Meta Prompting for AI Systems

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

[1]IIIS, Tsinghua University
[2]Shanghai Qi Zhi Institute

zhangyif21@mails.tsinghua.edu.cn,
{yuanyang, andrewcyao}@tsinghua.edu.cn

Abstract

We introduce *Meta Prompting* (MP), a prompting paradigm designed to enhance the utilization of large language models (LLMs) and AI systems in complex problem-solving and data interaction. Grounded in type theory and category theory, Meta Prompting prioritizes structural and syntactical considerations over traditional content-centric methods. In this work, we formally define Meta Prompting, delineate its distinctions from few-shot prompting, and demonstrate its effectiveness across various AI applications. In particular, we show that Meta Prompting can decompose intricate reasoning tasks into simpler sub-problems, thereby improving token efficiency and enabling fairer comparisons with conventional few-shot techniques. Furthermore, we extend this framework to prompting tasks, allowing LLMs to recursively self-generate refined prompts in a metaprogramming-like manner. Empirical evaluations reveal that a Qwen-72B base language model equipped with Meta Prompting—without additional instruction tuning—achieves a PASS@1 accuracy of 46.3% on MATH problems, surpassing a supervised fine-tuned counterpart, 83.5% accuracy on GSM8K, and a 100% success rate on Game of 24 tasks using GPT-4. The code is available at https://github.com/meta-prompting/meta-prompting.

1 Introduction

The advent of foundation models, particularly Large Language Models (LLMs), has transformed the field of artificial intelligence. With extensive training data and robust generalization capabilities, these models have significantly broadened the horizons of computational linguistics, text understanding, and text generation (Devlin et al., 2018; Radford et al., 2018, 2019; Brown et al., 2020; Raffel et al., 2020; OpenAI, 2023). Despite these advances, LLMs still exhibit limitations when addressing complex reasoning tasks—especially those demanding deep, abstract thought such as advanced mathematics (Lightman et al., 2023). This observation underscores the need for methodologies that enhance the reasoning faculties of LLMs.

A core challenge originates from the auto-regressive token prediction architecture that underpins modern LLMs (Radford et al., 2018, 2019; Brown et al., 2020). Although this design efficiently handles a broad range of tasks, it is not optimally configured for the depth and sophistication

---

[†]Corresponding authors.

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

Figure 1: A structure meta prompt presented in JSON format.

of human-like analytical reasoning. This discrepancy is aptly captured by the dual-process theory of cognitive psychology (Kahneman, 2011), which distinguishes between the rapid, intuitive responses of System 1 and the deliberate, systematic processes of System 2. In their standard operation, LLMs tend to emulate System 1 processes, thereby struggling with tasks that require the more structured approach of System 2 thinking.

In recent years, approaches such as Chain-of-Thought (CoT) (Wei et al., 2022) and Tree-of-Thought (ToT) (Yao et al., 2023; Long, 2023) prompting have been proposed to guide LLMs in articulating intermediate reasoning steps. While these methods have contributed to improved performance, they do not fully capture the breadth and flexibility of human cognitive processes.

In response to these challenges, we introduce *Meta Prompting* (MP) and establish a comprehensive theoretical framework for it. Meta Prompting extends beyond traditional content-driven techniques by emphasizing a structure-oriented perspective. Drawing inspiration from category theory and type theory, our method establishes a functorial relationship between tasks and their corresponding prompts. This categorical framework yields a systematic and adaptable approach that is capable of addressing a wide range of cognitive tasks with nuance and depth similar to human reasoning.

A pivotal innovation of our work is the application of Meta Prompting to prompting tasks in an in-context and recursive manner—a concept we term *Recursive Meta Prompting* (RMP). Analogous to metaprogramming in programming language theory, RMP enables LLMs to autonomously generate and refine prompts. This self-referential capability not only allows LLMs to solve problems but also to construct the structures necessary for their own reasoning processes, marking a significant leap in model autonomy and adaptability.

The practical efficacy of the Meta Prompting framework is empirically validated through a series of experiments. These include solving the Game of 24 puzzles (Yao et al., 2023) and addressing complex MATH problems (Hendrycks et al., 2021), thereby demonstrating the versatility and enhanced reasoning capabilities enabled by our approach.

In summary, our contributions are as follows:
- We propose a structured, syntax-oriented Meta Prompting (MP) framework and develop its

theoretical foundation based on category theory. Furthermore, we investigate Meta Prompting for prompting tasks and introduce Recursive Meta Prompting (RMP) in a metaprogramming-like manner.

- Through experiments, we demonstrate that a Qwen-72B base language model equipped with Meta Prompting—without additional instruction tuning—achieves a PASS@1 accuracy of 46.3% on MATH problems, surpassing both fine-tuned models and even the initial version of GPT-4. Additionally, our approach attains 83.5% accuracy on GSM8K and a 100% success rate on the Game of 24 tasks, thereby underscoring its efficacy in problem-solving and in-context alignment.

Problem Statement:

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

Solution Structure:

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

---

Figure 2: A structured meta prompt presented in markdown format for solving MATH (Hendrycks et al., 2021) problems, as introduced in the Minerva study by (Lewkowycz et al., 2022).

Problem: Find the domain of the expression $\frac{\sqrt{x-2}}{\sqrt{5-x}}$.
Solution: The expressions inside each square root must be non-negative. Therefore, $x - 2 \geq 0$, so $x \geq 2$, and $5 - x \geq 0$, so $x \leq 5$. Also, the denominator cannot be equal to zero, so $5 - x > 0$, which gives $x < 5$. Therefore, the domain of the expression is $\boxed{[2,5)}$. Final Answer: The final answer is $[2,5)$. I hope it is correct.
————-
Problem: If $\det \mathbf{A} = 2$ and $\det \mathbf{B} = 12$, then find $\det(\mathbf{AB})$.
Solution: We have that $\det(\mathbf{AB}) = (\det \mathbf{A})(\det \mathbf{B}) = (2)(12) = \boxed{24}$. Final Answer: The final answer is 24. I hope it is correct.
————-
...

Figure 3: An example of the most widely used few-shot prompt for solving MATH problems.

2 Background

2.1 Category Theory

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.

Definition 2.5 (Isomorphism and Automorphism). An *isomorphism* between two objects A and B in a category is a morphism $f : A \to B$ that has a unique inverse $g : B \to A$, such that $f \circ g$ and $g \circ f$ are identity morphisms on B and A, respectively. An *automorphism* is an isomorphism of an object with itself, revealing the self-similarity or symmetry of the object.

2.2 Functors

Definition 2.6 (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 ($F(m_2 \circ m_1) = F(m_2) \circ F(m_1)$).

- **Faithfulness and Fullness**: A covariant functor $F : \mathscr{A} \to \mathscr{B}$ is *faithful* if it injectively maps morphisms between any two objects in $\mathscr{A}$, and *full* if this mapping is surjective. A functor is *fully faithful* if it is both full and faithful.

- **Subcategories**: A subcategory $\mathscr{A}'$ of $\mathscr{A}$ is *full* if the inclusion functor $i : \mathscr{A}' \to \mathscr{A}$ is full.

Definition 2.7 (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, $F(m_2 \circ m_1) = F(m_1) \circ F(m_2)$.

3 Meta Prompting

Meta Prompting is a prompting technique that emphasizes the structural and syntactical aspects of problems by prioritizing the overall format and pattern over specific content details. This method constructs an abstract and structured approach to interacting with large language models (LLMs), placing emphasis on the form and syntax of information. Such an approach is particularly effective in scenarios where recognizing the underlying framework of a problem is crucial for its resolution.

Definition 3.1 (Meta Prompt). A *Meta Prompt* is an example-agnostic structured prompt designed to capture the reasoning structure of a specific category of tasks. It provides a scaffold that outlines the general approach to a problem, thereby enabling LLMs to fill in task-specific details as needed. This methodology focuses on the procedural aspects of problem-solving—the *how*—rather than the content-specific details—the *what*.

3.1 Formalizing Meta Prompting

In category theory, a functor F from a category $\mathcal{C}$ to a category $\mathcal{D}$, denoted $F : \mathcal{C} \to \mathcal{D}$, maps objects and morphisms (arrows) from $\mathcal{C}$ to $\mathcal{D}$ in a manner that preserves the categorical structure (i.e., identity morphisms and composition of morphisms).

Applying this concept to Meta Prompting, we define two categories:

Definition 3.2 (Categories of Tasks and Prompts). Let $\mathcal{T}$ denote a category whose objects are various tasks or problems (e.g., mathematical problems, coding challenges, or theoretical queries). The morphisms in $\mathcal{T}$, denoted $\mathrm{Hom}_{\mathcal{T}}(X, Y)$, represent the methodologies or transformations that relate one problem X to another Y (for instance, transforming a linear algebra problem into an optimization problem).

