Chain of Methodologies: Scaling Test Time Computation without Training

Anonymous ACL submission

Abstract

Large Language Models (LLMs) often struggle with complex reasoning tasks due to insufficient in-depth insights in their training data, which are frequently absent in publicly available documents. This paper introduces the Chain of Methodologies (CoM), a simple and innovative iterative prompting framework designed to build structured reasoning processes by injecting human methodological insights, thereby enabling LLMs to perform long and effective reasoning for complex tasks. Assuming that LLMs possess certain metacognitive abilities, CoM leverages user-defined methodologies to stimulate the cognitive insights that LLMs have learned implicitly from training data. Experimental results indicate that CoM outperforms competitive baselines, highlighting the potential of training-free prompting methods as general solutions for complex reasoning tasks and the possibility of incorporating human-like methodological insights to bridge the gap to human-level reasoning.

1 Introduction

Recently, OpenAI's o1 (OpenAI, 2024) showcases the possibility of using a long chain of thoughts to improve the reasoning ability of Large Language Models (LLMs). During these long thoughts, OpenAI's o1 displays high-level cognitive abilities, such as problem decomposition, error recognition, and correction, which constantly steer the thoughts in the right direction. OpenAI confers o1 with such abilities through training with reinforcement learning.

This paper conducts novel research on investigating the possibility of allowing LLMs to have the same universal self-guiding ability as OpenAI's o1 in conducting long and structured reasoning in various domains by merely using prompts and without relying on instruction fine-tuning.

This new problem is challenging: while LLMs can be fine-tuned using a relatively large dataset

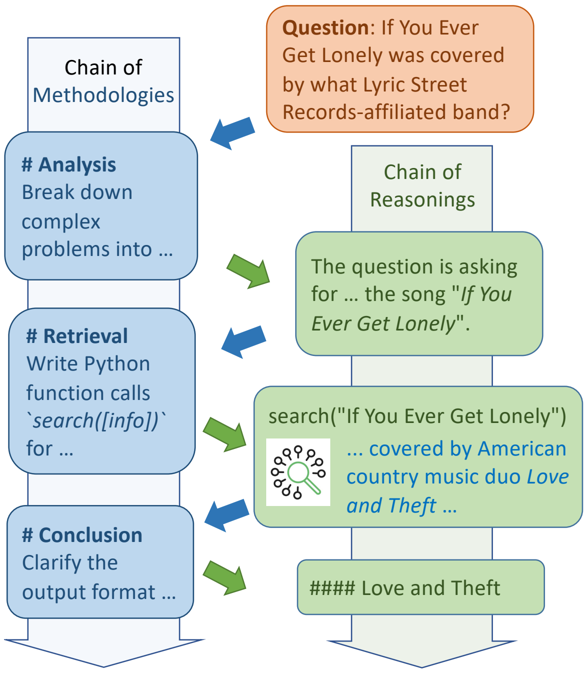

Figure 1: An illustration of our Chain of Methodologies reasoning process, where the generation of methodologies and reasoning interleaves. A methodology (in blue) is selected based on the historical reasoning status, while the next reasoning step (in green) is guided by the previously selected methodology.

for universal instruction-following ability, conventional prompts are mostly employed to induce instruction following in specific tasks with few-shot examples, due to the limitations of LLMs regarding context length and the accuracy of extracting information from long contexts. Therefore, pure prompting methods are seldom used in universal task solving, although prompting offers clear advantages over extensive fine-tuning in many aspects, such as low cost, fast deployment, high sample efficiency, and no risk of catastrophic forgetting or biasing the model with fine-tuning data.

Our approach is based on the discovery of metacognitive knowledge (the ability to reason about one's own reasoning processes) in the following prior work on using LLMs. Pedagogical research shows that improving human learners' metacognitive knowledge can enhance their reasoning capabilities, and positive results are reported for metacognitive prompts on introspective evaluation and self-reflection (Wang and Zhao, 2024). Microsoft directly uses "do not hallucinate" in their system prompt, aiming to reduce hallucination in Phi3 (et al, 2024), and (Didolkar et al., 2024) observes improved mathematical reasoning when allowing LLMs to determine the type of skill required to solve a problem, where the skill type is used to retrieve in-context examples.

