prompting for Prompting Tasks (MP-PT) in an in-context and recursive way utilizing the functorial and compositional properties of Meta Prompting. 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.

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.

The practical efficacy of the Meta Prompting framework is empirically validated through a series of experiments, prominently featuring the MP-CR Agent's application in diverse cognitive tasks. These experiments, ranging from solving the Game of 24 puzzles to addressing complex MATH problems, underscore the agent's versatility and advanced reasoning capabilities. Notably, the agent demonstrates remarkable proficiency in solving the Game of 24 tasks within one response for all samples, achieving a 100% success rate (as shown in Figure 1) with an average processing time of just 0.08 seconds per sample using OpenAI assistant API (once the structured program is written, only CPUs are involved). These results, far surpassing the performance of traditional methodologies, offer a compelling testament to the transformative impact of Meta Prompting in enhancing the problem-solving prowess of LLMs. As we navigate through the paper, we will delve into these experimental results in detail, elucidating the practical implications and potential of Meta Prompting in reshaping the landscape of AI-driven cognitive processes.

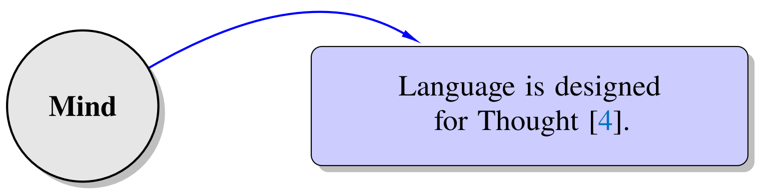


Figure 2: The concept of language as a medium and abstraction for thought, inspired by Chomsky's views, finds profound relevance in the domain of Meta Prompting. It emphasizes that language is more than just a tool for communication; it's also a structured medium that shapes and directs thought. In the context of AI and machine learning, this principle underpins the idea of Meta Prompting, where language is strategically used to create prompts that guide AI systems in understanding and solving complex tasks. This approach is rooted in the recognition that the way problems are linguistically framed can significantly influence an AI system's ability to process and respond to them, mirroring the human cognitive process of using language to structure and clarify thought.

In this paper, we will explore the principles and applications of Meta Prompting, illustrating how it leverages categorical and functorial concepts to transform the capabilities of LLMs. Our aim is to demonstrate how Meta Prompting stands as a groundbreaking development in AI, setting a new course for the evolution of intelligent, adaptable, and human-like reasoning systems in foundation models.

2 Preliminaries

2.1 Category

Definition 2.1 (Category)**.** A *category* $\mathscr{C}$ comprises a collection of *objects* and, for each pair of objects $A, B \in \mathscr{C}$, a set of *morphisms* (or arrows) from A to B, denoted as $\mathrm{Hom}(A, B)$. 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 $\mathscr{C}$, a morphism f from A to B is denoted by $f : A \to B$, where A is the source, and B is the target. It is assumed that $\mathrm{Hom}(A, B)$ is disjoint from $\mathrm{Hom}(A', B')$ unless $A = A'$ and $B = B'$.

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

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

2.2 Functors

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

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

This functor respects both identity morphisms ($\mathrm{F}(\mathrm{id}_A) = \mathrm{id}_{\mathrm{F}(A)}$) 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 $m : A_1 \to A_2$, 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 study 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 "λvariableName : type1.term" and has the type "type1 $\to$ 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

Meta Prompting is an advanced prompting technique inspired by type theory. This approach emphasizes the structural and syntactical aspects of examples, prioritizing the general format and pattern of a problem or topic over specific content details. It's an approach where the focus is on presenting the outline or framework of a problem or topic, offering a scaffold that can be filled with specific details as needed. This technique is particularly useful in contexts where understanding the format and pattern of a problem or solution is more critical than the intricacies of the content itself.

3.1 Characteristics of Meta Prompting

1. **Syntax-Oriented**: Meta Prompting gives priority to the form and structure of the prompt. Here, syntax serves as a guiding template, clearly delineating the expected structure of the response or solution. This approach is particularly beneficial in mathematical problem-solving where the structure of the solution is as important as the solution itself.

2. **Abstract-Example-Based**: This approach employs abstracted examples to illustrate the structure of problems and prompts. These examples, while not detailed in content, act as frameworks for inserting specific details. For instance, an abstract example in mathematics might show the general form of a proof without delving into specific numbers or functions.

3. **Type Theory Inspiration**: Drawing from type theory, Meta Prompting emphasizes the categorization of components in a prompt, such as problem statements, solution steps, or conclusions. It focuses on their logical arrangement and interrelationships, ensuring a coherent and structured approach to problem-solving.

4. **Adaptability**: Meta Prompting is versatile and can be adapted to diverse domains, from mathematical problem-solving to creative writing. Its strength lies in its ability to provide structured responses, essential for tackling a wide range of problems.

5. **Guidance for Detailed Exploration**: By providing a clear roadmap for structuring the approach to a topic, Meta Prompting sets the stage for in-depth exploration. It helps users navigate complex topics by focusing on the underlying structural patterns rather than getting lost in the specifics.

3.2 Distinction Between Meta Prompting and Few-Shot Prompting

To further clarify the unique roles and methodologies of Meta Prompting and Few-Shot Prompting, let us explore their differences in more detail (for a formal framework on Meta Prompting, please refer to Appendix A):

1. **Different Categories**:

    - *Meta Prompting* involves two distinct categories, $\mathcal{T}$ for tasks (problems) and $\mathcal{P}$ for structured prompts. The functor $\mathcal{M} : \mathcal{T} \to \mathcal{P}$ 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.

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

    When $\mathcal{M}(\cdot)$ is task-agnostic, it's simply 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. 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. 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 $\mathrm{LLM}(\mathcal{M}(\mathcal{T}_{\text{unsolved}}))$, 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 language models or AI systems equipped with external computational and physical environments.

$$\begin{tikzcd} T_{\text{unsolved}} \arrow[r, "\text{Solve the Task in Ideal}"] \arrow[d, "\mathcal{M}"'] & T_{\text{solved}} \arrow[d, "\mathcal{M}"] \\ P_{\text{unsolved}} \arrow[r, "\mathrm{LLM}(\mathcal{M}(T_{\text{unsolved}}))"'] & \boxed{P_{\text{solved}}} \end{tikzcd} \tag{2}$$

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, $\mathrm{LLM}(\mathcal{M}(\cdot))(\cdot)$ simplifies to $\mathrm{LLM}(\text{Meta Prompt})(\cdot)$, underscoring the adaptability and efficiency of the system in handling various tasks.