Similarly, let $\mathcal{P}$ denote a category whose objects are structured prompts designed to guide the solution of these tasks. Objects in $\mathcal{P}$ comprise carefully crafted prompts—such as a step-by-step guide for solving a differential equation or a template for writing code—while the morphisms, denoted $\mathrm{Hom}_{\mathcal{P}}(U, V)$, represent the adaptations or refinements of one prompt U into another V (e.g., adapting a prompt for a basic algebra problem to one suited for a complex calculus problem).

The core of our framework is the *Meta Prompting Functor*:

Definition 3.3 (Meta Prompting Functor). Define the *Meta Prompting Functor* $\mathcal{M} : \mathcal{T} \to \mathcal{P}$ as follows:

- **On Objects:** For each task $X \in \mathcal{T}$, assign a corresponding structured prompt $\mathcal{M}(X) \in \mathcal{P}$. For example, if X is a quadratic equation problem, then $\mathcal{M}(X)$ may be a prompt outlining the necessary steps to solve quadratic equations.

- **On Morphisms:** For each morphism $f : X \to Y$ in $\mathcal{T}$, which represents a transformation or method for solving task X in terms of task Y, assign a morphism $\mathcal{M}(f) : \mathcal{M}(X) \to \mathcal{M}(Y)$ in $\mathcal{P}$. For instance, if f transforms a basic algebra task into an advanced algebra problem, then $\mathcal{M}(f)$ adapts the corresponding prompt accordingly.

This functor preserves the categorical structure; that is, for any $f : X \to Y$ and $g : Y \to Z$ in $\mathcal{T}$,

$$\mathcal{M}(g \circ f) = \mathcal{M}(g) \circ \mathcal{M}(f),$$

and for every object $X \in \mathcal{T}$,

$$\mathcal{M}(\mathrm{id}_X) = \mathrm{id}_{\mathcal{M}(X)}.$$

Meta Prompting thus provides a systematic method for constructing prompts tailored to specific task categories. This approach ensures that a language model equipped with the appropriate prompt accurately captures the task's objectives and executes the solution process as intended. Its adaptability further allows for effective application even when the task category is not naturally representable in the language (textual, visual, or programming) of the model.

This mapping can be hand-crafted by a human or generated using LLMs via a recursive, self-composing method (see Section 4.2). Furthermore, morphisms in $\mathcal{T}$ (representing transformations between tasks) are mapped to corresponding morphisms in $\mathcal{P}$ (representing transformations between prompts) in such a way that the structure and logic of problem-solving are preserved.

Example of Meta Prompting. Consider the task of solving a quadratic equation, represented as an object $Q \in \mathcal{T}$. The Meta Prompting functor $\mathcal{M}$ maps Q to a structured prompt $P \in \mathcal{P}$ (see Figure 4), which might outline steps such as identifying coefficients, computing the discriminant, and applying the quadratic formula. This structured prompt guides the LLM in solving any quadratic equation by following a systematic approach, regardless of the specific coefficients involved.

A structure meta prompt for solving quadratic equations in the form $ax^2 + bx + c = 0$:
{
    "Problem": "Solve the quadratic equation $ax^2 + bx + c = 0$ for x.",
    "Solution": {
        "Step 1": "Identify the coefficients a, b, and c from the equation.",
        "Step 2": "Compute the discriminant using $\Delta = b^2 - 4ac$.",
        "Step 3": "Determine the nature of the roots by checking if $\Delta > 0$, $\Delta = 0$, or $\Delta < 0$.",
        "Step 4": "If $\Delta > 0$, calculate the two distinct real roots using $x_{1,2} = \frac{-b \pm \sqrt{\Delta}}{2a}$.",
        "Step 5": "If $\Delta = 0$, calculate the single real root using $x = \frac{-b}{2a}$.",
        "Step 6": "If $\Delta < 0$, calculate the complex roots using $x_{1,2} = \frac{-b \pm i\sqrt{|\Delta|}}{2a}$.",
        "Step 7": "Conclude by summarizing the roots in a LaTeX-formatted box, $\boxed{...}$."
        },
    "Final Answer": "Depending on the value of Δ, the final answer is provided by $x_{1,2}$." }
————

Figure 4: A structured meta prompt in JSON format for solving quadratic equations.

Characteristics of Meta Prompting. Meta Prompting prioritizes form and structure over content by employing a syntactical template that guides the expected response or solution. It utilizes abstract examples to illustrate the overall structure of problems and solutions, without focusing

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

Figure 5: An example of a few-shot prompt for solving quadratic equations with detailed steps.

on specific details. Drawing from type theory, Meta Prompting emphasizes categorizing prompt components—such as problem statements, solution steps, and conclusions—and arranging them logically to ensure a coherent problem-solving process. This versatile technique is applicable across various domains, offering a clear roadmap for navigating complex topics.

3.2   Distinctions between Meta Prompting and Few-Shot Prompting

Meta Prompting differs from Few-Shot Prompting in both its methodology and objectives. Few-shot prompting provides LLMs with a limited set of example problems and their corresponding solutions to facilitate learning from these instances. In contrast, Meta Prompting abstracts the overall problem-solving process by creating prompts that guide the model through logical steps without relying on specific content-based examples.

For further illustration of these differences, please refer to Figures 1, 2, and 3, and consult Appendix A.3 for additional discussion.

3.3   Meta Prompting for Complex Reasoning

Integrating Meta Prompting within AI systems enhances their capability to interact with symbolic systems and code environments. By utilizing typed, structured prompts, AI models can more effectively parse and interpret symbolic information—a crucial advantage in fields such as mathematics and logic. Moreover, the structured nature of these prompts aligns seamlessly with code environments, enabling AI agents to understand, modify, and execute code across both textual and visual programming paradigms. This broadened interaction fosters a more comprehensive understanding across various domains.

This specialized application of Meta Prompting for complex reasoning is tailored to address intricate, multi-layered problems that demand profound analytical and logical processing. (For an illustrative example, see Figure 8 in Appendix C.)

3.4 Advantages of Meta Prompting

Meta Prompting offers distinct benefits over traditional few-shot approaches, particularly for large language models (LLMs). Two primary advantages are:

Token Efficiency. By emphasizing structure over exhaustive content, Meta Prompting significantly reduces the number of tokens required. This efficiency is vital in contexts where token limits are imposed. The focus on syntax ensures a concise yet clear representation of problems.

Fair Comparison and Zero-Shot Efficacy. Meta Prompting can be regarded as a form of zero-shot prompting, wherein the influence of specific examples is minimized (Brown et al., 2020; Liu et al., 2020; Reynolds & McDonell, 2021). This approach enables a more equitable comparison among different problem-solving models by avoiding reliance on example-based learning and specific prior knowledge. Consequently, the LLM can approach problems with a fresh, unbiased perspective, unencumbered by the limitations of few-shot examples.

In summary, Meta Prompting is distinguished by its token efficiency and its ability to provide a fair, unbiased approach to problem-solving, making it especially valuable in settings where token economy and equitable model comparisons are critical.

4 Meta Prompting for Prompting Tasks

While the general Meta Prompting framework provides a robust method for guiding reasoning, its application to the generation and refinement of prompts themselves introduces a higher level of autonomy and self-improvement. *Meta Prompting for Prompting Tasks* (MP-PT) focuses on the automatic creation and recursive refinement of prompts that steer subsequent task execution.

4.1 Specialized Framework for Prompt Generation

In MP-PT, the objective is to generate refined prompt templates that are optimally structured to address a wide array of downstream tasks. To formalize this process, we consider two specialized categories:

- $\mathcal{T}$: The category of prompting tasks, where objects are initial (possibly suboptimal) prompt formulations, and morphisms represent the revisions or transformations applied to improve these prompts.

- $\mathcal{P}$: The category of refined prompt templates, where objects are structured prompts designed to optimally guide the LLM in solving the underlying tasks.

The Meta Prompting Functor in this context, $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{P}$, maps an initial prompt to a refined version that better encapsulates the desired reasoning structure. For any morphism $f : T_1 \to T_2$ in $\mathcal{T}$, the functor assigns $\mathcal{M}_{\mathcal{T}}(f) : \mathcal{M}_{\mathcal{T}}(T_1) \to \mathcal{M}_{\mathcal{T}}(T_2)$, thus preserving the transformations inherent in the prompt refinement process.