This paper proposes Chain of Methodologies (CoM), a simple and task-agnostic iterative prompting technique aimed at achieving universal self-guiding ability for long and structured reasoning. Without relying on instruction fine-tuning, CoM uses methodology as a catalyst to stimulate LLMs to generate their next reasoning step based on the current reasoning history. Our basic idea is that, while LLMs struggle with complex reasoning tasks due to insufficient in-depth insights in the training data between problems and their respective solutions, we can enable smooth transitions between a problem and its solution steps by inserting an analysis of the methodology adopted before each solution step. We rely on the metacognitive knowledge in LLMs to select or generate a methodology that either explains or justifies the next solution steps.

Our CoM approach features (1) a list of methodologies written in our "when-what" format that facilitates selection based on the current reasoning history and then bridges it with the next reasoning step, and (2) a methodology-reasoning loop that iteratively chooses the next methodology to guide the next step of reasoning along a long and well-structured reasoning path. An example of the CoM reasoning path is illustrated in Figure 1 and two examples of our methodologies are listed in Figure 2.

Our contributions include a simple CoM framework and experimental studies. Our CoM framework produces faithful reasoning containing steps that are structured and explainable. It is also highly extensible in that users can enhance the framework by modifying the list of methodologies in plain text. We evaluated our task-agnostic CoM framework with two representative and challenging applications: mathematical reasoning and retrieval-augmented generation. Experimental results show that our CoM outperforms competitive baselines on these tasks with diverse LLMs.

2 Chain of Methodologies

2.1 Overview

We aim to use prompts to stimulate the high-level cognitive (metacognitive) knowledge in existing LLMs, enabling them to possess the same universal self-guiding ability as OpenAI's o1 to successfully carry out long and structured reasoning sequences across various domains. These prompts should be task-agnostic and effective in guiding thought processes, and we find prompts about methodology to be ideal candidates for this purpose. Methodology is a critical component of any discipline or field that requires a structured approach to understanding, problem-solving, or conducting research. It provides a framework that ensures tasks are executed consistently and effectively.

Our Chain of Methodologies (CoM) framework features a list of user-defined methodologies and a methodology-reasoning iteration. Each methodology provides a guideline for the next reasoning step based on the current reasoning history. The reasoning process of CoM alternates between a methodology selection step and a methodology-guided reasoning step, as illustrated in Figure 1.

List of Methodologies: Throughout human history, the accumulation and evolution of problem-solving methodologies have relied on fundamental processes such as trial and error, reflection, and self-correction based on problem-solving experiences. Unlike AlphaGo, which operates within a defined set of rules and a closed action space, human learning occurs in an open action space that is more complex and challenging to optimize. To navigate this complexity, we integrate human knowledge and experience related to task completion through established methodologies. Let $M = \{m^{(1)}, m^{(2)}, \cdots, m^{(n)}\}$ denote the list of n user-defined methodologies.

Reasoning iterations: CoM conduct a maximum number of K steps for each question Q. In step k where $1 \leq k \leq K$, we first prompt an LLM_m with prompt template P_p to select a methodology $m_k \in M$ based on the current reasoning history h_k:

$$m_k = LLM_p(M, Q, h_k, P_p) \quad (1)$$

, and then prompt an LLM_r with prompt tem-

```
## Analysis
- When: In step 1.
- What: Analyze the category and solution type of the question, list the key facts, variables, relations, constraints with their associated values, and clarify the required output format. Break down complex problems into simpler steps while maintaining critical context. Propose a sequence of methodologies necessary to tackle the remaining reasoning steps iteratively and explain how they are related to the final result.

## Retrieval
- When: Fact-based information from the internet is needed.
- What: Write 1-3 line(s) of Python function call(s) `search([information],topk=3)` for each information needed to retrieve. The function `search` has been defined and imported for you, which returns a text summary for the argument `information`. Place your code in a single ```python\n...``` code-block. Finally, accurately simulated the retrieved output by yourself.
```

