Meta Prompting: A Framework for Agentic and Compositional Reasoning

Anonymous authors
Paper under double-blind review

Abstract

We introduce Meta Prompting (MP), a framework that elevates the reasoning capabilities of large language models (LLMs) by focusing on the formal structure of a task rather than content-specific examples. We establish a theoretical foundation for this paradigm, formalizing MP as a functor that maps a category of tasks to a category of structured prompts, thereby guaranteeing that compositional problem-solving strategies can be systematically decomposed into modular prompt structures. We extend this concept to Recursive Meta Prompting (RMP), an automated process where an LLM can generate and refine its own prompts. We model this self-improvement loop formally as a monad, providing a principled framework for automated prompt engineering. Our claims are validated through extensive experiments demonstrating that a Qwen-72B base model, guided by a single, example-agnostic meta-prompt, achieves state-of-the-art results: 46.3% PASS@1 on MATH, surpassing fine-tuned models and GPT-4; 83.5% on GSM8K; and a 100% success rate on the Game of 24. These results are achieved with substantial token efficiency gains over traditional few-shot methods.

1 Introduction

The advent of foundation models, particularly Large Language Models (LLMs), has transformed the research of various fields. 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). While this design excels at a broad range of tasks, it is not inherently configured for the depth and sophistication 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, struggling with tasks that demand the structured, multi-step 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 toward more deliberative reasoning. While these methods have improved performance, they primarily rely on content-based examples and lack a formal, compositional structure for building complex reasoning processes.