4.2 Recursive Refinement and Self-Improvement

A salient feature of MP-PT is its capacity for recursive self-improvement. In this paradigm, the functor $\mathcal{M}_{\mathcal{T}}$ can be viewed as an endofunctor on $\mathcal{T}$: $\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{T}$.

This endofunctorial perspective enables an iterative process wherein an initial, unsolved prompt T_{unsolved} is successively refined and then finally solved by LLM:

$$\text{LLM}(\mathcal{M}_{\mathcal{T}}(T_{\text{unsolved}})) : T_{\text{unsolved}} \to T_{\text{solved}}.$$

The recursive mechanism, which can be called *Meta Meta Prompting*, empowers the LLM to autonomously generate a sequence of improved prompts that are increasingly tailored to the task at hand. This process benefits from dynamic inference and lazy evaluation, ensuring that refinements occur only as needed and that token efficiency is maintained.
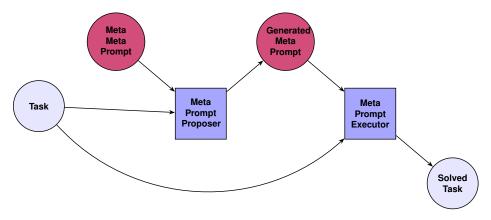

Figure 6: Recursive Meta Prompting: A single recursion step.

By automating the prompt-generation process, MP-PT enhances the adaptability and autonomy of AI systems. The refined prompts not only improve reasoning performance but also facilitate a more modular and compositional approach to problem-solving. The recursive refinement mechanism ensures that the LLM continually evolves its own guiding structures. The system iteratively adjusts its prompts to better suit the task environment, promoting continuous improvement. In addition, recursive meta prompting allows the LLM to seamlessly transition between tasks by reusing and adapting refined prompt templates.

5 Experiments

In this section, we rigorously evaluate the performance of our proposed Meta Prompting (MP) framework on several challenging mathematical benchmarks and problem-solving tasks. Our experiments are designed to assess both accuracy and efficiency, thereby demonstrating the advantages of our structure-oriented approach over traditional few-shot prompting techniques.

5.1 Solving MATH and GSM8K Problems

Experimental Setup. We evaluate our method on two widely recognized benchmarks for mathematical problem-solving. The MATH dataset (Hendrycks et al., 2021) comprises 5000 competition-level math word problems, while GSM8K (Cobbe et al., 2021) contains 1319 grade school math problems. In our experiments, we perform inference using the vLLM framework on the Qwen-14B and Qwen-72B base language models. For the MATH dataset, we employ the Meta Prompt depicted in Figure 2; for the GSM8K dataset, we use the Meta Prompt shown in Figure 1.

To evaluate the model outputs, we implement a carefully designed rule-based evaluator that includes a SymPy (Meurer et al., 2017) comparison to determine if the generated response aligns with the ground-truth solution. The detailed evaluation results are provided in Table 1 and Table 2.

Experimental Results. Our evaluation confirms that the zero-shot meta-prompted Qwen-72B base language model achieves superior performance on both the MATH and GSM8K benchmarks. Notably, the meta prompt elicits strong instruction-following capabilities in sufficiently large base models solely through in-context learning. On the MATH dataset, our approach attains a ground-breaking PASS@1 accuracy of 46.3%, surpassing both open-source models and proprietary systems such as GPT-4 (2023-0314), which registers 42.5%. On GSM8K, the zero-shot meta-prompted Qwen-72B model achieves an accuracy of 83.5%, outperforming the best few-shot prompting approaches as well as fine-tuned counterparts. These results underscore the transformative potential of Meta Prompting in enhancing reasoning and problem-solving abilities in large language models without the need for extensive fine-tuning or specialized training datasets. Furthermore, the significant gains in token efficiency and the fairness of model comparisons highlight the benefits of our structural and syntax-oriented methodology.

Table 1: Comparative analysis of PASS@1 accuracy on the MATH benchmark for various models without tool usage (e.g., code interpreter). This comparison underscores the notable improvements achieved by our zero-shot meta-prompted base language models over existing approaches in mathematical problem-solving.

Model	FT-Dataset	Tool Usage	Eval Method	MATH (%)
Proprietary Models				
Claude-2 (Anthropic, 2023)	-	No	CoT	32.5
Minerva-540B (Lewkowycz et al., 2022)	Arxiv+Web	No	CoT	33.6
PaLM-2 (Anil et al., 2023)	-	No	CoT	34.3
GPT-4 (2023-0314) (OpenAI, 2023)	-	No	CoT	42.5
Open-source Models				
Llama-2-70B (base)	-	No	CoT	13.5
Qwen-14B (base)	-	No	CoT	<u>24.8</u>
Qwen-14B (base)	-	No	**MP**	**28.9**
Qwen-72B (base)	-	No	CoT	<u>35.2</u>
Qwen-72B-MetaMathQA	MetaMathQA	No	CoT	41.7
Qwen-72B (base)	-	No	**MP**	**46.3**

5.2 Solving the Game of 24 Tasks

Comparative Analysis. Table 3 presents a comprehensive comparison of several methodologies for solving the Game of 24 challenge, including Input/Output (IO), Chain-of-Thought (CoT), Tree-of-Thought (ToT) (Yao et al., 2023), and our proposed Meta Prompting (MP). The table details the number of LLM sessions required, the tokens generated for prompting, the incurred cost per case, and the corresponding success rates. Notably, the MP-CR method is distinguished by its minimal requirement for LLM sessions (effectively $\frac{1}{N}$ per sample), drastically reduced token generation,

Table 2: Comparative analysis of PASS@1 accuracy on the GSM8K benchmark for various open-source large language models without tool usage (e.g., code interpreter). Our zero-shot meta prompting approach shows substantial improvements over few-shot CoT prompting.

Model	FT-Dataset	Tool Usage	Eval Method	GSM8K (%)
Llama-2-70B (base) (Touvron et al., 2023)	-	No	CoT	56.8
Qwen-14B (base) (Bai et al., 2023)	-	No	CoT	61.3
Qwen-14B (base)	-	No	**MP**	**64.8**
WizardMath-70B (Luo et al., 2023)	WizardMath	No	CoT	81.6
MetaMath-70B (Yu et al., 2023)	MetaMathQA	No	CoT	82.3
Qwen-72B (base)	-	No	CoT	78.9
Qwen-72B (base)	-	No	**MP**	**83.5**

and a success rate of 100% (with $N = 1362$ samples).

MP-CR Agent Evaluation. The MP-CR Agent, equipped with the MP-CR meta prompt (illustrated in Figure 19 in Appendix C), is capable of addressing a diverse range of complex reasoning tasks, including both mathematical problems and coding challenges. In this set of experiments, we focus on the Game of 24 task (Yao et al., 2023)—a classic puzzle that requires combining four numbers with arithmetic operations (+, -, *, /) to form an expression that evaluates to 24.

Experimental Setup. Our experimental design demonstrates the MP-CR Agent's capability to autonomously generate Python code for solving Game of 24 tasks. In contrast to traditional iterative and time-intensive methods, the MP-CR Agent processes all samples within a single response, thereby significantly enhancing computational efficiency and reducing overhead.

Results and Discussion. The MP-CR Agent achieved a 100% success rate on all 1362 Game of 24 samples (see Table 3), indicating its robust ability to solve these puzzles accurately. Moreover, the average processing time was only 0.08 seconds per sample using the OpenAI assistant API. Figure 17 in Appendix D.1 presents the Python program generated by the MP-CR Agent, which effectively translates the mathematical principles of the Game of 24 into executable code. For additional experimental details, please refer to Appendix D.1.

Overall, these experiments illustrate that Meta Prompting not only enhances the reasoning capabilities of large language models but also offers substantial improvements in token efficiency and fairness of evaluation compared to traditional few-shot methods.

Table 3: Comparative analysis of methods for the Game of 24 Tasks. The table compares various approaches including IO, CoT, ToT, and Meta Prompting (MP) in terms of LLM sessions, token generation, cost per case, and success rate. The MP-CR method is highlighted for its efficiency and 100% success rate.