Figure 2: Two example methodology definitions in our *when-what* format.

plate P_r to generate the next reasoning sequence r_k based on methodology m_k and history h_k:

$$r_k = LLM_r(M, Q, h_k, m_k, P_r) \tag{2}$$

, where the reasoning history contains all previous reasoning sequences $h_k = [r_1, r_2, \cdots, r_{k-1}]$. In this paper, we simply use the same instruction fine-tuned LLM for both LLM_m and LLM_r, which is frozen during the application of our framework.

2.2 Methodology Definition

Our emphasis is on a framework that utilizes a user-defined list of methodologies instead of studying the philosophy of finding a universally applicable set of methodologies, whose existence is a debated topic between universalism and contextualism. From a pragmatic perspective, we focus on how to represent each methodology to facilitate methodology selection and methodology-based reasoning.

To clarify the differences between method and methodology, a method is a specific technique or systematic procedure to accomplish a task, while a methodology encompasses the principles and rationale that guide the choice and use of methods. Each methodology in our user-defined list should specify two key fields: *when* and *what*. The *when* field indicates the applicable stage during the reasoning lifecycle of the methodology, as well as the context and factors that influence the choice of the methodology. The *what* field outlines the systematic approach, the action selection criteria, and the expected outcomes of the methodology.

Specifically, a methodology is defined in markdown format with three fields: (1) a name, (2) *when*: the situation and timing for application, and (3) *what*: the specification and details, including principles, tools, techniques, and procedures to use. Two methodology definitions are exemplified in Figure 2.

Next, we discuss various types of methodology definitions. We broadly divide them into three categories: *analysis*, *coding*, and *reflection*. The *analysis* methodologies inspire the LLM to organize information, including extracting facts, variables, relations, constraints, and objectives from the question; breaking down the initial question into manageable sub-problems; planning the sequence of actions to be taken; and summarizing, rearranging, and distilling the information obtained so far. The *coding* methodologies prompt the LLM to generate formal languages to be executed by their respective solvers to obtain accurate results and to use external tools, e.g., search engines, by calling predefined functions attached to the solvers. The *reflection* methodologies encourage the LLM to identify errors and provide constructive feedback through self-reflection or self-verification in order to adjust the approach and propose alternative strategies for the next steps. Figure 3 in the Appendix lists the task-agnostic methodology definitions we used in our experiments.

To conclude, the utilization of methodologies serves a multifaceted purpose: (1) providing human input methodologies to stimulate the metacognitive ability of LLMs to compensate for the lack of in-depth insights in their training data for conducting complex reasoning, (2) establishing a natural connection through explanation or justification between the current reasoning situation and its solution in the next step, and (3) providing an educated guess for the next step to avoid the search space of stochastic search methods such as MCTS (Qi et al., 2024) and RL (OpenAI, 2024; Snell et al., 2024; Zelikman et al., 2022) over a universal reasoning space that is much larger than those of games, such as AlphaGo.

Last, our framework is designed to be conveniently extended: users only need to update the list of methodology definitions in plain text to make it more comprehensive for general thinking or tailor it to a specific set of skills that accurately target a particular task.

2.3 Methodology-Reasoning Iterations

As illustrated in Figure 1, CoM alternates between prompting the LLM to generate the next methodology and the next methodology-based reasoning sequence for a maximum number of K iterations.

The first prompt instructs the LLM to select a methodology for the next reasoning steps, which concatenates the list of user-defined methodology definitions, the question, the history of previous methodology-based reasoning sequences, and an instruction with additional information about the reasoning stage and the output format for the LLM to select the most suitable methodology.

The second prompt contains all the information from the first prompt, as well as the methodology selected using the first prompt. It encourages the LLM to follow the guidance in the methodology while reasoning. The second prompt also requires the output to include the following items: (1) acknowledgment of the selected methodology by repeating its name, (2) a chain of thought reasoning process or a program that implements the methodology, and (3) a summarized result of the reasoning or a guessed output of the program.