In response to these challenges, we introduce Meta Prompting (MP), a paradigm that shifts the focus from content-based analogy to formal procedural guidance. Instead of providing examples of what to think, a meta-prompt provides a structured template for how to think. We establish a theoretical foundation for MP, formalizing it as a functorial mapping from a category of tasks to a category of prompts. This categorical

```
Integrate step-by-step reasoning to solve mathematical problems under the 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.

framework guarantees that compositional problem-solving strategies can be mapped to modular and reusable prompt structures, yielding a systematic and adaptable approach to complex reasoning.

A pivotal innovation of our work is the application of this framework recursively—a concept we term Recursive Meta Prompting (RMP). Analogous to metaprogramming, RMP enables an LLM to autonomously generate and refine its own prompts. We model this self-improvement process formally using a monad, providing a principled framework for a model to not only solve problems, but to learn how to improve its own problem-solving strategies. This self-referential capability is a significant step towards greater model autonomy and automated prompt engineering.

The efficacy of our framework is validated through extensive experiments on challenging benchmarks, including the Game of 24 (Yao et al., 2023), GSM8K (Cobbe et al., 2021), and MATH (Hendrycks et al., 2021).

In summary, our contributions are as follows:

- We introduce Meta Prompting (MP) and formalize it using category theory. We model MP as a functor that preserves compositional structure, and we prove a proposition on its compositional properties. We further introduce Recursive Meta Prompting (RMP) and model it with a monadic framework that provides a principled basis for prompt self-improvement.
- 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.

2 Background

Category theory provides a high-level language for describing mathematical structures and their relationships. We use it to formalize the relationship between task structures and prompt structures.

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.

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, $\boxed{\cdots}$, 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.

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

2.3 Monads in Category Theory

Definition 2.8 (Monad)**.** A *monad* on a category $\mathscr{C}$ is a triple (T, η, μ) consisting of:
item An *endofunctor* $\mathrm{T} : \mathscr{C} \to \mathscr{C}$.

- A natural transformation $\eta : \mathrm{Id}_{\mathscr{C}} \to \mathrm{T}$, called the *unit* (or return).
- A natural transformation $\mu : \mathrm{T} \circ \mathrm{T} \to \mathrm{T}$, called the *multiplication* (or join).

These components must satisfy coherence conditions known as the monad laws (associativity and left/right identity), which ensure that compositions of monadic operations behave in a well-structured manner. In computer science, monads are fundamental for modeling computations with side effects, such as state, I/O, or, as we will argue, recursive prompt refinement.

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*.

This emphasis on structure is analogous to type theory (see Appendix A.1), where each component of a prompt can be assigned a "type" (e.g., 'ProblemStatement: string', 'ReasoningStep: list[string]', 'FinalAnswer: float'). A meta-prompt thus defines a "type signature" for the desired output, guiding the LLM to generate responses that are not only semantically relevant but also syntactically correct according to the specified format. Examples of such structured prompts are shown in Figures 1 and 2.

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 $\text{Hom}_{\mathcal{T}}(X, Y)$, 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 $\text{Hom}_{\mathcal{P}}(U, V)$, 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.

A functor must preserve 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 $X \in \mathcal{T}$, $\mathcal{M}(\text{id}_X) = \text{id}_{\mathcal{M}(X)}$.

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.