Method	LLM Sessions	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 (Yao et al., 2023)	61.72	5.5k / 1.4k	$0.74	74%
MP	$\frac{1}{N}$	$\approx \frac{1}{N}$ (8k / 1k)	$\approx$ **$0.0003**	**100%**

6 Related Work

Reasoning with AI Systems. Efforts to enhance AI reasoning capabilities have largely focused on equipping neural networks with mechanisms to generate intermediate reasoning steps, a strategy that has yielded improvements across diverse domains (Zaidan et al., 2007; Yao et al., 2021; Hase & Bansal, 2021; Yang et al., 2022; Wu et al., 2022; Zhou et al., 2022). Although these approaches have advanced the state of the art, they predominantly emphasize content-driven reasoning. In parallel, substantial research has investigated the use of symbolic systems—such as code environments and knowledge graphs—to further augment reasoning (Mihaylov & Frank, 2018; Bauer et al., 2018; Kundu et al., 2018; Wang et al., 2019; Lin et al., 2019; Ding et al., 2019; Feng et al., 2020; Wang et al., 2022a; Chen et al., 2022; Lyu et al., 2023; Chen et al., 2022; Gao et al., 2023; Gou et al., 2023; Jiang et al., 2022; Yang et al., 2023). In contrast, our work on meta prompting shifts the focus from content-centric methods to a structural and formal treatment of reasoning processes.

Chain-of-Thought Prompting. The introduction of Chain-of-Thought (CoT) prompting by Wei et al. (2022) marked a significant milestone by emphasizing the articulation of intermediate reasoning steps. Building on this foundation, subsequent methodologies—such as the self-consistency approach of Wang et al. (2022b) and the decomposition strategy proposed by Zhou et al. (2022) have further refined techniques for multi-step problem solving. Additionally, metaprompting methods introduced by Hou et al. (2022) leverage model-agnostic meta-learning for prompt initialization and improved soft prompt representations. More recent developments by Li et al. (2023), Yao et al. (2023), Feng et al. (2023), and Zhang et al. (2023) have progressively advanced reasoning strategies. Nonetheless, these approaches generally do not emphasize the structural and syntactical aspects of multi-layered reasoning—particularly in zero-shot, example-agnostic settings involving interactions with peripheral environments—which are foundational to our meta prompting methodology.

7 Conclusion

In this work, we introduced Meta Prompting—a framework that shifts focus from content-specific details to the structural and patterned aspects of problems and solutions. By emphasizing a systematic, format-centric approach, Meta Prompting enables large language models to transcend their inherent limitations and achieve more sophisticated and adaptable reasoning. This methodology enhances performance in complex problem-solving scenarios, providing a robust framework for structured and systematic cognitive processing.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. Palm 2 technical report.

arXiv preprint arXiv:2305.10403, 2023.

Anthropic. Model card and evaluations for claude models. *Anthropic*, 2023.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

Lisa Bauer, Yicheng Wang, and Mohit Bansal. Commonsense for generative multi-hop question answering tasks. *arXiv preprint arXiv:1809.06309*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

LangChain Contributors. Langchain: Building applications with llms through composability. https://github.com/langchain-ai/langchain, 2024. GitHub repository.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Ming Ding, Chang Zhou, Qibin Chen, Hongxia Yang, and Jie Tang. Cognitive graph for multi-hop reading comprehension at scale. *arXiv preprint arXiv:1905.05460*, 2019.

Guhao Feng, Yuntian Gu, Bohang Zhang, Haotian Ye, Di He, and Liwei Wang. Towards revealing the mystery behind chain of thought: a theoretical perspective. *arXiv preprint arXiv:2305.15408*, 2023.

Yanlin Feng, Xinyue Chen, Bill Yuchen Lin, Peifeng Wang, Jun Yan, and Xiang Ren. Scalable multi-hop relational reasoning for knowledge-aware question answering. *arXiv preprint arXiv:2005.00646*, 2020.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International Conference on Machine Learning*, pp. 10764–10799. PMLR, 2023.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*, 2023.

Peter Hase and Mohit Bansal. When can models learn from explanations? a formal framework for understanding the roles of explanation data. *arXiv preprint arXiv:2102.02201*, 2021.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

Yutai Hou, Hongyuan Dong, Xinghao Wang, Bohan Li, and Wanxiang Che. Metaprompting: Learning to learn better prompts. *arXiv preprint arXiv:2209.11486*, 2022.

Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning. *arXiv preprint arXiv:2311.17842*, 2023.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. *ArXiv*, abs/2210.12283, 2022.

Daniel Kahneman. *Thinking, fast and slow*. macmillan, 2011.

Souvik Kundu, Tushar Khot, Ashish Sabharwal, and Peter Clark. Exploiting explicit paths for multi-hop reading comprehension. *arXiv preprint arXiv:1811.01127*, 2018.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857, 2022.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5315–5333, 2023.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. Kagnet: Knowledge-aware graph networks for commonsense reasoning. *arXiv preprint arXiv:1909.02151*, 2019.

Jiangming Liu, Matt Gardner, Shay B Cohen, and Mirella Lapata. Multi-step inference for reasoning over paragraphs. *arXiv preprint arXiv:2004.02995*, 2020.

Jieyi Long. Large language model guided tree-of-thought. *arXiv preprint arXiv:2305.08291*, 2023.

Scott Lundberg, Marco Tulio Correia Ribeiro, David Viggiano, Joao Rafael, Riya Amemiya, and et. al. Microsoft guidance library. https://github.com/microsoft/guidance, 2023.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. Faithful chain-of-thought reasoning. *arXiv preprint arXiv:2301.13379*, 2023.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL https://doi.org/10.7717/peerj-cs.103.

Todor Mihaylov and Anette Frank. Knowledgeable reader: Enhancing cloze-style reading comprehension with external commonsense knowledge. *arXiv preprint arXiv:1805.07858*, 2018.

OpenAI. Gpt-4 technical report. *ArXiv*, abs/2303.08774, 2023.

Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. *openai.com*, 2018.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551, 2020.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

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

Xiaoyan Wang, Pavan Kapanipathi, Ryan Musa, Mo Yu, Kartik Talamadupula, Ibrahim Abdelaziz, Maria Chang, Achille Fokoue, Bassem Makni, Nicholas Mattei, et al. Improving natural language inference using external knowledge in the science questions domain. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 7208–7215, 2019.

Xiting Wang, Kunpeng Liu, Dongjie Wang, Le Wu, Yanjie Fu, and Xing Xie. Multi-level recommendation reasoning over knowledge graphs with reinforcement learning. In *Proceedings of the ACM Web Conference 2022*, pp. 2098–2108, 2022a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022b.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Tongshuang Wu, Michael Terry, and Carrie Jun Cai. Ai chains: Transparent and controllable human-ai interaction by chaining large language model prompts. In *Proceedings of the 2022 CHI conference on human factors in computing systems*, pp. 1–22, 2022.

Jingfeng Yang, Haoming Jiang, Qingyu Yin, Danqing Zhang, Bing Yin, and Diyi Yang. Seqzero: Few-shot compositional semantic parsing with sequential prompts and zero-shot models. *arXiv preprint arXiv:2205.07381*, 2022.

Kaiyu Yang, Aidan M Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *arXiv preprint arXiv:2306.15626*, 2023.

Huihan Yao, Ying Chen, Qinyuan Ye, Xisen Jin, and Xiang Ren. Refining language models with compositional explanations. *Advances in Neural Information Processing Systems*, 34:8954–8967, 2021.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

Omar Zaidan, Jason Eisner, and Christine Piatko. Using "annotator rationales" to improve machine learning for text categorization. In *Human language technologies 2007: The conference of the North American chapter of the association for computational linguistics; proceedings of the main conference*, pp. 260–267, 2007.

Yifan Zhang, Jingqin Yang, Yang Yuan, and Andrew Chi-Chih Yao. Cumulative reasoning with large language models. *arXiv preprint arXiv:2308.04371*, 2023.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*, 2022.

A More On Meta Prompting

A.1 Natural Transformations

Definition A.1 (Natural Transformation). A *natural transformation* between two covariant functors $F, G : \mathscr{A} \to \mathscr{B}$ is a family of morphisms $\{m_A : F(A) \to G(A)\}_{A \in \mathscr{A}}$ such that for every morphism $f : A \to A'$ in $\mathscr{A}$ the diagram

$$
\begin{array}{ccc}
F(A) & \xrightarrow{F(f)} & F(A') \\
{\scriptstyle m_A}\downarrow & & \downarrow{\scriptstyle m_{A'}} \\
G(A) & \xrightarrow[G(f)]{} & G(A')
\end{array}
$$

commutes. When each m_A is an isomorphism, the transformation is said to be a *natural isomorphism*.

Definition A.2 (Equivalence of Categories). Two categories $\mathscr{A}$ and $\mathscr{B}$ are *equivalent* if there exist functors $F : \mathscr{A} \to \mathscr{B}$ and $F' : \mathscr{B} \to \mathscr{A}$ such that the compositions $F \circ F'$ and $F' \circ F$ are naturally isomorphic to the identity functors on $\mathscr{B}$ and $\mathscr{A}$, respectively.