Following the second prompt, a solver will be invoked to post-process the LLM output with the second prompt to facilitate the programming ability of LLMs (Chen et al., 2023). Currently, we only implement a Python interpreter, which is invoked once Python code blocks are detected in the output. This Python interpreter executes the code in a safe environment with several common packages imported. After execution, we replace the guessed output of the program in the LLM output with the stdout output of the code. This enables accurate reasoning on tasks that require computation, such as mathematical tasks, implementing the human methodology: "You should use a calculator for tasks that involve complex calculations." It also allows for various types of tool usage via the Python API during the reasoning process, including web search, knowledge base retrieval, and even invocation of other LLMs and manipulation of the LLM's own reasoning process (Cao et al., 2023).

Our Python interpreter executes code in a sandbox, which is a new process with a safe global scope where the code can only use a limited set of built-in functions and import from a given set of packages. We set a timeout of 1 minute for each process. We empirically found that a larger timeout does not bring significant performance improvements in our experimental tasks. Users can extend the tool-using capacity of the CoM framework by adding the corresponding methodology definition and implementing functions in the Python interpreter. For instance, if we want to enable Google search, we can add a methodology definition that specifies the existence of a function named "search" and the meaning of its arguments, and then implement and add this function to the global scope of the Python interpreter.

Our methodology selection prompt and methodology-based reasoning prompt are listed in Figure 4 in the Appendix.

3 Related Work

We draw inspiration from existing work that extensively explores different prompt designs, such as Chain-of-Thought (Wei et al., 2022), Least-to-Most (Zhou et al., 2023), Self-Consistency (Wang et al., 2023b), and Tree-of-Thoughts (Cao et al., 2023). Various methodologies have been proposed to enhance problem-specific performance, including question rephrasing, dividing subtasks, verification, symbolic grounding (Lyu et al., 2023; Xu et al., 2024a; Wang et al., 2023a; Zelikman et al., 2022; Wang et al., 2024), factuality and faithfulness verification for reasoning chains (Wang et al., 2024), as well as explicit separation of knowledge retrieval and reasoning steps to organize decision-making (Jin et al., 2024). These approaches are effective in designated contexts where the main objective is to enhance explicit reasoning capabilities in areas such as arithmetic, commonsense reasoning, and symbolic reasoning.

The key difference between our work and prior research on workflows/pipeline approaches (Anonymous, 2024b) is that prior studies use a set of predefined, hardcoded actions, while our work utilizes the metacognitive abilities of LLMs to select a methodology to derive actions step-by-step based on reasoning history. Additionally, we curated a set of task-agnostic methodologies targeting a broad range of unseen applications. We identify several contemporary works with various objectives that are closely related to ours, which we list below.

Buffer of Thoughts (Yang et al., 2024c) derives high-level guidelines from previously completed tasks and stores them in a buffer for reuse in the future, enabling learning from experience and improving efficiency by distilling level-2 slow thinking into level-1 fast thinking. This work differs from ours in that its high-level guidelines contain problem-specific reasoning chains or code templates targeting particular tasks, e.g., complex multi-query tasks.

Skill-based CoT (Didolkar et al., 2024) explores the metacognitive capabilities of LLMs in mathematical problem-solving. This work first labels each question with a corresponding skill. The labeled skills are then clustered to reduce redundancy. During inference, one of these skills is selected, and the skill-relevant examples are retrieved for in-context learning.

Induction-augmented generation (Zhang et al., 2023b) finds key concepts in the question and uses an inductive prompting template to extract their close concepts and common attributes to facilitate more accurate reasoning processes.

rStar (Qi et al., 2024) demonstrates a self-play mutual reasoning approach that significantly improves the reasoning capabilities of small language models without fine-tuning. This method uses a costly Monte Carlo Tree Search (MCTS) with a set of five reasoning-inducing prompts.