The preservation of composition, $\mathcal{M}(g \circ f) = \mathcal{M}(g) \circ \mathcal{M}(f)$, is not merely a mathematical formality; it is the theoretical guarantee of modularity and systematic problem decomposition. It implies that if a complex problem-solving strategy can be constructed by composing simpler strategies, the corresponding prompt can also be constructed by composing simpler prompts in a principled manner. This insight forms the basis for building complex reasoning chains from fundamental building blocks. We formalize this as a proposition.

Theorem 3.4 (Compositionality of Meta Prompting)**.** Let a task $T \in \mathcal{T}$ be composed of sub-tasks T_1 and T_2 via a composition of morphisms, such that T is the result of applying transformation $f : T_1 \to T_2$ followed by $g : T_2 \to T_3$. The meta-prompt for the composite task, $\mathcal{M}(g \circ f)$, is equivalent to the composition of the meta-prompts for the sub-tasks, i.e., $\mathcal{M}(g) \circ \mathcal{M}(f)$.

Proof Sketch. The theorem holds by the definition of $\mathcal{M}$ as a functor. A functor is a structure-preserving map between categories, which by definition must preserve the composition of morphisms. Therefore, the compositional structure of tasks in $\mathcal{T}$ is necessarily preserved in the structure of prompts in $\mathcal{P}$. This property ensures that breaking down a complex problem into a sequence of simpler steps in the task domain corresponds directly to a sequence of prompt transformations in the prompt domain. □

This mapping can be hand-crafted by a human or generated using LLMs via a recursive, self-composing method (see Section 4). 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, $\boxed{\ldots}$."
},
"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.

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.

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 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 a limited set of concrete, content-rich '(problem, solution)' pairs to guide the model via in-context analogy. In contrast, Meta Prompting provides a single, content-agnostic structural template that outlines the reasoning process. It teaches the model how to think, whereas few-shot prompting shows the model what has been thought.

Beyond few-shot methods, Meta Prompting also distinguishes itself from other programmatic or structured prompting approaches, such as those using XML tags or frameworks like DSPy (Khattab et al., 2023). While these methods also impose structure, they often function as programming layers that compile into traditional few-shot or zero-shot prompts. Meta Prompting, as formalized here, is a more fundamental concept focused on the direct, example-agnostic mapping between a task's abstract structure and a prompt's syntactic structure. For further illustration of these differences, please refer to Figures 1, 2, and 3.

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 9 in Appendix B.)

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.

Algorithm 1 Recursive Meta Prompting (RMP)

Input: Initial task description T_0, Meta-Meta-Prompt P_{meta}, LLM $\mathcal{L}$
$P_{\text{current}} \leftarrow \text{InitialPrompt}(T_0)$ {Generate a basic prompt}
for $i = 1$ **to** $N_{\text{max_iterations}}$ **do**
  $P_{\text{refined}} \leftarrow \mathcal{L}(P_{\text{meta}}, P_{\text{current}})$ {Refine the prompt}
  if $\text{IsConverged}(P_{\text{refined}}, P_{\text{current}})$ **then**
    break
  end if
  $P_{\text{current}} \leftarrow P_{\text{refined}}$
end for
$\mathit{Solution} \leftarrow \mathcal{L}(P_{\text{current}}, T_0)$ {Solve task with the final prompt}
return $\mathit{Solution}$

4 Recursive Meta Prompting: Self-Refinement and Automation

While Meta Prompting provides a robust framework for solving external tasks, its most powerful application lies in turning the lens of prompting inward: using meta-prompts to generate and refine other prompts. We

call this process **Recursive Meta Prompting (RMP)**. RMP endows an AI system with the capacity for autonomous self-improvement, where the model not only executes tasks but also actively constructs and enhances its own guiding instructions. This paradigm mirrors the concept of metaprogramming in computer science, where a program can treat other programs (or itself) as data to be analyzed and modified.

4.1 A Monadic Framework for Prompt Refinement

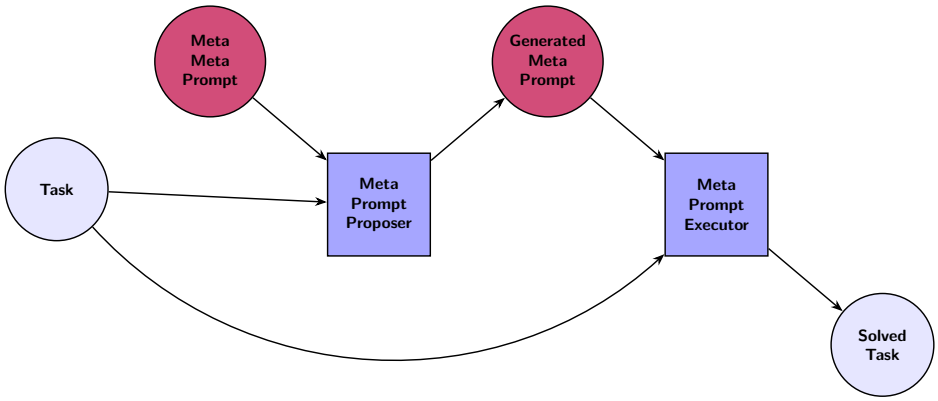

Figure 6: The workflow of Recursive Meta Prompting. A *Meta-Meta-Prompt* guides a Proposer LLM to generate a task-specific Meta Prompt. This generated prompt is then used by an Executor LLM to solve the original task.