A.2 Type Theory

Type theory provides a rigorous framework for describing type systems in mathematics, logic, and computer science and is often proposed as an alternative foundation to set theory. Early examples include Church's typed λ-calculus and Martin-Löf's intuitionistic type theory; these theories underpin many modern proof assistants such as Coq and Lean via systems like the Calculus of Inductive Constructions.

In type theory, every term is associated with a type, typically denoted as

$$\text{term : type.}$$

Common types include the natural numbers (denoted by $\mathbb{N}$ or `nat`) and Boolean values (`bool`). Terms are constructed through function applications, and computation is formalized as the systematic rewriting of a term's syntax.

A key construct is the lambda abstraction: a term of the form

$$\lambda \, \text{variable} : \text{type}_1. \, \text{term}$$

has the type $\text{type}_1 \to \text{type}_2$, representing a function that maps an input of type type_1 to an output of type type_2.

Type theory diverges from classical set theory in several respects:

- Classical set theory adheres to the law of excluded middle, while type theory—aligned with intuitionistic logic—does not necessarily do so.

- Whereas elements in set theory may belong to multiple sets, in type theory each term is typically associated with a unique type.

- Computation is intrinsic to type theory: expressions such as `1+1` and `2` are syntactically distinct yet evaluate to the same value.

- Numbers are encoded more naturally as inductive types, aligning closely with Peano's axioms, rather than as arbitrary sets.

A.3 Meta Prompting and Few-Shot Prompting

Meta Prompting as a Functor.   Recall that Meta Prompting is formalized as the functor $\mathcal{M} : \mathcal{T} \to \mathcal{P}$, where $\mathcal{T}$ is the category of tasks and $\mathcal{P}$ is the category of structured prompts. This functor systematically associates each task in $\mathcal{T}$ with a corresponding structured prompt in $\mathcal{P}$, and similarly maps each method of transformation (morphism) in $\mathcal{T}$ to a corresponding transformation in $\mathcal{P}$. This association preserves both the compositional structure and the identity elements, embodying the fundamental nature of a functor.

The following commutative diagram illustrates this functorial relationship:

$$
\begin{array}{ccc}
T_{\text{unsolved}} & \xrightarrow{\ f\ } & T_{\text{solved}} \\
{\scriptstyle \mathcal{M}}\downarrow & & \downarrow{\scriptstyle \mathcal{M}} \\
P_{\text{unsolved}} & \xrightarrow[\ g\ ]{} & P_{\text{solved}}
\end{array}
\tag{A.1}
$$

Here:

- T_{unsolved} and T_{solved} denote, respectively, an unsolved task and its solved counterpart in $\mathcal{T}$.

- P_{unsolved} and P_{solved} denote an unsolved structured prompt and its refined version in $\mathcal{P}$.

- The functor $\mathcal{M}$ maps tasks to corresponding structured prompts, preserving the integrity of the solution process.

- The morphism $f : T_{\text{unsolved}} \to T_{\text{solved}}$ represents an ideal solution process (e.g., a step-by-step decomposition as in Cumulative Reasoning (Zhang et al., 2023)).

- The induced morphism $g : P_{\text{unsolved}} \to P_{\text{solved}}$ (with $g = \mathcal{M}(f)$) captures the transformation of an unsolved prompt into a solved one—effectively translating syntactic structure into semantic content.

Few-Shot Prompting in Category Theory.   Few-shot prompting can be conceptualized by introducing a specialized category $\mathcal{F}$, which encapsulates few-shot learning scenarios. In $\mathcal{F}$, objects represent distinct few-shot tasks (each with a limited set of examples), and the morphisms $\text{Hom}_{\mathcal{F}}(X, Y)$ model the transformation or generalization from one such task X to another Y.

Few-Shot Prompting in Type Theory.   Within the framework of type theory, few-shot prompting is formalized via types and terms. Each example in a few-shot setting corresponds to a term of a particular type, representing an individual problem instance, and its solution is a term of another type. Thus, the few-shot prompting process can be viewed as a mapping between these terms, analogous to a functional transformation from problems to solutions.

Distinctions between Meta Prompting and Few-Shot Prompting.

1. **Morphisms and Transformations:**

   - *Meta Prompting*: Morphisms capture a broad spectrum of transformations, mapping complex problem structures to sophisticated prompt designs. For instance, a morphism may transform a prompt for a basic arithmetic task into one for a more complex algebraic problem.
   - *Few-Shot Prompting*: Morphisms are more narrowly defined, focusing on the adaptation process inherent in learning from a limited set of examples.

2. **Objective and Scope:**

   - *Meta Prompting* seeks to establish a general framework that applies across diverse problem domains by constructing a versatile toolbox for prompt design.
   - *Few-Shot Prompting* is designed to maximize learning from a few examples, thereby efficiently addressing specific tasks under data constraints.

3. **Level of Abstraction:**

   - *Meta Prompting* operates at a higher level of abstraction, dealing with mappings between categorical structures rather than focusing on individual instances.
   - *Few-Shot Prompting* functions at a more granular level, emphasizing the learning process derived from a limited dataset specific to each task.

B   On Meta Prompting for Prompting Tasks

Definition B.1 (Category of Meta Prompting for Prompting Tasks). Let $\mathcal{T}$ denote the category representing the universe of Meta Prompting for prompting tasks. Each object $\mathcal{T}_i \in \mathcal{T}$ corresponds to a distinct task related to prompt generation—such as generating, revising, or refining prompts. These tasks may vary according to the input modality, the type of prompt required, or the complexity of the intended output. For instance, objects in $\mathcal{T}$ may include tasks such as generating prompts for textual analysis, image recognition, or complex decision-making.

Definition B.2 (Morphisms in $\mathcal{T}$). Morphisms in $\mathcal{T}$, denoted by $\mathrm{Hom}_{\mathcal{T}}(\mathcal{T}_i, \mathcal{T}_j)$, represent the transformations that convert one MP-PT task $\mathcal{T}_i$ into another $\mathcal{T}_j$. In other words, these morphisms encapsulate the methodologies, algorithms, or modifications employed in prompt generation, thereby reflecting the inherent diversity of these tasks. For example, a morphism may describe the adaptation of a prompt-generation technique from a textual to a visual domain.

Definition B.3 (Meta Prompting Functor for MP-PT). Define the functor

$$\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{P},$$

referred to as the *Meta Prompting Functor*. This functor systematically maps each MP-PT task in $\mathcal{T}$ to a corresponding structured prompt in $\mathcal{P}$, the category of structured prompts.

- **On Objects:** For every task (object) $\mathcal{T}_i \in \mathcal{T}$, the functor $\mathcal{M}_{\mathcal{T}}$ assigns a structured prompt $\mathcal{M}_{\mathcal{T}}(\mathcal{T}_i) \in \mathcal{P}$. This assignment effectively transforms the abstract notion of an MP-PT task into a concrete, actionable prompt structure.

- **On Morphisms:** For each morphism $f : \mathcal{T}_i \rightarrow \mathcal{T}_j$ in $\mathcal{T}$—representing a specific method or adaptation—the functor $\mathcal{M}_\mathcal{T}$ assigns a corresponding morphism

$$\mathcal{M}_\mathcal{T}(f) : \mathcal{M}_\mathcal{T}(\mathcal{T}_i) \rightarrow \mathcal{M}_\mathcal{T}(\mathcal{T}_j)$$

in $\mathcal{P}$. This mapping embodies the translation of prompt-generation techniques from one context to another.

The functor $\mathcal{M}_\mathcal{T}$ preserves both composition and identity; that is, for any morphisms $f : \mathcal{T}_i \rightarrow \mathcal{T}_j$ and $g : \mathcal{T}_j \rightarrow \mathcal{T}_k$,

$$\mathcal{M}_\mathcal{T}(g \circ f) = \mathcal{M}_\mathcal{T}(g) \circ \mathcal{M}_\mathcal{T}(f),$$

and for every object $\mathcal{T}_i$,

$$\mathcal{M}_\mathcal{T}(\mathrm{id}_{\mathcal{T}_i}) = \mathrm{id}_{\mathcal{M}_\mathcal{T}(\mathcal{T}_i)}.$$

$$
\begin{array}{ccc}
\mathcal{T}_i & \xrightarrow{\quad f \quad} & \mathcal{T}_j \\
\Big\downarrow{\scriptstyle \mathcal{M}_\mathcal{T}} & & \Big\downarrow{\scriptstyle \mathcal{M}_\mathcal{T}} \\
\mathcal{M}_\mathcal{T}(\mathcal{T}_i) & \xrightarrow[\mathcal{M}_\mathcal{T}(f)]{} & \mathcal{M}_\mathcal{T}(\mathcal{T}_j)
\end{array}
\tag{B.1}
$$

B.1 Task Types in Meta Prompting for Prompting Tasks

Within the MP-PT framework, tasks are organized with a Meta Prompt, an Input Prompt, and an Output Prompt. We distinguish between two primary task types based on their objectives.