Training-based methods pursuing long chains of thought include STaR (Zelikman et al., 2022), which demonstrates that a language model iteratively trained on its reasoning history that leads to correct answers can solve increasingly difficult problems. In (Snell et al., 2024), small models are fine-tuned to perform more reasoning steps using reinforcement learning with beam search, lookahead search, and best-of-N verifiers. ReST-MCTS (Zhang et al., 2024) integrates process reward guidance with tree search MCTS to collect higher-quality reasoning traces. With similar ideas, AFlow (Anonymous, 2024a) iteratively refines task-specific workflows.

4 Experiments

4.1 Experiment Setup

We evaluate the effectiveness of two components in CoM: methodology selection and methodology-guided reasoning.

LLMs: We report experiment results conducted on a relatively larger Qwen2-72B-Instruct and a relatively smaller Qwen2.5-7B-Instruct (Yang et al., 2024a), as well as a recent open-source model reminiscent of OpenAI's o1, named Macro-o1 (Zhao et al., 2024), which is a fine-tuned Qwen2-7B-Instruct with a combination of the filtered Open-O1 CoT dataset (Team, 2024), Macro-o1 CoT dataset, and Macro-o1 Instruction dataset. We use the LLM API provided by Siliconflow (sil), with settings: max_tokens=1024, temperature=0.2, top_k=40, top_p=0.95, n=1.

Dataset: We evaluate CoM with the same set of methodology definitions (Figure 3) on the test splits of three datasets: AIME, GSM8K, and HotpotQA.

We use the "AIME Problem Set: 1983-2024" dataset (Zhang et al., 2023a). As part of the American Invitational Mathematics Examination (AIME), it includes a variety of problems, such as complex algebraic equations, geometric puzzles, and advanced number theory, aimed at testing mathematical understanding and problem-solving skills.

GSM8K (Cobbe et al., 2021) is a dataset of high-quality, linguistically diverse grade school math word problems that take between 2 and 8 steps of elementary calculations ($+ - \times \div$) to solve.

HotpotQA (Yang et al., 2018) is a comprehensive dataset for multi-hop, multi-step question answering. Each entry in the dataset includes a question, an answer, and supporting facts that indicate the titles of the relevant paragraphs providing the necessary information to answer the questions.

In the hard portion of the HotpotQA dataset, we simulate retrieval-augmented generation (Anonymous, 2025) experiments, where we only present the LLMs with the question, excluding the supporting facts and context paragraphs. When the LLMs generate code that calls the search with keywords, we employ fuzzy string matching to select the top-k most similar supporting facts to the keywords and return their corresponding context paragraphs as the retrieval results.

4.2 Baselines

We conduct all our experiments using zero-shot prompting, as current few-shot approaches utilize

Table 1: Results for AIME, GSM8K, and Hard Hotpot Tasks of a large LLM: Qwen2-72B-Instruct

	AIME	GSM8K	Hard Hotpot			
	Acc	Acc	EM	F1	Prec	Rec
CoM	23.15%	91.58%	0.38	0.484	0.471	0.587
Workflow	22.77%	92.27%	0.36	0.441	0.450	0.497
CoT	17.15%	93.10%	0.26	0.376	0.380	0.425
MCoT	14.58%	92.49%	0.27	0.372	0.380	0.390

Table 2: Results for AIME, GSM8K, and Hard Hotpot Tasks of a small LLM: Qwen2.5-7B-Instruct

	AIME	GSM8K	Hard Hotpot			
	Acc	Acc	EM	F1	Prec	Recall
CoM	25.4%	84.53%	0.33	0.417	0.420	0.504
Workflow	23.58%	84.00%	0.35	0.439	0.437	0.575
CoT	20.15%	91.51%	0.17	0.245	0.254	0.244
MCoT	17.58%	89.92%	0.15	0.216	0.222	0.215

task-specific few-shot examples, making it inappropriate to compare them with cross-domain methods. We compare CoM with three baselines that employ recent prompting techniques and utilize the same list of methodology definitions (Figure 3). We also compare CoM with Macro-o1 (Zhao et al., 2024), a recent open-source model reminiscent of OpenAI o1.