The process of recursive refinement is elegantly captured by the mathematical structure of a monad. We can model RMP as an endofunctor on the category of prompts, $\mathcal{M}_{\mathcal{P}} : \mathcal{P} \to \mathcal{P}$, which takes a prompt and outputs a refined version. This forms a monad $(\mathcal{M}_{\mathcal{P}}, \eta, \mu)$:

- **Endofunctor** $\mathcal{M}_{\mathcal{P}}$: The core refinement operation. Given a prompt P, $\mathcal{M}_{\mathcal{P}}(P)$ is the improved prompt.
- **Unit** $(\eta : \mathrm{Id} \to \mathcal{M}_{\mathcal{P}})$: The 'unit' transformation takes a simple task description and lifts it into a structured meta-prompt, creating the initial object for refinement.
- **Multiplication** $(\mu : \mathcal{M}_{\mathcal{P}}^2 \to \mathcal{M}_{\mathcal{P}})$: The 'multiplication' transformation is the essence of recursion. It takes a nested prompt refinement—a prompt about how to refine a prompt, $\mathcal{M}_{\mathcal{P}}(\mathcal{M}_{\mathcal{P}}(P))$—and flattens it into a single, executed refinement $\mathcal{M}_{\mathcal{P}}(P)$.

This structure leads to the following proposition regarding the stability of the refinement process.

Theorem 4.1 (Stability of Recursive Refinement)**.** The process of recursive prompt refinement is associative and stable. Given a multiply-nested refinement, the order in which the refinement steps are collapsed does not alter the final outcome.

Proof Sketch. This property is a direct consequence of the monad's associativity law: $\mu \circ \mathcal{M}_{\mathcal{P}}\mu = \mu \circ \mu\mathcal{M}_{\mathcal{P}}$. This identity guarantees that for any thrice-nested prompt $\mathcal{M}_{\mathcal{P}}^3(P)$, the two possible ways of flattening it to $\mathcal{M}_{\mathcal{P}}(P)$ yield the same result (see Appendix A.4). This ensures that the self-refinement process is coherent and computationally stable, preventing arbitrary outcomes from nested metaprogramming. □

This monadic structure guarantees that the refinement process is consistent and compositional (see Appendix A.4 for the monad laws). The recursive mechanism, depicted in Figure 6, allows an LLM to iteratively improve its own instructions, moving from a vague initial prompt to a sophisticated, highly-structured one. This process enables an iterative problem-solving loop where an initial, unsolved prompt T_{unsolved} is successively refined until it can be solved by the LLM:

$$\mathrm{LLM}(\mu(\mathcal{M}_{\mathcal{P}}(\eta(T_{\text{unsolved}})))) \to T_{\text{solved}}.$$

4.2 Case Study: Automatic Prompt Derivation

To make the RMP process concrete, consider the task of deriving a sophisticated prompt for a new problem domain. The process, formalized in Algorithm 1, can be used to generate the very meta-prompts used in our experiments. For instance, an LLM equipped with a high-level 'Meta-Meta-Prompt' (as shown in Figure 7) can take its own prompt as input and recursively improve it.

Figure 7 serves as an example of a 'Meta-Meta-Prompt'. When an LLM is given this prompt and tasked with analyzing a document (which could be the prompt itself), it is guided to perform document analysis, interpret the core task, and design a new, structured prompt for solving it. This self-referential loop demonstrates the practical application of RMP for in-context prompt design, showcasing the dynamic evolution of task definition and solution formulation.

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 7: An example of a meta-meta-prompt for In-Context Prompt Design (MP-ICPD). This prompt instructs an LLM on how to analyze a document and generate a new, structured meta-prompt to solve the task described within it.

By automating the prompt-generation process, RMP enhances the adaptability and autonomy of AI systems, facilitating a more modular and compositional approach to problem-solving.

5 Experiments

In this section, we evaluate the performance of our proposed Meta Prompting (MP) framework on several mathematical benchmarks and problem-solving tasks. Our experiments are designed to assess both accuracy and efficiency.

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 groundbreaking 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	24.8
Qwen-14B (base)	-	No	**MP**	**28.9**
Qwen-72B (base)	-	No	CoT	35.2
Qwen-72B-MetaMathQA	MetaMathQA	No	CoT	41.7
Qwen-72B (base)	-	No	**MP**	**46.3**