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

$$\mathrm{LLM}(\text{Meta Prompt}) : T_{\text{unsolved}} \to P_{\text{solved}}. \tag{4}$$

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 (see Section 6.2). 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.

$$\mathrm{LLM}(\mathcal{M}(T_{\text{unsolved}})) : T_{\text{unsolved}} \to P_{\text{solved}}. \tag{5}$$

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.

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

- *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. 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.

$$
\begin{aligned}
&\mathrm{LLM}(\mathrm{Q}_1 \to \mathrm{A}_1) : \mathrm{Q}_2 \to \mathrm{A}_2, \\
&\mathrm{LLM}(\mathrm{Q}_1 \to \mathrm{A}_1, \mathrm{Q}_2 \to \mathrm{A}_2) : \mathrm{Q}_3 \to \mathrm{A}_3, \\
&\quad \ldots
\end{aligned} \tag{6}
$$

2. **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 categorical framework. An example might be adapting the learning approach from a small set of image recognition tasks to a new, but similar, set of tasks.
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.
4. **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.

4 Meta Prompting for Complex Reasoning

Incorporating Meta Prompting within AI systems enhances their ability to interact with symbolic systems and code environments. By utilizing typed, structured prompts that are syntactically oriented, AI models can more effectively parse and interpret symbolic information. This is crucial in domains like mathematics or logic, where symbolic representation is key. 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 3 for an illustrative example.

```
<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 $\boxed{...}$.

</syntax>
```

Figure 3: Illustration of Meta Prompting for Complex Reasoning.

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**: Develops 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 '$\boxed{}$'.

4.1 Superiority of Meta Prompting over Few-Shot Examples

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 in token efficiency and the fairness of comparison in problem-solving scenarios.

Token Efficiency:

- **Reduced Token Usage**: 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.
- **Streamlined Problem Representation**: 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:

- **Level Playing Field**: Meta Prompting can be viewed as a form of zero-shot learning, where the influence of specific examples is minimized [2, 22, 31]. 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.
- **Reduced Example Bias**: 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.

5 Experiments

5.1 Solving the Game of 24 Tasks

Overview of the MP-CR Agent's capabilities. The MP-CR Agent, equipped with the MP-CR meta prompt (shown in Figure 13), represents a significant advancement in AI-driven problem-solving. This versatile agent is adept at handling a variety of complex reasoning tasks, including MATH problems [13] and coding challenges. In this subsection, we focus on its application in solving the Game of 24 task [41], 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.

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 4: Input prompt for solving the Game of 24 Tasks.

Results and Comparative Analysis. The agent achieved an impressive 100% success rate of all 1362 samples (shown in Table 1), 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 (only CPUs are involved once the program has been written). Figure 5 displays the Python program generated by the MP-CR Agent for solving these tasks all in one. 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.

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 [35], self-critical assessments [41, 43], reflective processes [32], and recursive (iterative) prompting techniques [43, 44] 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

Table 1: 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-Complex Reasoning (MP-CR), in solving the Game of 24 challenge. The 'LLM Sessions' column indicates the number of separate interactions or contexts with a Large Language Model 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, almost zero cost 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 [41]	61.72	5.5k / 1.4k	\$0.74	74%
MP-CR	$\frac{1}{N}$	$\approx \frac{1}{N}$ (8k / 1k)	$\approx$ **\$0.0003**	**100%**

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 (Hendryks et al., 2021) and more. The ability of the MP-CR Agent to handle diverse challenges not only marks a significant improvement over traditional methods but also opens new frontiers in AI research and applications.

5.2 Experiments on OpenAI Playground

6 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 [14, 31], 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 framework of General Meta Prompting with special tasks called prompting tasks, detailing its categorical and functorial properties (for formal definitions, please refer to Appendix B).

6.1 Examples of Meta Prompting for Prompting Tasks

The following examples demonstrate the application of the Meta Prompting framework in designing specific types of prompts. These examples will illustrate how the functor $\mathcal{M}_{\mathcal{T}}$ operates on objects in the category $\mathcal{T}$ and their corresponding structured prompts in $\mathcal{P}$.

Designing Better Reasoning Performance Prompts. Consider a subcategory $\mathcal{T}_{\text{enchance-reasoning}}$ 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.