CoT (Wei et al., 2022) simply prompts the LLM to generate a chain of reasoning steps, and shows it the format of the final result.

MCoT provides the same list of methodology definitions as CoM, along with a CoT instruction to encourage the LLM to utilize these methodologies in an appropriate order to guide its reasoning. MCoT assesses whether the LLM can enhance reasoning with methodologies in a single-turn, non-interactive prompting scenario; it embodies ideas from prior work, such as Least-to-Most (Zhou et al., 2023) and Metacognitive-Prompting (Wang and Zhao, 2024).

Note that the first two baselines prompt the LLM once, and they are not allowed to generate code since a second prompt is required to synthesize the code output into the required format.

Workflow differs from CoM only in that it uses a fixed sequence of methodologies for each task. For each task, we selected the most frequently chosen methodology sequence by CoM (Table 4). Workflow prompts the LLM to reason over multiple turns, guided by each methodology in the sequence. For AIME and GSM8K, the methodology sequence is [Analysis, Coding, Variation, Conclusion], while for Hard Hotpot, the sequence is [Analysis, Retrieval, Conclusion]. Workflow embodies ideas in the literature, including Program-of-Thoughts (Chen et al., 2023), Cognitive Prompting (Wang and Zhao, 2024), workflow/pipeline (Jin et al., 2024; Anonymous, 2024b), RAG (Anonymous, 2025), etc.

4.3 Performance Comparison

Experimental results using the relatively large LLM Qwen2-72B-Instruct in Figure 1 show that our CoM outperforms other baselines on AIME and Hard Hotpot, with 38.5% accuracy and 28.7% F1 increases over CoT. However, on GSM8K, CoT is slightly better than CoM, probably due to its simplicity and benchmark leakage (Xu et al., 2024b). Generally, Workflow performs second best, while the results of MCoT show that the methodologies hardly benefit from a single prompt.

On AIME, the accuracy of CoM is 1.7% higher, and on Hard Hotpot, the F1 score of CoM is 9.8% higher than that of Workflow when using Qwen2-72B-Instruct. This indicates that CoM outperforms Workflow due to greater flexibility in methodology selection, which in turn provides evidence that the metacognitive ability in LLMs enables the correct selection of methodology sequences, and our methodology-based step-by-step reasoning is effective.

When using the relatively smaller LLM Qwen2.5-7B-Instruct, as shown in Figure 2, CoM still performs the best on the challenging math dataset AIME. However, in GSM8K, lower accuracies are observed in both CoM and Workflow, which seem to relate to the decrease in model capac-

Table 3: Performance of Macro-o1 and Qwen2.5-7B-Instruct on AIME Tasks and Hard Hotpot

	AIME	Hard Hotpot			
	Acc	EM	F1	Prec	Rec
Macro-o1					
CoT	14.47	0.12	0.20	0.20	0.27
MCoT	10.50	0.09	0.19	0.19	0.25
Qwen2.5-7B-Instruct					
CoT	20.15	0.17	0.25	0.25	0.24
MCoT	17.58	0.15	0.22	0.22	0.22
CoM	25.4	0.33	0.42	0.42	0.51

Table 4: Top 52.2% selected methodology sequences on AIME

	Methodology Sequence
22.0%	Analysis Coding Validation Conclusion
16.2%	Analysis Coding Conclusion
5.4%	Analysis Coding Validation Reflection Flexibility Conclusion
4.4%	Analysis Coding Validation Reflection Conclusion
4.3%	Analysis Coding Validation Reflection Flexibility Validation Conclusion

ity for long-range reasoning and instruction following. On Hard Hotpot, CoM is slightly lower than Workflow, indicating that the relatively smaller model is weaker than the larger ones in terms of metacognitive abilities, i.e., methodology selection.