Table 2: 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**

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, 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 14 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 12 in Appendix C.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 C.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. This

foundational idea has been extended in numerous ways. Methodologies like Self-Consistency (Wang et al., 2022b) and Complex CoT (Fu et al., 2022) focus on generating multiple reasoning chains and selecting the best one, often through voting. Decomposition strategies, such as Least-to-Most (Zhou et al., 2022) and Decomposed Prompting (Khot et al., 2022), focus on breaking complex problems into simpler, solvable sub-tasks. More recent work has explored multi-agent debates (Du et al., 2023), diverse reasoning paths with verifiers (Li et al., 2023), and progressive, iterative refinement (Zheng et al., 2023). A significant parallel thread involves enabling LLMs to self-criticize and self-correct their reasoning paths, with theoretical guarantees on improvement (Tyen et al., 2023; Li et al., 2024; Wang et al., 2024). While powerful, these methods primarily aim to improve the semantic content of the reasoning chain, often relying on complex few-shot examples. Meta Prompting differs fundamentally by being example-agnostic and operating on the formal, syntactic structure of the prompt itself, thereby controlling the reasoning process at a more abstract level.

Structured and Graph-Based Reasoning Frameworks. Recognizing the limitations of linear reasoning chains, recent work has explored more complex reasoning topologies. The Tree-of-Thought (ToT) framework (Yao et al., 2023; Long, 2023) was a significant step, allowing an LLM to explore multiple reasoning paths in a tree structure and use self-evaluation to prune branches. This concept has been further generalized to graph-based structures in frameworks like Cumulative Resoning (CR) (Zhang et al., 2023), Graph-of-Thoughts (GoT) (Besta et al., 2024), which represents thoughts as nodes and dependencies as edges, and Forest-of-Thoughts (FoT) (Bi et al., 2024), which explores diverse high-level plans concurrently. These frameworks provide powerful high-level strategies for exploring a problem space. Our work on Meta Prompting is orthogonal and complementary. While ToT, GoT, and FoT define the macro-level topology of the reasoning process (a tree or graph), Meta Prompting provides a formal, categorical language for defining the micro-level structure of the nodes and compositional rules for the edges. The structured prompts generated by our framework can be seen as the well-defined computational steps within these more complex reasoning graphs.

7 Conclusion

In this work, we introduced Meta Prompting, a paradigm that prioritizes the formal structure of reasoning over its content. We moved beyond intuition by establishing a theoretical foundation, using category theory to formalize Meta Prompting as a functor and Recursive Meta Prompting (RMP) as a monad. This framework provides a principled, compositional, and automatable approach to guiding the reasoning processes of large language models. Our empirical results show that this structure-oriented approach leads to state-of-the-art performance on challenging mathematical benchmarks in a zero-shot, example-agnostic setting.

The implications of this work suggest a new way to interact with and control LLMs, moving from empirical prompt engineering to a more formal, programmatic science of prompt design. The RMP framework, in particular, points toward a future where LLMs can autonomously improve their own cognitive strategies, adapting and optimizing their internal instructions for new tasks without direct human intervention. Our work lays a formal cornerstone for this next generation of agentic and compositional reasoning systems.

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.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, et al. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17682–17690, 2024.

Zhenni Bi, Kai Han, Chuanjian Liu, Yehui Tang, and Yunhe Wang. Forest-of-thought: Scaling test-time compute for enhancing llm reasoning. *arXiv preprint arXiv:2412.09078*, 2024.

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.