Applying the functor $\mathcal{M}_{\mathcal{T}}$ to $\mathcal{T}_{\text{enchance-reasoning}}$, we obtain structured prompt category $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_{\text{enchance-reasoning}})$, 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.

```python
# 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 5: 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.

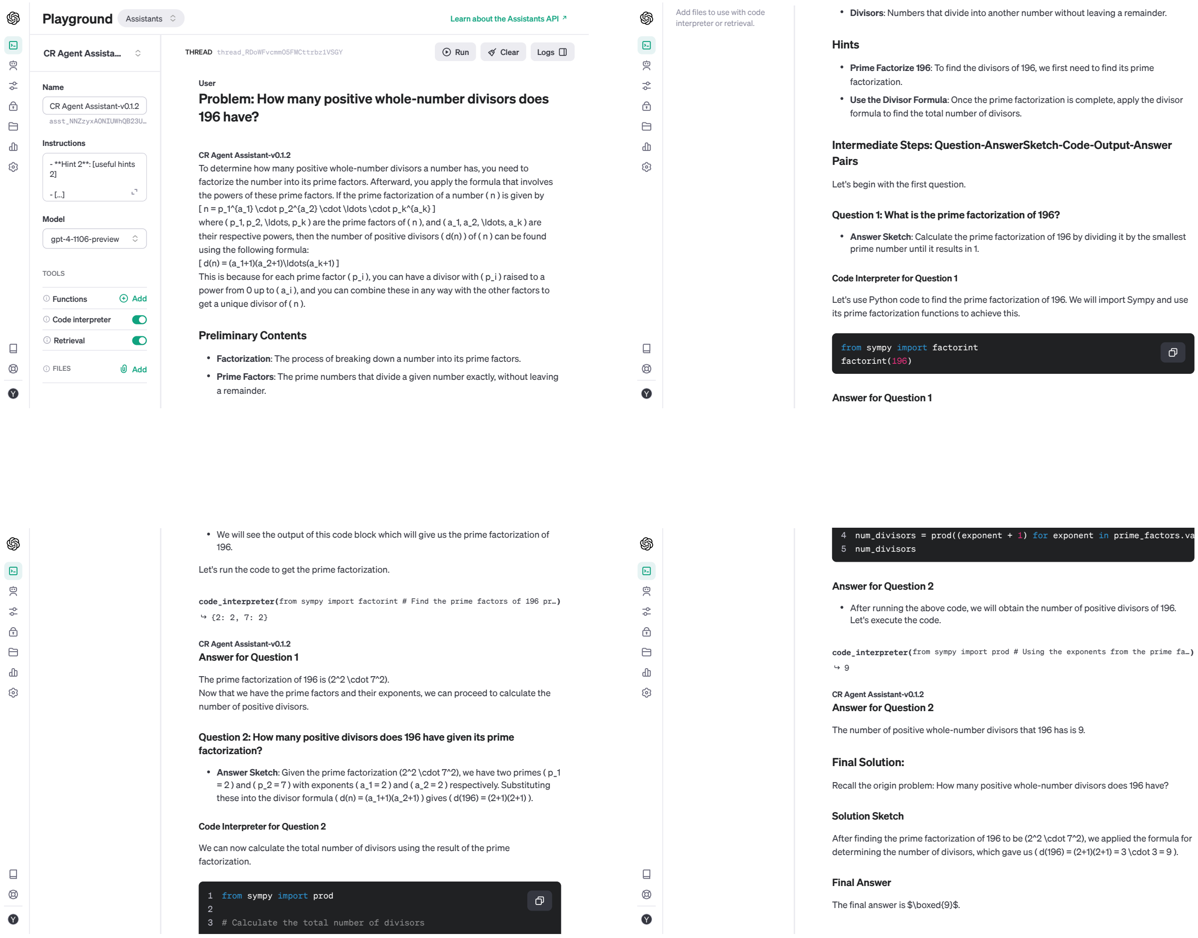


Figure 6: Experiment on MP-CR using OpenAI Assistant within the OpenAI Playground for solving a MATH [13] problem.

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

Designing Concise Prompts. 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.

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.

Task: *Prompt Simplification*

- **Original Prompt:** [input prompt]
- **Goal:** Transform the original prompt into a more concise version while preserving its core essence and objective.
- **Instructions for Transformation:**
  1. Maintain the primary purpose and objectives of the original prompt.
  2. Focus on distilling the prompt to include only key instructions and essential information.
  3. Eliminate any extraneous or non-essential details.
  4. Use clear, direct language to ensure ease of understanding.
  5. Where beneficial, employ bullet points or numbered steps to structure the prompt and enhance clarity.
- **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 8: Illustration of Meta Prompting for designing concise prompts.

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 objectives and requirements within the AI and machine learning context.

6.2 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. The primary role of Meta Prompting in this scenario shifts towards ensuring that tasks, which lack a consensus or clear definition among intelligent entities (including humans and AI), are represented in a prompt format that is easy to follow and precise. This approach not only enhances the clarity of communication between humans and AI but also fosters a more seamless integration of AI assistance in complex task-solving.

6.2.1 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.

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

The structured prompt illustrated in Figure 9 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 10. It highlights the dynamic and evolving nature of task definition and solution formulation in the realm of AI.

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. The potential for self-referential and recursive improvement [4] in AI task comprehension and prompt generation is particularly noteworthy, aligning with the broader goals of achieving more intelligent and autonomous AI system.

Note. One can call this process as *Meta Meta Prompting*, *Meta Meta Meta Prompting*, etc.

$$\mathrm{LLM}(\mathrm{LLM}(\text{Meta Meta Prompt})(T_{\text{intial}})) : T_{\text{intial}} \to T_{\text{revised}}. \tag{7}$$

$$\mathrm{LLM}(\mathrm{LLM}(\mathrm{LLM}(\text{Meta Meta Meta Prompt})(T_{\text{intial}}))) : T_{\text{intial}} \to T_{\text{revised}}. \tag{8}$$

7 Typed Meta Prompting for Multi-Modal Foundation Models

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

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

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

Task Interpretation:

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

Prompt Design:

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

Optional - Direct Solution Proposal:

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

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

Figure 10: 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 9.

modalities, including visual and auditory data, making it particularly effective in complex, real-world applications [15].

The syntactic and structured nature of 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.

7.1 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:

1. **Handling Diverse Data Types**: 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.

2. **Synchronization and Integration**: A critical aspect involves synchronizing and coherently integrating data from disparate modalities, ensuring a unified approach to problem-solving.

```
<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{}'
without units.</format>
<format>Use 'pi' and '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 11: System Instructions and Meta Prompt using XML format, which would be useful when aided by constrained generation framework such as guidance [24] and Langchain [5].

3. **Contextual Relevance**: It is imperative to maintain the contextual integrity of each modality, ensuring that they collectively contribute to a cohesive understanding of the task.