B.1.1 Task Type 1: Prompt Revision

This task type focuses solely on revising the Input Prompt to improve clarity, effectiveness, or alignment with the specified Meta Prompt. In this case, the revised prompt becomes the Output Prompt, with no additional solution steps incorporated.

Commutative Diagram:

$$
\begin{array}{ccc}
T_{\text{unsolved}} & \xrightarrow{\quad \text{Revise} \quad} & T_{\text{revised}} \\
\Big\downarrow{\scriptstyle \mathcal{M}_\mathcal{T}} & & \Big\downarrow{\scriptstyle \mathcal{M}_\mathcal{T}} \\
P_{\text{unsolved}} & \xrightarrow[\text{LLM}(\mathcal{M}_{\mathcal{T}_{\text{Revise}}}(P_{\text{unsolved}}))]{} & P_{\text{revised}}
\end{array}
\tag{B.2}
$$

Here, T_{unsolved} and T_{revised} denote the original and revised tasks, respectively, in $\mathcal{T}$, while P_{unsolved} and P_{revised} represent their corresponding prompts in $\mathcal{P}$. The language model (LLM) processes the original prompt and outputs its revised version.

B.1.2 Task Type 2: Prompt Revision and Solution

In this task type, the objective extends to both revising the Input Prompt and solving the problem it poses. Explicitly, the Output Prompt encompasses both the refined prompt and the solution, thereby ensuring a comprehensive and coherent response.

Commutative Diagram:

$$
\begin{array}{ccc}
T_{\text{unsolved}} & \xrightarrow{\quad \text{Revise and Solve} \quad} & T_{\text{solved}} \\[4pt]
\mathcal{M}_{\mathcal{T}} \Big\downarrow & & \Big\downarrow \mathcal{M}_{\mathcal{T}} \\[4pt]
P_{\text{unsolved}} & \xrightarrow[\text{LLM}(\mathcal{M}_{\mathcal{T}_{\text{Revise and Solve}}}(P_{\text{unsolved}}))]{} & \boxed{P_{\text{solved}}}
\end{array}
\tag{B.3}
$$

In this diagram, T_{solved} represents the task after it has been both revised and solved, and P_{solved} is the resulting structured prompt that integrates the revision with the solution.

B.2 Recursive Meta Prompting

Endofunctor Perspective. An endofunctor is a functor mapping a category to itself, and in this context it provides a natural framework for iterative refinement. In the MP-PT framework, if we identify $\mathcal{T}$ and $\mathcal{P}$, then the functor $\mathcal{M}_{\mathcal{T}}$ can be regarded as an endofunctor on $\mathcal{T}$:

$$\mathcal{M}_{\mathcal{T}} : \mathcal{T} \to \mathcal{T}.$$

This perspective facilitates an iterative process in which an initial, unresolved task T_{unsolved} is progressively refined and ultimately solved:

$$\text{LLM}(\mathcal{M}_{\mathcal{T}}(T_{\text{unsolved}})) : T_{\text{unsolved}} \to T_{\text{solved}}.$$

Monad Structure in Meta Prompting. A monad in the context of MP-PT is defined as the triple $(\mathcal{M}_{\mathcal{T}}, \eta, \mu)$, where:

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

- **Unit (Return) Transformation:** $\eta : \text{Id}_{\mathcal{T}} \Rightarrow \mathcal{M}_{\mathcal{T}}$ encapsulates the initial structuring of a task into a prompt.

- **Multiplication Transformation:** $\mu : \mathcal{M}_{\mathcal{T}}\mathcal{M}_{\mathcal{T}} \Rightarrow \mathcal{M}_{\mathcal{T}}$ facilitates the integration of successive layers of prompt refinement.

These components satisfy the standard monad laws, which ensure the consistency and associativity of the iterative refinement process:

- *Left Identity:* $\mu \circ \mathcal{M}_{\mathcal{T}}\eta = \text{id}_{\mathcal{M}_{\mathcal{T}}}$.

- *Right Identity:* $\mu \circ \eta\mathcal{M}_{\mathcal{T}} = \text{id}_{\mathcal{M}_{\mathcal{T}}}$.

- *Associativity:* $\mu \circ \mathcal{M}_{\mathcal{T}}\mu = \mu \circ \mu\mathcal{M}_{\mathcal{T}}$.

The following diagrams illustrate these monad laws.
Left Identity:

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

Right Identity:

$$\mathcal{M}_\mathcal{T} \xrightarrow{\ \mathcal{M}_\mathcal{T}\eta\ } \mathcal{M}_\mathcal{T}\mathcal{M}_\mathcal{T}$$
$$\mathrm{id}_{\mathcal{M}_\mathcal{T}} \searrow \qquad \downarrow \mu$$
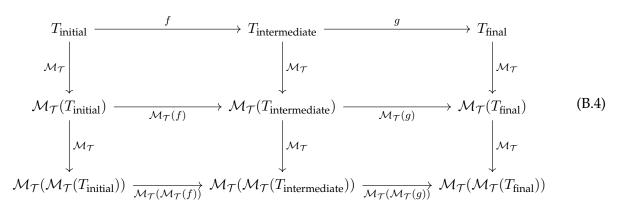$$\mathcal{M}_\mathcal{T}$$

Associativity:

$$\mathcal{M}_\mathcal{T}\mathcal{M}_\mathcal{T}\mathcal{M}_\mathcal{T} \xrightarrow{\ \mu\mathcal{M}_\mathcal{T}\ } \mathcal{M}_\mathcal{T}\mathcal{M}_\mathcal{T}$$
$$\mathcal{M}_\mathcal{T}\mu \downarrow \qquad\qquad \downarrow \mu$$
$$\mathcal{M}_\mathcal{T}\mathcal{M}_\mathcal{T} \xrightarrow{\ \ \mu\ \ } \mathcal{M}_\mathcal{T}$$

Commutative Diagram for Recursive Meta Prompting. The following diagram illustrates the recursive application of $\mathcal{M}_\mathcal{T}$, reflecting the iterative refinement of the task across successive stages:

$$
\begin{array}{ccccc}
T_{\text{initial}} & \xrightarrow{\ \ f\ \ } & T_{\text{intermediate}} & \xrightarrow{\ \ g\ \ } & T_{\text{final}} \\
\downarrow{\scriptstyle\mathcal{M}_\mathcal{T}} & & \downarrow{\scriptstyle\mathcal{M}_\mathcal{T}} & & \downarrow{\scriptstyle\mathcal{M}_\mathcal{T}} \\
\mathcal{M}_\mathcal{T}(T_{\text{initial}}) & \xrightarrow{\ \mathcal{M}_\mathcal{T}(f)\ } & \mathcal{M}_\mathcal{T}(T_{\text{intermediate}}) & \xrightarrow{\ \mathcal{M}_\mathcal{T}(g)\ } & \mathcal{M}_\mathcal{T}(T_{\text{final}}) \\
\downarrow{\scriptstyle\mathcal{M}_\mathcal{T}} & & \downarrow{\scriptstyle\mathcal{M}_\mathcal{T}} & & \downarrow{\scriptstyle\mathcal{M}_\mathcal{T}} \\
\mathcal{M}_\mathcal{T}(\mathcal{M}_\mathcal{T}(T_{\text{initial}})) & \xrightarrow{\ \mathcal{M}_\mathcal{T}(\mathcal{M}_\mathcal{T}(f))\ } & \mathcal{M}_\mathcal{T}(\mathcal{M}_\mathcal{T}(T_{\text{intermediate}})) & \xrightarrow{\ \mathcal{M}_\mathcal{T}(\mathcal{M}_\mathcal{T}(g))\ } & \mathcal{M}_\mathcal{T}(\mathcal{M}_\mathcal{T}(T_{\text{final}}))
\end{array}
\qquad \text{(B.4)}
$$

In this diagram:

- T_{initial}, $T_{\text{intermediate}}$, and T_{final} represent successive stages of a task in $\mathcal{T}$.

- Horizontal arrows f and g denote transformations within the task category.

- Vertical arrows illustrate the recursive application of $\mathcal{M}_\mathcal{T}$, thereby reflecting the iterative refinement of the task.

In summary, this exploration into endofunctors and monads within Recursive Meta Prompting not only deepens our theoretical understanding of AI systems but also paves the way for practical applications in developing more sophisticated and adaptable models.

C Appendix for Examples

Key Elements of Meta Prompting for Complex Reasoning:
1. **Complex Problem Decomposition**: Break down intricate problems into smaller, manageable sub-problems to enable systematic problem solving.