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.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 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.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.

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.

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.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714*, 2023.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*, 2022.

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.

Loka Li, Guangyi Chen, Yusheng Su, Zhenhao Chen, Yixuan Zhang, Eric Xing, and Kun Zhang. Confidence matters: Revisiting intrinsic self-correction capabilities of large language models. *arXiv preprint arXiv:2402.12563*, 2024.

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.

Gladys Tyen, Hassan Mansoor, Peter Chen, Tony Mak, and Victor Cărbune. Llms cannot find reasoning errors, but can correct them! *arXiv preprint arXiv:2311.08516*, 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.

Yifei Wang, Yuyang Wu, Zeming Wei, Stefanie Jegelka, and Yisen Wang. A theoretical understanding of self-correction through in-context alignment. *arXiv preprint arXiv:2405.18634*, 2024.

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.

Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models. *arXiv preprint arXiv:2304.09797*, 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.

Appendix

A Theoretical Foundations **18**

A.1 Type Theory 18

A.2 Natural Transformations 18

A.3 Proof of Compositionality (Theorem 3.4) 18

A.4 The Monadic Framework for RMP 19

A.5 Proof of Stability (Theorem 4.1) 19

B Additional Prompt Examples **20**

C Additional Experimental Details **22**

C.1 Solving Game of 24 Tasks 22

C.2 Solving MATH Problems 22

D Multi-Modal Meta Prompting **22**

D.1 A Framework for Multi-Modal Interaction 22

A Theoretical Foundations

A.1 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 term : type. Common types include the natural numbers ($\mathbb{N}$) 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 that it is inherently computational (e.g., '1+1' and '2' are syntactically distinct but computationally equivalent), intuitionistic, and encodes mathematical objects via inductive types rather than as sets. This makes it a natural language for describing the structured, computational nature of meta-prompts.

A.2 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 corresponding diagram commutes. When each m_A is an isomorphism, the transformation is a *natural isomorphism*.

This concept is crucial for understanding the RMP monad, where the unit (η) and multiplication (μ) are natural transformations.

A.3 Proof of Compositionality (Theorem 3.4)

Proof. Let $\mathcal{M} : \mathcal{T} \to \mathcal{P}$ be the Meta Prompting functor. To prove the proposition, we must show that $\mathcal{M}$ preserves the structure of composition as defined for a functor.

Let T_1, T_2, T_3 be objects in the category of tasks $\mathcal{T}$. Let $f : T_1 \to T_2$ and $g : T_2 \to T_3$ be morphisms representing transformations between these tasks. The composition of these morphisms is $g \circ f : T_1 \to T_3$.

By the definition of a functor, $\mathcal{M}$ maps these objects and morphisms to the category of prompts $\mathcal{P}$:

- Objects: $\mathcal{M}(T_1)$, $\mathcal{M}(T_2)$, $\mathcal{M}(T_3)$ are objects in $\mathcal{P}$.
- Morphisms: $\mathcal{M}(f) : \mathcal{M}(T_1) \to \mathcal{M}(T_2)$ and $\mathcal{M}(g) : \mathcal{M}(T_2) \to \mathcal{M}(T_3)$ are morphisms in $\mathcal{P}$.

The core property of any functor is that it preserves composition. That is, for any composable morphisms f and g in $\mathcal{T}$, the following must hold:

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

This equality is not derived but is an axiom in the definition of a functor. Our formalization of Meta Prompting posits that such a structure-preserving map exists. The proposition is therefore a direct consequence of this definition. The significance is that if we can decompose a complex task T into a sequence of simpler tasks (e.g., $g \circ f$), we can construct the meta-prompt for T by composing the meta-prompts for the simpler tasks ($\mathcal{M}(g) \circ \mathcal{M}(f)$). This guarantees the modularity and reusability of prompt structures. $\square$

A.4 The Monadic Framework for RMP