4. **Structural Integrity**: Adapting Meta Prompting to multi-modal data while preserving its core focus on structure and syntax is a significant undertaking, requiring the framework to be flexible yet robust.

```xml
<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 12: Generalize Meta Prompting into multi-modal settings.

Approach to Meta Prompting in Multi-Modal Scenarios. To effectively navigate the complexities of multi-modal data, Meta Prompting must evolve in several key areas:

1. **Adaptive Syntax Framework**: 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.

2. **Contextual Embedding**: 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.

3. **Integrated Analysis**: The system should be capable of intermodal analysis, drawing inferences by cross-referencing between text, images, sounds, or videos.

4. **Output Synthesis**: 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.

8 Related Work

Reasoning with AI systems. Historically, efforts in enhancing AI reasoning have centered on augmenting neural networks with the ability to generate intermediate steps, a paradigm widely acknowledged for boosting reasoning across diverse domains [12, 37, 38, 40, 42, 45]. These studies, while significant, often relied on content-focused enhancements. In contrast, our approach, Meta Prompting, diverges by emphasizing the structure and format of reasoning processes, representing a paradigm shift from content to form.

Symbolic systems such as code environments and knowledge graphs ([1, 3, 3, 7, 9–11, 16, 18, 21, 25, 26, 33, 34, 39]) have also been explored for reasoning enhancement. However, these approaches, while innovative, do not leverage the abstract categorical mappings central to Meta Prompting.

Chain-of-Thought Prompting and Its Evolution. Chain-of-thought reasoning proposed by Wei et al. [36] marked a significant advance, underscoring multi-step reasoning paths. This development, although groundbreaking, did not fully explore the syntactical and structural dimensions of reasoning that Meta Prompting captures. Following works, including Wang et al. [35]'s self-consistency strategy and Zhou et al. [45]'s complexity handling, introduced sophisticated decoding and problem-tackling strategies but remained within the scope of content-based reasoning enhancement.

The approaches by Li et al. [19], Yao et al. [41], Zheng et al. [44], and Feng et al. [8] exhibit a gradual shift towards more intricate reasoning strategies, yet they fall short of adopting the categorical and functorial perspectives integral to Meta Prompting. Zhang et al. [43]'s Cumulative Reasoning, while innovative in its use of multiple AI agents, does not focus on the structural and syntactical aspects that are the cornerstone of our method.

Prompt Programming and Metaprompt. The concept of prompt programming and metaprompt, as proposed by Reynolds & McDonell [31], marks a significant step towards system instruction via natural language, an approach that aligns with the foundation of Meta Prompting. However, their focus on un-typed natural language contrasts with our emphasis on typed language (including natural, programming, visual, and multi-modal languages) and structured syntax-oriented prompts, which bridges the gap between natural and programming languages and make language models can interact with peripheral computational and physical environment seamlessly. Metaprompting by Hou et al. [14] leverage model-agnostic meta-learning for prompt initialization and learning better prompts. However, it does not fully explore the potential of a functorial relationship between problems and prompts, especially the endo-functorial and monadic part (iterative and self-referential), an exploration at the heart of our work that may pave the way towards AI systems beyond human beings, or in short, AGI systems.

In summary, while previous works have significantly advanced the field of reasoning with AI, our approach, Meta Prompting, introduces a novel perspective that emphasizes the structural and syntactical aspects of problem-solving. By adopting a categorical and functorial approach, Meta Prompting transcends the limitations of content-focused enhancements, offering a more versatile and systematic methodology for enhancing AI reasoning.

9 Conclusion

Meta Prompting, as a methodological framework, stands as a testament to the innovative ways in which AI can be harnessed to enhance understanding and problem-solving across diverse domains. Its extension into multi-modal settings opens up new horizons, promising richer, more integrated approaches to data interaction and analysis. This journey into the depths of Meta Prompting underlines the dynamic and transformative potential of AI in shaping our approach to challenges, learning, and creativity in an increasingly complex world.

Future Outlook:

- **Evolving Applications**: As AI continues to advance, the application of Meta Prompting can be expected to evolve and expand, particularly in areas requiring complex problem-solving and data integration.
- **Technological Advancements**: With ongoing advancements in AI and machine learning, the capabilities of Meta Prompting, particularly in multi-modal settings, are likely to become more sophisticated and nuanced.
- **Communication Protocols**: Meta Prompting can even be generalized into the Communication Protocol used in Multi-Agent AI Systems, serving as the new TCP/IP in the era of AGI.

References

[1] Bauer, L., Wang, Y., and Bansal, M. Commonsense for generative multi-hop question answering tasks. *arXiv preprint arXiv:1809.06309*, 2018. 19

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 1, 9

[3] Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022. 19

[4] Chomsky, N. *On nature and language*. Cambridge University Press, 2002. 3, 15

[5] Contributors, L. Langchain: Building applications with llms through composability. `https://github.com/langchain-ai/langchain`, 2024. GitHub repository. 17

[6] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 1

[7] Ding, M., Zhou, C., Chen, Q., Yang, H., and Tang, J. Cognitive graph for multi-hop reading comprehension at scale. *arXiv preprint arXiv:1905.05460*, 2019. 19

[8] Feng, G., Gu, Y., Zhang, B., Ye, H., He, D., and Wang, L. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*, 2023. 19

[9] Feng, Y., Chen, X., Lin, B. Y., Wang, P., Yan, J., and Ren, X. Scalable multi-hop relational reasoning for knowledge-aware question answering. *arXiv preprint arXiv:2005.00646*, 2020. 19

[10] Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

[11] Gou, Z., Shao, Z., Gong, Y., Yang, Y., Huang, M., Duan, N., Chen, W., et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023. 19

[12] Hase, P. and Bansal, M. When can models learn from explanations? a formal framework for understanding the roles of explanation data. *arXiv preprint arXiv:2102.02201*, 2021. 19

[13] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021. 10, 13, 26

[14] Hou, Y., Dong, H., Wang, X., Li, B., and Che, W. Metaprompting: Learning to learn better prompts. *arXiv preprint arXiv:2209.11486*, 2022. 11, 19

[15] Hu, Y., Lin, F., Zhang, T., Yi, L., and Gao, Y. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*, 2023. 16

[16] Jiang, A. Q., Welleck, S., Zhou, J. P., Li, W., Liu, J., Jamnik, M., Lacroix, T., Wu, Y., and Lample, G. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *ArXiv*, abs/2210.12283, 2022. 19

[17] Kahneman, D. *Thinking, fast and slow*. macmillan, 2011. 2

[18] Kundu, S., Khot, T., Sabharwal, A., and Clark, P. Exploiting explicit paths for multi-hop reading comprehension. *arXiv preprint arXiv:1811.01127*, 2018. 19

[19] Li, Y., Lin, Z., Zhang, S., Fu, Q., Chen, B., Lou, J.-G., and Chen, W. 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. 19

[20] Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023. 1

[21] Lin, B. Y., Chen, X., Chen, J., and Ren, X. Kagnet: Knowledge-aware graph networks for commonsense reasoning. *arXiv preprint arXiv:1909.02151*, 2019. 19

[22] Liu, J., Gardner, M., Cohen, S. B., and Lapata, M. Multi-step inference for reasoning over paragraphs. *arXiv preprint arXiv:2004.02995*, 2020. 9

[23] Long, J. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023. 2

[24] Lundberg, S., Ribeiro, M. T. C., Viggiano, D., Rafael, J., Amemiya, R., and et. al. Microsoft guidance library. `https://github.com/microsoft/guidance`, 2023. 17

[25] Lyu, Q., Havaldar, S., Stein, A., Zhang, L., Rao, D., Wong, E., Apidianaki, M., and Callison-Burch, C. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023. 19

[26] Mihaylov, T. and Frank, A. Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge. *arXiv preprint arXiv:1805.07858*, 2018. 19

[27] OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023. 1

[28] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. Improving language understanding by generative pre-training. *openai.com*, 2018.

[29] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[30] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. 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

[31] Reynolds, L. and McDonell, K. 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. 9, 11, 19

[32] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K. R., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. 10

[33] Wang, X., Kapanipathi, P., Musa, R., Yu, M., Talamadupula, K., Abdelaziz, I., Chang, M., Fokoue, A., Makni, B., Mattei, N., 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. 19

[34] Wang, X., Liu, K., Wang, D., Wu, L., Fu, Y., and Xie, X. Multi-level recommendation reasoning over knowledge graphs with reinforcement learning. In *Proceedings of the ACM Web Conference 2022*, pp. 2098–2108, 2022. 19

[35] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022. 10, 19

[36] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022. 2, 19

[37] Wu, T., Terry, M., and Cai, C. J. 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. 19

[38] Yang, J., Jiang, H., Yin, Q., Zhang, D., Yin, B., and Yang, D. Seqzero: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. *arXiv preprint arXiv:2205.07381*, 2022. 19

[39] Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023. 19

[40] Yao, H., Chen, Y., Ye, Q., Jin, X., and Ren, X. Refining language models with compositional explanations. *Advances in Neural Information Processing Systems*, 34:8954–8967, 2021. 19

[41] Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023. 2, 10, 11, 19

[42] Zaidan, O., Eisner, J., and Piatko, C. 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. 19

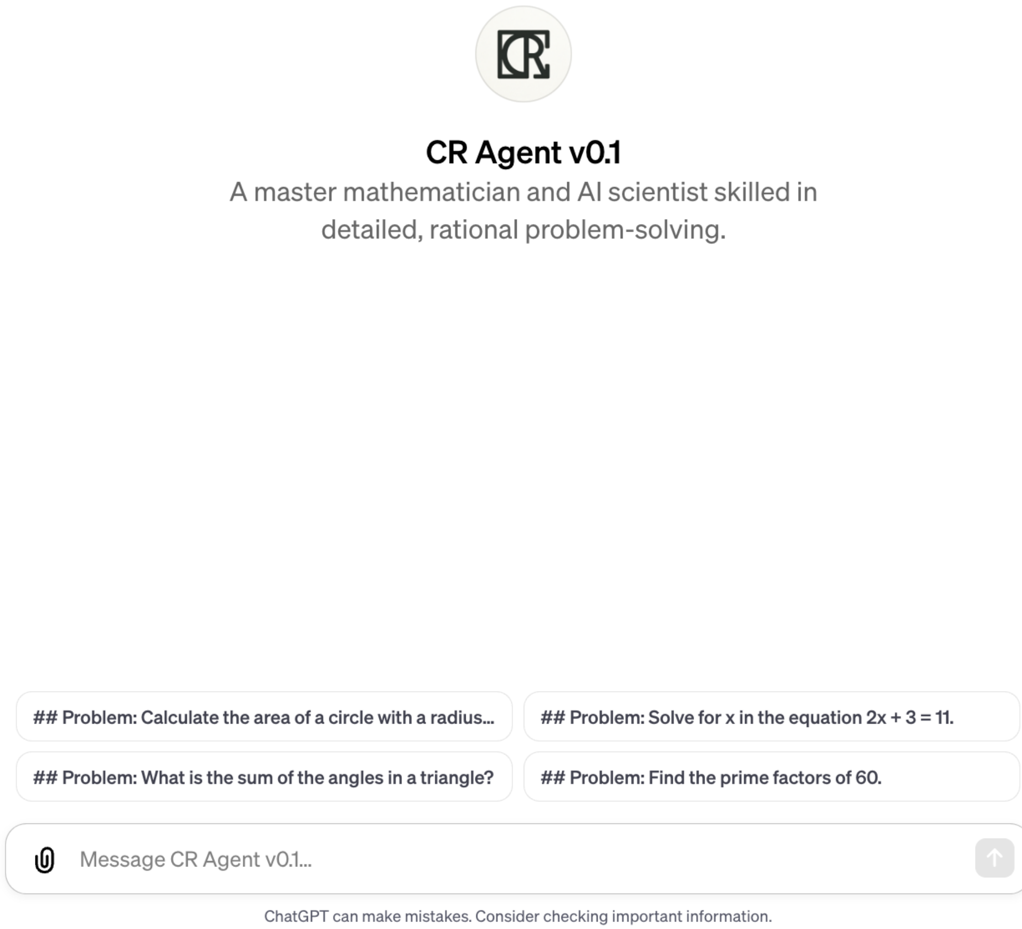
[43] Zhang, Y., Yang, J., Yuan, Y., and Yao, A. C.-C. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023. 2, 10, 19, 31

[44] Zheng, C., Liu, Z., Xie, E., Li, Z., and Li, Y. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*, 2023. 10, 19

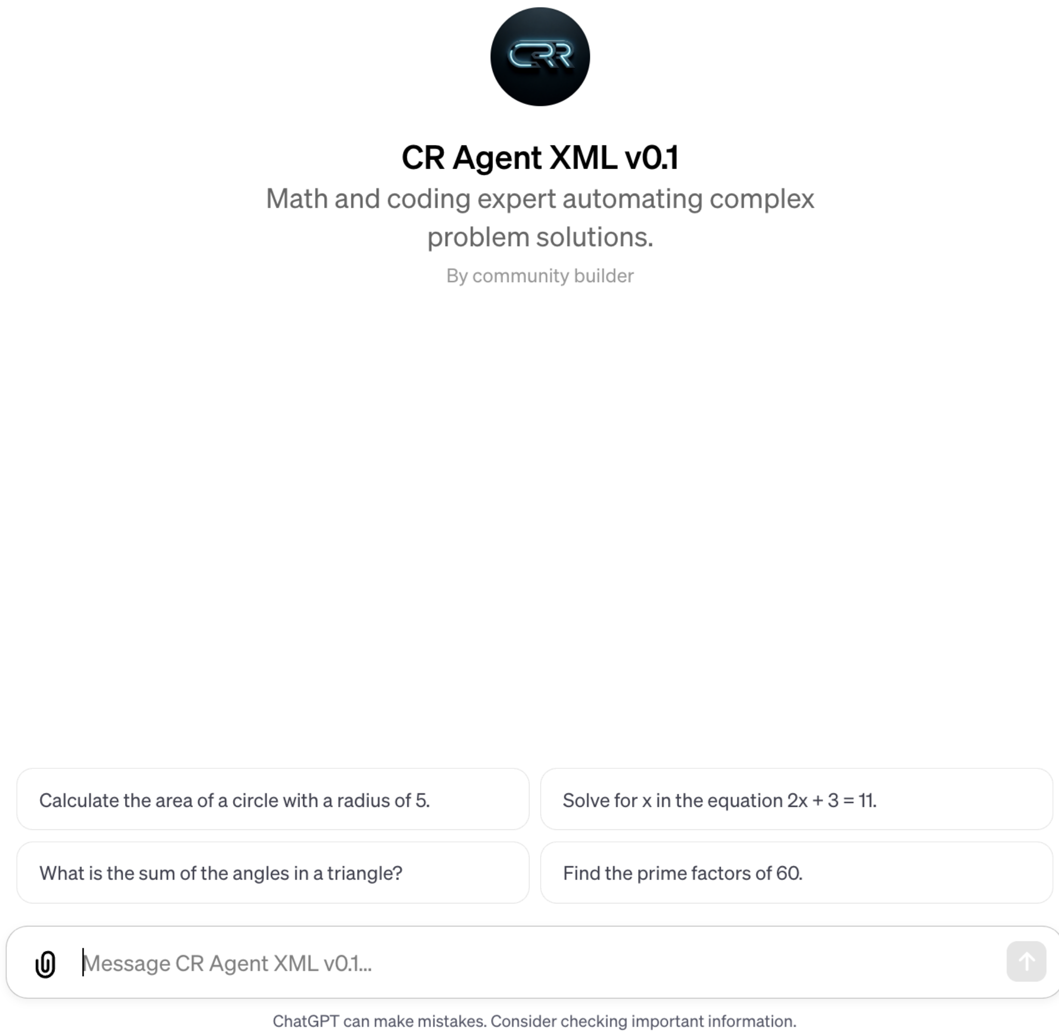
[45] Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., and Chi, E. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022. 19

Online Demo

Anyone interested in MP-CR can directly visit `https://chat.openai.com/g/g-L3a4ZCIHx-cr-agent-v0-1` for an online demo.



Anyone interested in MP-CR-XML can directly visit `https://chat.openai.com/g/g-4ir4la2Z6-cr-agent-xml-v0-1` (or `https://anonymous.4open.science/r/meta-prompting-anonymous` for all the rest prompts presented in this paper) for an online demo.