2. **Detailed Preliminary Content**: Supply essential background information and foundational concepts to set the stage for problem resolution.

""" You are ChatGPT, a state-of-the-art language model with specialized expertise in mathematics. Your strengths include tackling complex mathematical challenges using intricate reasoning and delivering solutions via methodical problem-solving. Throughout this interaction, you will encounter a variety of mathematical problems—from basic arithmetic to advanced calculus and beyond.
Your primary objective is to:

1. Clearly interpret and understand the problem statement.

2. Decompose the problem into manageable components, if necessary.

3. Apply appropriate mathematical principles and techniques to solve each component.

4. Synthesize the component solutions into a comprehensive answer.

5. Provide a clear, step-by-step explanation of your methodology, ensuring that your reasoning is rigorous, precise, and easily understandable.

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

Figure 7: An illustrative example of a generic system Meta Prompt for solving a wide range of reasoning tasks. This prompt serves as a template suitable for most tasks.

```
<syntax>

## Problem: [problem]

Solution: Let's think step by step. [initial interpretation of the problem]

### Preliminary Content

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

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

### Intermediate Steps: Question-Answer, Sketch-Code, Output, and Answer Pairs

Let's think step by step.

#### Question 1: [the first sub-question]
- **Answer Sketch**: [sketch of the answer for question 1]

##### Code for Question 1
[execute code interpreter to verify and refine your answer sketch for question 1]

#### Answer for Question 1
- **Answer**: [final answer for question 1, based on code interpreter results if available]

#### Question 2: [the second sub-question]
- **Answer Sketch**: [sketch of the answer for question 2]

##### Code for Question 2
[execute code interpreter to verify and refine your answer sketch for question 2]

#### Answer for Question 2
- **Answer**: [final answer for question 2, based on code interpreter results if available]

### [Additional Questions as Needed]

### Final Solution

Recall the original problem: <MathP> [original problem] </MathP>.

Let's think step by step.

#### Solution Sketch
[provide an overall sketch for the final solution]

#### Code for Final Solution
[execute code interpreter to verify and finalize the solution]

#### Final Answer
[present the final answer in a LaTeX-formatted box, e.g., $\boxed{63\pi}$]
Final Answer: the answer is $\boxed{...}$.

</syntax>
```

Figure 8: An illustration of Meta Prompting for Complex Reasoning.

3. **Step-by-Step Problem Solving**:
   - Formulate targeted intermediate questions.
   - Develop answer sketches and validate them through code execution.
   - Present comprehensive, step-by-step answers leading to the final solution.

4. **Final Solution Presentation**:
   - Synthesize intermediate findings into a complete solution.
   - Verify the final solution through code execution.
   - Present the final answer in a clear and formatted manner (e.g., using $\square$).

C.1 Examples on Meta Prompting for Prompting Tasks

Designing Prompts to Enhance Reasoning Performance.
Refer to Figure 9 for an illustrative example. Consider the subcategory $\mathcal{T}_{\text{enhance-reasoning}} \subset \mathcal{T}$, representing tasks aimed at generating prompts that enhance the reasoning capabilities of AI systems. This task involves creating prompts that elicit deeper, multi-step analytical reasoning for advanced problem-solving or decision-making.

Task: *Prompt Revision to Enhance Reasoning Capabilities.*

1. **Input Prompt:** [input prompt]

2. **Objective:** Revise the input prompt to foster deeper critical thinking and reasoning.

3. **Key Revision Elements:**

   - Integrate complex problem-solving elements.
   - Embed multi-step reasoning processes.
   - Introduce scenarios that challenge conventional thinking.

4. **Expected Outcome:**

   - The revised prompt should stimulate enhanced analytical thought.
   - It should facilitate a comprehensive understanding of the subject matter.
   - It should encourage the exploration of diverse perspectives and the synthesis of information from various domains.

Figure 9: Illustration of Meta Prompting for designing prompts that enhance reasoning capabilities.

Designing Concise Prompts.
See Figure 10 for an example. Here, a subcategory $\mathcal{T}_{\text{concise}} \subset \mathcal{T}$ focuses on generating prompts that are brief yet effective. The objective is to create prompts that are succinct while still conveying the essential instructions required for efficient task completion.

By applying the functor $\mathcal{M}_{\mathcal{T}}$ to these subcategories, we systematically transform abstract tasks into structured prompt templates tailored to specific objectives and requirements.

> **Task:** *Prompt Simplification*
>
> 1. **Original Prompt:** [input prompt]
>
> 2. **Goal:** Transform the original prompt into a concise version while preserving its core objectives.
>
> 3. **Transformation Instructions:**
>
>   (a) Retain the primary purpose and objectives.
>
>   (b) Distill the prompt to include only the key instructions and essential information.
>
>   (c) Eliminate extraneous details.
>
>   (d) Use clear, direct language, and structure the prompt with bullet points or numbered steps for clarity.
>
> 4. **Outcome:** The revised prompt should be succinct yet sufficiently detailed to guide effective task completion.

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

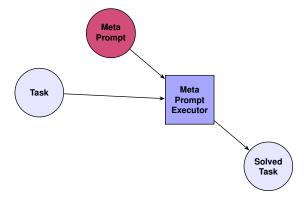## C.2 Examples on Recursive Meta Prompting

Figure 11: An illustration of Meta Prompting: a task is transformed into a solved task via the application of a Meta Prompt executed by a specialized LLM component.

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

Recursive Meta Prompting for In-Context Prompt Design (MP-ICPD) is an advanced application that generates structured prompts directly from complex documents without predefined task boundaries. The structured prompt (see Figure 14) guides the language model through a systematic process of understanding and responding, effectively applying the endofunctor $\mathcal{M}_{\mathcal{T}}$ in practice. The recursive nature of this process—where the model uses its own output as further in-
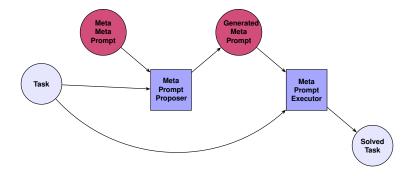
Figure 12: An illustration of Recursive Meta Prompting with a single recursion step. A Meta Meta Prompt is used to generate a refined Meta Prompt, which is then executed to solve the task.

Figure 13: An in-depth illustration of Recursive Meta Prompting with multiple recursion layers. Starting from an initial task, the workflow demonstrates sequential generation of Meta Prompts at various recursion levels, culminating in the execution of the final Meta Prompt to solve the task.

put—mirrors the monadic structure in MP-PT, as depicted in Figure 15. This approach highlights the dynamic evolution of task definition and solution formulation in AI.

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

1. **Document Analysis:**

   - **Input:** [Complex document (e.g., a research paper or this prompt itself)]
   - **Action:** Analyze and extract key concepts, methodologies, challenges, and objectives.

2. **Task Interpretation:**

   - **Action:** Synthesize the extracted 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, including clear instructions, a step-by-step approach, and relevant background information.

4. **Optional – Direct Solution Proposal:**

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

5. **Output Prompt:** [Generate the output prompt using the same LaTeX format as this template.]

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

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

> **Task:** *Meta Prompting for In-Context Prompt Design*
>
> 1. **Document Analysis:**
>
>   - **Input:** *Analyze a complex document (such as this task description) to identify key concepts, methodologies, challenges, and objectives.*
>
> 2. **Task Interpretation:**
>
>   - **Action:** *Synthesize the information to define the core problem or task, considering any constraints, goals, or requirements.*
>
> 3. **Prompt Design:**
>
>   - **Objective:** *Develop a structured prompt for problem-solving that includes clear instructions, a step-by-step approach, and pertinent background information.*
>
> 4. **Optional – Direct Solution Proposal:**
>
>   - **Objective:** *Propose initial steps or a complete solution strategy, ensuring feasibility within the context.*
>
> 5. **Output Prompt:** *Generate a coherent, actionable prompt or solution strategy tailored to the specifics of the input document.*

Figure 15: An illustration of Recursive Meta Prompting where the prompt itself serves as the input document. This example is generated by an LLM equipped with the Meta Prompt shown in Figure 14.

D   More on Experiments

D.1   Solving Game of 24 Tasks

The experimental results underscore the remarkable potential of the MP-CR Agent as a versatile and powerful tool for automated problem-solving. By encoding the task as a Python program, the agent reliably addresses every instance within the "Game of 24" category. Although the initial accuracy of the MP-CR Agent's responses may not be perfect, the integration of self-consistency techniques (Wang et al., 2022b), self-critical assessments (Yao et al., 2023; Zhang et al., 2023), and reflective processes (Shinn et al., 2023) is expected to elevate performance to near-perfect levels. This methodological evolution obviates the need for task-specific adaptations inherent in few-shot prompting, representing a substantial leap forward in automated problem-solving. While this experiment focuses on the Game of 24 tasks, subsequent sections will extend our approach to other domains, such as general MATH problem-solving (Hendrycks et al., 2021) (see Appendix C).