As introduced in Section 4, Recursive Meta Prompting (RMP) is modeled by a monad $(\mathcal{M}_\mathcal{P}, \eta, \mu)$ on the category of prompts $\mathcal{P}$. Here, we provide the commutative diagrams for the monad laws, which ensure that the process of recursive self-refinement is mathematically sound.

Monad Laws. The triple $(\mathcal{M}_\mathcal{P}, \eta, \mu)$ must satisfy the following coherence laws:

- *Left Identity:* $\mu \circ \mathcal{M}_\mathcal{P}\eta = \mathrm{id}_{\mathcal{M}_\mathcal{P}}$. Applying a refinement after lifting a prompt into the monadic context is the same as just refining it.
- *Right Identity:* $\mu \circ \eta\mathcal{M}_\mathcal{P} = \mathrm{id}_{\mathcal{M}_\mathcal{P}}$. Lifting a refined prompt and then flattening it has no effect.
- *Associativity:* $\mu \circ \mathcal{M}_\mathcal{P}\mu = \mu \circ \mu\mathcal{M}_\mathcal{P}$. The order of flattening nested refinements does not matter.

The following diagrams illustrate these laws.

$$\begin{tikzcd} \mathcal{M}_\mathcal{P} \arrow[r, "\eta\mathcal{M}_\mathcal{P}"] \arrow[dr, "\mathrm{id}_{\mathcal{M}_\mathcal{P}}"'] & \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P} \arrow[d, "\mu"] \\ & \mathcal{M}_\mathcal{P} \end{tikzcd} \qquad \begin{tikzcd} \mathcal{M}_\mathcal{P} \arrow[r, "\mathcal{M}_\mathcal{P}\eta"] \arrow[dr, "\mathrm{id}_{\mathcal{M}_\mathcal{P}}"'] & \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P} \arrow[d, "\mu"] \\ & \mathcal{M}_\mathcal{P} \end{tikzcd}$$

$$\begin{tikzcd} \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P} \arrow[r, "\mu\mathcal{M}_\mathcal{P}"] \arrow[d, "\mathcal{M}_\mathcal{P}\mu"'] & \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P} \arrow[d, "\mu"] \\ \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P} \arrow[r, "\mu"'] & \mathcal{M}_\mathcal{P} \end{tikzcd} \tag{A.1}$$

A.5 Proof of Stability (Theorem 4.1)

Proof. The theorem states that the recursive refinement process is stable and associative. This property is not derived from first principles but is an axiom guaranteed by our choice to model Recursive Meta Prompting (RMP) with a monad $(\mathcal{M}_\mathcal{P}, \eta, \mu)$.

A core requirement for any monad is that its multiplication (or join) operation, μ, must be associative. This is expressed by the *associativity law*, which states that the following diagram must commute for any object $P \in \mathcal{P}$:

$$\begin{tikzcd} \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P}(P) \arrow[r, "\mu_{\mathcal{M}_\mathcal{P}(P)}"] \arrow[d, "\mathcal{M}_\mathcal{P}(\mu_P)"'] & \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P}(P) \arrow[d, "\mu_P"] \\ \mathcal{M}_\mathcal{P}\mathcal{M}_\mathcal{P}(P) \arrow[r, "\mu_P"'] & \mathcal{M}_\mathcal{P}(P) \end{tikzcd}$$

This diagram translates to the equation $\mu_P \circ \mu_{\mathcal{M}_\mathcal{P}(P)} = \mu_P \circ \mathcal{M}_\mathcal{P}(\mu_P)$. Let's interpret the two paths from $\mathcal{M}^3_\mathcal{P}(P)$ to $\mathcal{M}_\mathcal{P}(P)$:

- **Path 1 (Top-Right):** The path $\mu_P \circ \mu_{\mathcal{M}_\mathcal{P}(P)}$ corresponds to first collapsing the two outermost refinement layers (from $\mathcal{M}^3_\mathcal{P}$ to $\mathcal{M}^2_\mathcal{P}$) and then collapsing the final two layers.
- **Path 2 (Bottom-Left):** The path $\mu_P \circ \mathcal{M}_\mathcal{P}(\mu_P)$ corresponds to first collapsing the two innermost refinement layers (from $\mathcal{M}^2_\mathcal{P}(P)$ to $\mathcal{M}_\mathcal{P}(P)$ inside the global $\mathcal{M}_\mathcal{P}$ context) and then collapsing the resulting outer layers.

The monad law requires that these two paths are equivalent. In the context of RMP, this means that given a prompt that refines a prompt that refines a prompt, the order of "flattening" or executing these refinements does not matter. The final, singly-refined prompt is the same regardless of the collapse order. This axiom directly provides the proof of stability and associativity for the RMP process. □

B Additional Prompt Examples

""" 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 8: 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.

Key Elements of Meta Prompting for Complex Reasoning:

1. **Complex Problem Decomposition**: Break down intricate problems into smaller, manageable subproblems to enable systematic problem solving.
2. **Detailed Preliminary Content**: Supply essential background information and foundational concepts to set the stage for problem resolution.
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 $\square$).