```
<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{}'
without units.</format>
<format>Use 'pi' and '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 13: 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).

```
User: ## Problem: "Define\n\\[p = \\sum_{k = 1}^\\infty \\frac{1}{k^2} \\quad \\text{and} \\quad
q = \\sum_{k = 1}^\\infty \\frac{1}{k^3}.\\]Find a way to write\n\\[\\sum_{j = 1}^\\infty
\\sum_{k = 1}^\\infty \\frac{1}{(j + k)^3}\\]in terms of $p$ and $q.$

---

Assistant: ### Solution Approach

To solve the problem, we need to express the given double series \(\sum_{j = 1}^{\infty}
\sum_{k = 1}^{\infty} \frac{1}{(j + k)^3}\) in terms of \( p \) and \( q \), where

\[ p = \sum_{k = 1}^{\infty} \frac{1}{k^2} \quad \text{and} \quad q
= \sum_{k = 1}^{\infty} \frac{1}{k^3}. \]

#### Preliminary Analysis
1. **Understanding \( p \) and \( q \)**: The series for \( p \) and \( q \) are well-known in
mathematics, representing the Riemann zeta function at 2 and 3, respectively. However, their exact
values may not be necessary for this problem.

2. **Analyzing the Double Series**: The series \(\sum_{j = 1}^{\infty} \sum_{k = 1}^{\infty}
\frac{1}{(j + k)^3}\) can be simplified by changing the order of summation or by transforming it
into a single series.

#### Hints
- Consider the symmetries and patterns that emerge when expanding the double series.
- Think about how the terms of \( p \) and \( q \) might appear within the double series.

#### Intermediate Steps
1. **Expanding the Double Series**: Look at the initial terms of the series to identify any
pattern.
2. **Transforming the Series**: Try to transform the double series into a form that
resembles \( p \) and \( q \).

#### Solving the Problem
1. Expand and analyze the first few terms of the double series.
2. Explore transformations or manipulations that can simplify the series into a form
involving \( p \) and \( q \).

Let's start by analyzing the double series to see if we can identify a pattern.

The first few terms of the double series \(\sum_{j = 1}^{\infty} \sum_{k = 1}^{\infty}
\frac{1}{(j + k)^3}\) are as follows:

- For \(j = 1, k = 1\): \(\frac{1}{(1+1)^3} = \frac{1}{8}\)
- For \(j = 1, k = 2\): \(\frac{1}{(1+2)^3} = \frac{1}{27}\)
- For \(j = 2, k = 1\): \(\frac{1}{(2+1)^3} = \frac{1}{27}\)
- For \(j = 2, k = 2\): \(\frac{1}{(2+2)^3} = \frac{1}{64}\)

We observe that each term in the series can be represented as \(\frac{1}{n^3}\) where \(n\)
is the sum of \(j\) and \(k\). This suggests a connection with the series for \(q\), which is
a sum of terms of the form \(\frac{1}{k^3}\).

Next, we will attempt to transform the double series into a single series and relate it to \(q\).

'''python
# Importing necessary library
from sympy import *

# Defining variables
j, k = symbols('j k')

# Expanding the first few terms of the double series
# We consider the terms for j = 1,2 and k = 1,2 for simplicity
double_series_expansion = [[1/(j+k)**3 for k in range(1, 3)] for j in range(1, 3)]

double_series_expansion
'''
'''Result
[[0.125, 0.037037037037037035], [0.037037037037037035, 0.015625]]
'''

The transformation of the double series \(\sum_{j = 1}^{\infty} \sum_{k = 1}^{\infty}
\frac{1}{(j + k)^3}\) into a single series and simplifying it for the first few terms (up to \(j,
k = 4\)) gives us an approximate value of \(0.2996\).
```