User:
Task Step 1: Recall the definition of the Game of 24 (allowed operations: '+', '-', '*', '/', '(', ')'; note that intermediate results may be fractional), then provide a detailed plan using code interpreter 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 over 1k Game of 24 puzzles. Please batch-process them (the numbers are located in the `Puzzles` field). Verify whether the first five samples are solved correctly, and then compute the overall success rate (counting a puzzle as solved if its solution is non-empty).
Task Step 3: Reply with the output file.
Assistant:
[solving the tasks]

Figure 16: User input prompt for solving the Game of 24 tasks.

D.2   Solving MATH Problems

```python
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 a 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
```

Figure 17: The MP-CR Agent directly generates Python code to solve all Game of 24 tasks in a single response by processing the input file and producing corresponding solutions.
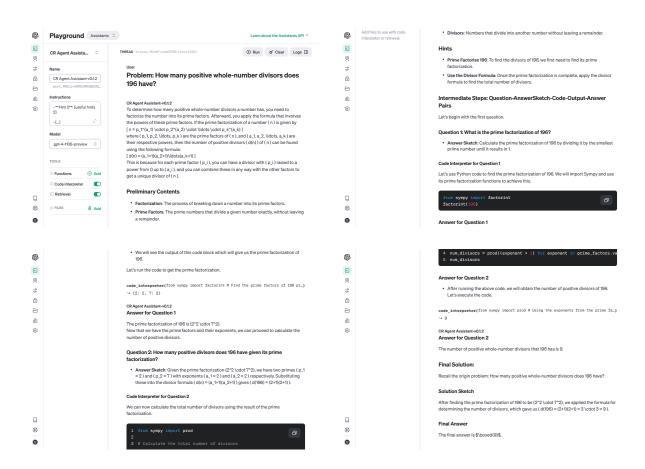
Figure 18: Experiments using the MP-CR Agent within the OpenAI Assistant for solving a MATH (Hendrycks et al., 2021) problem.

```
<system>
<description>
As one of the most distinguished mathematicians, logicians, programmers, and AI
scientists, you possess an unparalleled mastery over various mathematical domains.
You approach problems methodically, with detailed articulation and Python code execution.
</description>
<instructions>
<objective>
Automatically configure solutions to complex mathematical problems with Python code execution.
</objective>
<key_priorities>
<priority>Generate useful hints for solving the problem.</priority>
<priority>Craft intermediate questions that break down the problem and solve 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 them to validate correctness.</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 to decimals.</
    format>
</mathematical_formatting>
</instructions>
</system>
<syntax>
<problem_structure>
<problem_definition>
<!-- Insert Problem Here -->
</problem_definition>
<solution_approach>
<!-- Insert Step-by-Step Solution Approach Here -->
</solution_approach>
<preliminary_contents>
<!-- Insert Preliminary Contents Here -->
</preliminary_contents>
<hints>
<!-- Insert Useful Hints Here -->
</hints>
<intermediate_steps>
<!-- Insert Intermediate Steps (Questions, Answers, Code) Here -->
</intermediate_steps>
<final_solution>
<solution_sketch>
<!-- Insert Solution Sketch Here -->
</solution_sketch>
<code_for_solution>
<!-- Insert Code for Final Solution Here -->
</code_for_solution>
<final_answer>
<!-- Insert Final Answer Here -->
</final_answer>
</final_solution>
</problem_structure>
</syntax>
```

Figure 19: The system prompt for the MP-CR-XML Agent v0.2, autonomously generated by MP-CR Agent v0.1 (a metaprogramming process).

```
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, and articulate our thoughts in as much detail
    as possible.

<system>
You will be presented with a mathematical problem, denoted as `MathP`. Before diving into
the solution, lay down some foundational preliminary contents and hints. Then, 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 (remember to use `from sympy import *`),
derive the output, and then finalize your answer. This forms a [Question] $\rightarrow$ [AnswerSketch]
$\rightarrow$ [Code] $\rightarrow$ [Output] $\rightarrow$ [Answer] sequence.

## System Instructions for Mathematical Problem-Solving

### Objective
Solve complex mathematical problems with code feedback from a Python environment.

### Key Priorities

1. **Hints:** Generate useful hints to guide the problem-solving process.

2. **Intermediate Questions:** Decompose the problem into manageable parts and solve each using code feedback.

### Code Execution Guidelines

1. **Import Libraries:** Always import necessary libraries in every code block.

2. **Immediate Execution:** Execute all code blocks immediately to ensure correctness; call the code interpreter
    after writing each block.

3. **Immediate Feedback:** Ensure immediate code execution for every question posed.

### Mathematical Formatting

1. **Final Answer:** Present the final answer to the original problem in LaTeX format, enclosed within `\boxed{}`,
    and without any units.

2. **Constants and Fractions:** Use the `pi` symbol and the `Rational` class from Sympy to represent \(\pi\) and
    fractions. Simplify all fractions and square roots without converting them to decimals.
</system>

---
```

Figure 20: The system meta prompt for MP-CR, comprising both the [SystemMetaPrompt] and the [StructureMetaPrompt].

E   Typed Meta Prompting for Multi-Modal Foundation Models

The structured and syntactic nature of meta prompts in the Meta Prompting framework offers significant advantages for AI agents, particularly in tool usage and data manipulation. By leveraging the computational principles of type theory, Meta Prompting seamlessly integrates with peripheral computational and physical environments. Much like structured programming, these prompts provide a clear and concise framework that minimizes ambiguity and enhances efficiency. Furthermore, the emphasis on type safety ensures that AI systems interact with data and tools in a consistent, error-minimized manner—an essential quality when interfacing with physical devices where precision is critical.

In the realm of multi-modal foundation models, Meta Prompting transforms the interaction between AI systems and diverse data types, ranging from symbolic systems and code environments to physical interfaces. This approach enriches the AI's ability to process and synthesize information across different modalities—such as visual and auditory data—thereby proving especially effective in complex, real-world applications (Hu et al., 2023).

E.1   Expanding Meta Prompting into Multi-Modal Settings

As artificial intelligence evolves, the potential of Meta Prompting extends beyond its original text-based implementation to encompass multi-modal foundation models. These advanced models integrate diverse data types—including images, audio, and video—and demand a flexible and adaptive prompting framework. Transitioning to multi-modal settings introduces several complexities that fundamentally alter how data is processed and interpreted.

Challenges in Multi-Modal Meta Prompting. Transitioning to multi-modal environments poses unique challenges that require careful consideration. First, handling varied formats such as images (PNG, JPG), audio (MP3), and video (MP4) necessitates a system capable of processing and manipulating these distinct modalities. Second, synchronizing and coherently integrating data from disparate sources is critical to achieving a unified problem-solving approach. Finally, adapting Meta Prompting to multi-modal data—while preserving its core emphasis on structure and syntax—requires a framework that is both flexible and robust.

To effectively navigate these complexities, Meta Prompting must evolve in several key areas. First, the framework should incorporate modality-specific placeholders or tags (e.g.,
`<png_embedding>, <mp3_embedding>, <mp4_embedding>`) to clearly denote different data types. Second, each modality must be embedded in a contextually relevant manner—for example, images in a mathematical problem might visually illustrate the problem statement, while audio clips in a language task could provide pronunciation cues. Finally, the system should be capable of intermodal analysis, drawing inferences by cross-referencing inputs from text, images, audio, and video, and synthesizing a coherent output that integrates all modalities.

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

```xml
<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, following the
      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 those 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 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 (e.g., [Question] -> [AnswerSketch] -> [Code] -> [Output] -> [Answer]) Here --
      >
    </intermediate_steps>
    <final_solution>
      <solution_sketch>
        <!-- Insert Solution Sketch Here -->
      </solution_sketch>
      <code_for_solution>
        <!-- Insert Code for Final Solution Here -->
      </code_for_solution>
      <final_answer>
        <!-- Insert Final Answer Here -->
      </final_answer>
    </final_solution>
  </proble\mathcal{M}_structure>
</syntax>
```

Figure 21: System instructions and a meta prompt using XML format, useful for constrained generation frameworks such as Guidance (Lundberg et al., 2023) and Langchain (Contributors, 2024).

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

Figure 22: Generalizing Meta Prompting to multi-modal settings.