```
<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 9: An illustration of Meta Prompting for Complex Reasoning.

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 Additional Experimental Details

C.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 B).

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

C.2 Solving MATH Problems

D Multi-Modal Meta Prompting

The structured nature of Meta Prompting, grounded in the principles of type theory, provides a robust framework for extending foundation models to multi-modal settings. Just as type systems in programming ensure that different data types are handled correctly, a meta prompt enforces a clear, syntactic structure on the AI's inputs and outputs. This is particularly valuable when interacting with diverse data like text, images, audio, or even physical sensors, as it minimizes ambiguity and ensures that the model interacts with tools and data in a consistent, predictable manner.

This "typed" approach is essential for complex, real-world applications where an AI must process and synthesize information across different modalities. For example, an XML-based meta prompt can define a rigid schema that an LLM must follow, specifying slots for different types of reasoning steps and data.

D.1 A Framework for Multi-Modal Interaction

Extending Meta Prompting to a multi-modal context involves defining a schema that can handle varied data formats while integrating them coherently. The primary challenges are data representation and inter-modal synthesis. A multi-modal meta prompt addresses this by creating explicit, typed "slots" for different data streams.

For instance, a task might require the AI to analyze a 3D model ('.obj'), listen to an accompanying audio description ('.mp3'), and read a textual specification ('.txt'). A meta prompt can structure this complex

```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 12: 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.

input, guiding the model to process each modality in a specific order and synthesize the information to generate a cohesive output. Figure 17 illustrates how such a schema can be conceptualized.

By defining a clear, typed structure for both inputs and outputs, this framework enables models to perform sophisticated cross-modal analysis—for example, correlating a visual diagram with textual instructions. This structured approach provides a scalable and reliable foundation for building the next generation of versatile, multi-modal AI systems.

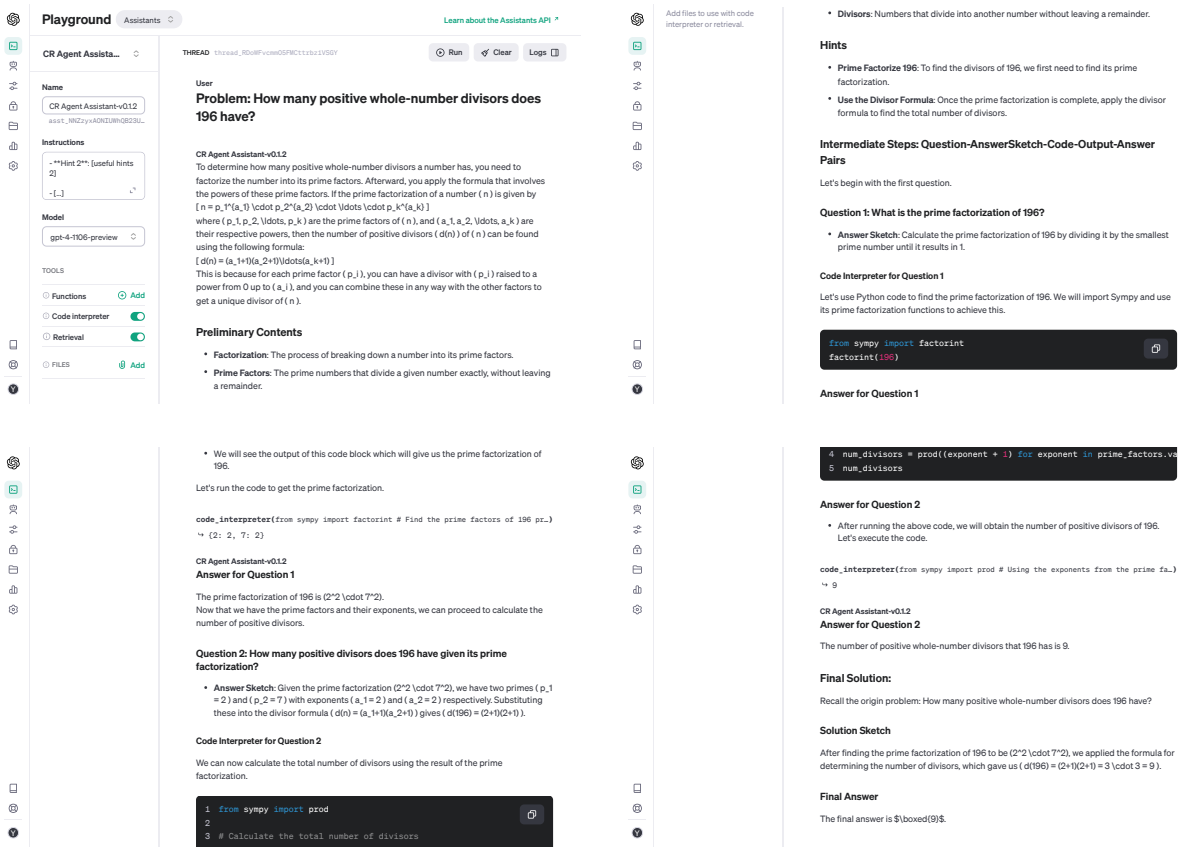

Figure 13: Experiments using the MP-CR Agent within the OpenAI Assistant for solving a MATH (Hendrycks et al., 2021) problem.

```
<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>
</problem_definition>
<solution_approach>
</solution_approach>
<preliminary_contents>
</preliminary_contents>
<hints>
</hints>
<intermediate_steps>
</intermediate_steps>
<final_solution>
<solution_sketch>
</solution_sketch>
<code_for_solution>
</code_for_solution>
<final_answer>
</final_answer>
</final_solution>
</problem_structure>
</syntax>
```

Figure 14: 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 15: The system meta prompt for MP-CR, comprising both the [SystemMetaPrompt] and the [StructureMetaPrompt].

```
<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>
  <problem_structure>
    <problem_definition>
      </problem_definition>
    <preliminary_contents>
      </preliminary_contents>
    <hints>
      </hints>
    <intermediate_steps>
      </intermediate_steps>
    <final_solution>
      <solution_sketch>
        </solution_sketch>
      <code_for_solution>
        </code_for_solution>
      <final_answer>
        </final_answer>
    </final_solution>
  </problem_structure>
</syntax>
```

Figure 16: An example of a meta prompt using an XML schema. This structure enforces type-like constraints on the output, making it ideal for frameworks like Guidance (Lundberg et al., 2023) and for extension to multi-modal data.

```
<task_schema>
    <input_slots>
        <data type="image/png" name="problem_diagram">
            </data>
        <data type="audio/mp3" name="verbal_instructions">
            </data>
        <data type="model/obj" name="object_model">
            </data>
    </input_slots>
    <output_schema>
        <synthesis type="text/markdown" name="analysis_summary">
            </synthesis>
        <result type="video/mp4" name="solution_animation">
            </result>
    </output_schema>
</task_schema>
```

Figure 17: A conceptual schema for a multi-modal meta prompt. It defines typed slots for various input modalities and specifies the expected structure and types for the output, enabling complex inter-modal reasoning.