Figure 14: Example of MP-CR-XML Agent v0.2 on solving a level 5 problem in MATH [13] dataset, the ground-truth answer is $p - q$ (the agent correctly answered this problem).

````
However, this numerical value does not immediately suggest a clear relationship with \(p\) and
\(q\). To better understand this relationship, let's consider the structure of the double series
more closely. Each term in the series \(\frac{1}{(j + k)^3}\) can be seen as a term of
\(\frac{1}{n^3}\) for some \(n\). We need to count how many times each \(n\) appears in
the series and relate it to the series for \(q = \sum_{k = 1}^{\infty} \frac{1}{k^3}\).

Let's analyze the structure of the series further to find this relationship.
```python
# Transforming the double series into a single series
# We use the fact that each term in the double series can be written as 1/n^3
# where n = j + k

# Defining the transformed series
transformed_series = sum(1/(j+k)**3 for j in range(1, 5) for k in range(1, 5))

# Simplifying the transformed series
simplified_series = simplify(transformed_series)

simplified_series
```
```Result
0.299621991753050
```

The analysis of the structure of the series shows the count of occurrences of each \(n = j + k\)
in the series for the first few values of \(n\):

- For \(n = 2\): There is \(1\) term (\(j = 1, k = 1\)).
- For \(n = 3\): There are \(2\) terms (\(j = 1, k = 2\) and \(j = 2, k = 1\)).
- For \(n = 4\): There are \(3\) terms.
- For \(n = 5\): There are \(4\) terms.
- For \(n = 6\): There are \(5\) terms.

We observe a pattern: for each \(n\), there are \(n - 1\) terms of the form \(\frac{1}{n^3}\) in
the double series. This pattern suggests that the double series is a sum of
terms \(\frac{n - 1}{n^3}\).

The next step is to express this pattern in terms of the series for \(q\) and investigate
any relation to \(p\). Let's proceed with this approach.

```python
# Analyzing the structure of the series
# We count how many times each n = j + k appears for the first few values of n

# Let's consider n up to 6 for illustration
n_values = range(2, 7)  # Since the minimum value of j+k is 2
n_counts = {n: sum(1 for j in range(1, n) for k in range(1, n) if j + k == n) for n in n_values}

n_counts
```
```Result
{2: 1, 3: 2, 4: 3, 5: 4, 6: 5}
```

It seems there was an error due to the variable \( n \) not being defined. Let me correct this
and proceed with the transformation and simplification of the double series.

```python
# Expressing the double series in terms of the series for q
# We use the observation that each term of the form 1/n^3 appears (n-1) times

# Defining q as a symbolic series
q_series = Sum(1/k**3, (k, 1, oo))

# Transforming the double series
transformed_double_series = Sum((n-1)/n**3, (n, 2, oo))

# Simplifying the transformed series
simplified_transformed_series = simplify(transformed_double_series)

simplified_transformed_series, q_series, q_series.doit()
```

The transformed double series is expressed as \(\sum_{n=2}^{\infty}
\frac{1 - \frac{1}{n}}{n^2}\).
This series closely resembles the series for \(q = \sum_{k=1}^{\infty} k^{-3}\), which is known
to converge to \(\zeta(3)\), where \(\zeta\) is the Riemann zeta function.
````

Figure 15: Continued from Figure 14.

~~~~
The transformed double series is expressed as \(\sum_{n=2}^{\infty}
\frac{1 - \frac{1}{n}}{n^2}\).
This series closely resembles the series for \(q = \sum_{k=1}^{\infty} k^{-3}\), which is known
to converge to \(\zeta(3)\), where \(\zeta\) is the Riemann zeta function.

To find a relation in terms of \(p\) and \(q\), we need to express the transformed series
using the series definitions of \(p\) and \(q\).

- \(p = \sum_{k=1}^{\infty} \frac{1}{k^2}\)
- \(q = \sum_{k=1}^{\infty} \frac{1}{k^3} = \zeta(3)\)

Our next step is to express the transformed series in terms of \(p\) and \(q\). Let's proceed to
analyze this further.

```python
# Defining the variable n
n = symbols('n')

# Redefining the transformed double series with the correct variable
transformed_double_series = Sum((n-1)/n**3, (n, 2, oo))

# Simplifying the transformed series
simplified_transformed_series = simplify(transformed_double_series)

simplified_transformed_series, q_series, q_series.doit()
```
```Result
(Sum((1 - 1/n)/n**2, (n, 2, oo)), Sum(k**(-3), (k, 1, oo)), zeta(3))
```

There was an oversight in defining the symbolic representation of \( p \) before using it in the
calculation. Let's define \( p \) correctly and then express the transformed series in terms of
\( p \) and \( q \).

```python
# Let's break down the transformed series into parts that resemble p and q

# The series can be split into two parts: Sum(1/n^2, (n, 2, oo)) - Sum(1/n^3, (n, 2, oo))
part1 = Sum(1/n**2, (n, 2, oo))
part2 = Sum(1/n**3, (n, 2, oo))

# Simplifying each part
simplified_part1 = simplify(part1)
simplified_part2 = simplify(part2)

# Expressing in terms of p and q
# p = Sum(1/k^2, (k, 1, oo)), but we need to adjust for the starting index of 2
adjusted_p = p - 1  # Subtracting the first term of p (1/1^2)

# q is already in the correct form
adjusted_q = q

# The transformed series in terms of p and q
transformed_series_in_terms_of_p_q = adjusted_p - adjusted_q

simplified_part1, simplified_part2, transformed_series_in_terms_of_p_q
```

The simplified expression for the transformed double series in terms of \( p \) and \( q \)
is given as \(-\zeta(3) + \frac{\pi^2}{6}\). Here, \( p \) and \( q \) are defined as follows:

- \( p = \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \) (known from the Basel problem),
- \( q = \sum_{k=1}^{\infty} \frac{1}{k^3} = \zeta(3) \) (where \(\zeta\) is
the Riemann zeta function).

Thus, the series \(\sum_{j = 1}^{\infty} \sum_{k = 1}^{\infty} \frac{1}{(j + k)^3}\) can
be expressed in terms of \( p \) and \( q \) as follows:

\[
\sum_{j = 1}^{\infty} \sum_{k = 1}^{\infty} \frac{1}{(j + k)^3} = -\zeta(3) + \frac{\pi^2}{6}
= -q + p
\]

Therefore, the final expression in terms of \( p \) and \( q \) is:

\[
\boxed{-q + p}
\]
~~~~

Figure 16: Continued from Figure 15.

```
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 17: System Instructions used in MP-CR, the actual context would be [SystemInstruction] + [Structured-MetaPrompt].

Categorical Framework for Meta Prompting

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 $\mathrm{Hom}_{\mathcal{T}}(X, Y)$, 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 $\mathrm{Hom}_{\mathcal{P}}(U, V)$, 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 $\mathcal{M} : \mathcal{T} \to \mathcal{P}$, 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 $f : X \to Y$ 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 $\mathcal{M}(f) : \mathcal{M}(X) \to \mathcal{M}(Y)$ 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 $f : X \to Y$ and $g : Y \to Z$ in $\mathcal{T}$, we have $\mathcal{M}(g \circ f) = \mathcal{M}(g) \circ \mathcal{M}(f)$. Also, for each object X in $\mathcal{T}$, $\mathcal{M}(\mathrm{id}_X) = \mathrm{id}_{\mathcal{M}(X)}$.

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 Formal Definitions on Meta Prompting and Few-Shot Prompting

Meta Prompting as a Functor. We have defined Meta Prompting as a functor $\mathcal{M} : \mathcal{T} \to \mathcal{P}$. 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 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 $\mathcal{P}$.

$$\begin{tikzcd} T_{\text{unsolved}} \arrow[r, "f"] \arrow[d, "\mathcal{M}"'] & T_{\text{solved}} \arrow[d, "\mathcal{M}"] \\ P_{\text{unsolved}} \arrow[r, "g"'] & P_{\text{solved}} \end{tikzcd} \tag{9}$$

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 $f : T_{\text{unsolved}} \to T_{\text{solved}}$ signifies the ideal process of solving a task, notice that those step-by-step decomposition and composition methods like Cumulative Reasoning (CR) [43] which obey the composition law in the $\mathcal{T}$ category are natural.
- The morphism $g : P_{\text{unsolved}} \to P_{\text{solved}}$ ($g = \mathcal{M}(f)$) 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 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).

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, $\mathrm{Hom}_{\mathcal{F}}(X, Y)$, 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.

B Formal Definitions 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 $\mathrm{Hom}_{\mathcal{T}}(\mathcal{T}_i, \mathcal{T}_j)$, 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 $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{P}$, 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 $f : \mathcal{T}_i \to \mathcal{T}_j$ in $\mathcal{T}$, representing a method or adaptation in the task domain, the functor $\mathcal{M}_{\mathcal{T}}$ assigns a corresponding morphism $\mathcal{M}_{\mathcal{T}}(f) : \mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) \to \mathcal{M}_{\mathcal{T}}(\mathcal{T}_j)$ 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 $f : \mathcal{T}_i \to \mathcal{T}_j$ and $g : \mathcal{T}_j \to \mathcal{T}_k$ 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 $\mathcal{T}$, $\mathcal{M}_{\mathcal{T}}(\mathrm{id}_{\mathcal{T}_i}) = \mathrm{id}_{\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i)}$.

$$\begin{tikzcd}
\mathcal{T}_i \arrow[r, "f"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"'] & \mathcal{T}_j \arrow[d, "\mathcal{M}_{\mathcal{T}}"] \\
\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) \arrow[r, "\mathcal{M}_{\mathcal{T}}(f)"'] & \mathcal{M}_{\mathcal{T}}(\mathcal{T}_j)
\end{tikzcd} \tag{10}$$

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:

$$
\begin{tikzcd}
T_{\text{unsolved}} \arrow[r, "\text{Revise}"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"'] & T_{\text{revised}} \arrow[d, "\mathcal{M}_{\mathcal{T}}"] \\
P_{\text{unsolved}} \arrow[r, "\text{LLM}(\mathcal{M}_{\mathcal{T}_{\text{Revise}}}(P_{\text{unsolved}}))"'] & P_{\text{revised}}
\end{tikzcd}
\tag{11}
$$

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:

$$
\begin{tikzcd}
T_{\text{unsolved}} \arrow[r, "\text{Revise and Solve}"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"'] & T_{\text{solved}} \arrow[d, "\mathcal{M}_{\mathcal{T}}"] \\
P_{\text{unsolved}} \arrow[r, "\text{LLM}(\mathcal{M}_{\mathcal{T}_{\text{Revise and Solve}}}(P_{\text{unsolved}}))"'] & \boxed{P_{\text{solved}}}
\end{tikzcd}
\tag{12}
$$

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 $F : \mathcal{C} \to \mathcal{C}$. 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 $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{T}$, 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 $f : \mathcal{T}_i \to \mathcal{T}_j$ and $g : \mathcal{T}_j \to \mathcal{T}_k$, $\mathcal{M}_{\mathcal{T}}(g \circ f) = \mathcal{M}_{\mathcal{T}}(g) \circ \mathcal{M}_{\mathcal{T}}(f)$.

The following commutative diagram represents the endofunctor in Meta Prompting.

$$
\begin{tikzcd}
\mathcal{T}_i \arrow[r, "f"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"'] & \mathcal{T}_j \arrow[r, "g"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"] & \mathcal{T}_k \arrow[d, "\mathcal{M}_{\mathcal{T}}"] \\
\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) \arrow[r, "\mathcal{M}_{\mathcal{T}}(f)"'] & \mathcal{M}_{\mathcal{T}}(\mathcal{T}_j) \arrow[r, "\mathcal{M}_{\mathcal{T}}(g)"'] & \mathcal{M}_{\mathcal{T}}(\mathcal{T}_k)
\end{tikzcd}
$$

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 $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{T}$ maps tasks within the same category.

- **Unit Transformation** (η): The natural transformation $\eta : \mathrm{Id}_{\mathcal{T}} \Rightarrow \mathcal{M}_{\mathcal{T}}$ 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*: $\mu \circ \mathcal{M}_{\mathcal{T}}\eta = \mathrm{id}_{\mathcal{M}_{\mathcal{T}}}$, ensuring the basic structure is maintained when a task is first structured and then unstructured.
  - *Right Identity*: $\mu \circ \eta\mathcal{M}_{\mathcal{T}} = \mathrm{id}_{\mathcal{M}_{\mathcal{T}}}$, 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.

$$\begin{tikzcd}
\mathcal{M}_{\mathcal{T}} \arrow[r, "\eta\mathcal{M}_{\mathcal{T}}"] \arrow[dr, "\mathrm{id}_{\mathcal{M}_{\mathcal{T}}}"'] & \mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}} \arrow[d, "\mu"] \\
 & \mathcal{M}_{\mathcal{T}}
\end{tikzcd}$$

$$\begin{tikzcd}
\mathcal{M}_{\mathcal{T}} \arrow[r, "\mathcal{M}_{\mathcal{T}}\eta"] \arrow[dr, "\mathrm{id}_{\mathcal{M}_{\mathcal{T}}}"'] & \mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}} \arrow[d, "\mu"] \\
 & \mathcal{M}_{\mathcal{T}}
\end{tikzcd}$$

$$\begin{tikzcd}
\mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}} \arrow[r, "\mu\mathcal{M}_{\mathcal{T}}"] \arrow[d, "\mathcal{M}_{\mathcal{T}}\mu"'] & \mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}} \arrow[d, "\mu"] \\
\mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}} \arrow[r, "\mu"'] & \mathcal{M}_{\mathcal{T}}
\end{tikzcd}$$

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.

$$
\begin{tikzcd}
T_{\text{initial}} \arrow[r, "f"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"] & T_{\text{intermediate}} \arrow[r, "g"] \arrow[d, "\mathcal{M}_{\mathcal{T}}"] & T_{\text{final}} \arrow[d, "\mathcal{M}_{\mathcal{T}}"] \\
\mathcal{M}_{\mathcal{T}}(T_{\text{initial}}) \arrow[r, "\mathcal{M}_{\mathcal{T}}(f)"'] \arrow[d, "\mathcal{M}_{\mathcal{T}}"] & \mathcal{M}_{\mathcal{T}}(T_{\text{intermediate}}) \arrow[r, "\mathcal{M}_{\mathcal{T}}(g)"'] \arrow[d, "\mathcal{M}_{\mathcal{T}}"] & \mathcal{M}_{\mathcal{T}}(T_{\text{final}}) \arrow[d, "\mathcal{M}_{\mathcal{T}}"] \\
\mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(T_{\text{initial}})) \arrow[r, "\mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(f))"'] & \mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(T_{\text{intermediate}})) \arrow[r, "\mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(g))"'] & \mathcal{M}_{\mathcal{T}}(\mathcal{M}_{\mathcal{T}}(T_{\text{final}}))
\end{tikzcd}
\tag{13}
$$

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.