Next, we compare Macro-o1 (Zhao et al., 2024), an LLM fine-tuned on datasets such as the Open-O1 CoT dataset (Team, 2024), with our prompting-only CoM. Table 3 shows that fine-tuning fails to improve Macro-o1 on AIME and Hard Hotpot, which likely indicates that the fine-tuning data is not universal enough.

4.4 Methodology Selection Patterns

We extract from the reasoning history saved during the experiment the methodology sequences selected by CoM and identify the most frequent patterns. The results on AIME with Qwen2-72B-Instruct are shown in Table 4, where we report the five most frequent patterns that dominate 52% of the answers from CoM.

In 22.0% of the cases, CoM prioritized a well-structured approach of first analyzing, then generating and executing code, which is followed by result validation and conclusion. In 16.2% of the cases, the model seems to be confident about its code and skips the validation step. For the other 3 cases, more steps for error corrections are invoked, suggesting errors may occur during validation. The most frequent methodology sequences generated by CoM indicate that LLMs possess the metacognitive ability to plan their own reasoning steps during the reasoning process.

4.5 Ablation Study

We study the relative importance of each component of our CoM, including the Python interpreter, and each of the methodologies we used. Here, in contrast to excluding the *code* methodology, which prevents CoM from generating code, removing the Python interpreter still allows the LLM to generate code, but the LLM then needs to guess the output of the code by itself without an interpreter. Our ablation study is conducted with Qwen2.5-7B-Instruct.

As listed in Table 5, the interpreter is very important for both tasks, which shows that the code output guessed by the LLM without using the code interpreter for both math calculation and knowledge retrieval is unreliable. Secondly, for hard math problems in AIME, systematic analysis of the data and constraints in the problem is vital for the correctness of the reasoning. For AIME, all methodologies we provided are useful, each contributing to a 7-10% improvement in accuracy. In AIME, we disable retrieval for experimental simplicity. For Hard Hotpot, where reasoning relies on retrieved information, retrieval is clearly the most important methodology.

4.6 Error Analysis

Errors made in CoM are conventional LLM errors such as hallucination, misunderstanding, and instruction-following errors. We manually inspected the first 10 error cases in CoM on the GSM8K dataset. We found that in most of these cases, methodology selection is not perfect. Three error cases are due to hallucination, where the wrong answers are given directly without the necessary calculation process. Two cases are due to translation errors from natural language to math; for example, "born early" is translated to a reduction in age. Three cases are due to language understanding errors; for instance, "restart downloading" is understood as "continue downloading", and "every second" is understood as "from the second". In one error case, the initial calculation is correct, but then a validation step causes an error because the

Table 5: Ablation Study Results for COM Method

CoM	AIME (%)	Hard Hotpot
No Ablation	25.4	0.4174
- Interpreter	14.1 (-44.5%)	0.25 (-40.2%)
- Analysis	18.7 (-26.6%)	0.38 (-8.4%)
- Coding	23.3 (-8.3%)	0.38 (-8.8%)
- Retrieval	-	0.22 (-46.8%)
- Validation	23.9 (-7.2%)	0.4 (-3%)
- Reflection	22.8 (-10.5%)	0.38 (-9%)
- Synthesis	23 (-9.3%)	0.4 (-3%)

Table 6: Average Speed of Experiments in Seconds per Iteration (Multiplied by 50)

	AIME	GSM8K	Hard Hotpot
Macro-o1			
CoT	96.0	33.5	19.0
MCoT	84.5	42.5	20.0
Qwen2.5-7B-Instruct			
CoM	91.0	34.0	50.5
Workflow	36.0	18.0	21.5
CoT	19.0	5.0	3.5
MCoT	19.0	8.0	3.5

LLM believes "servings" must be an integer. In one error case, the LLM generates more than one code block, although the methodology definition contains an instruction to generate a single standalone code block.

4.7 Efficiency

We examine the inference efficiency in terms of total inference time for speed and the number of inferences for cost. Table 6, shows that the speed of CoM is around 5 times that of CoT in AIME and 7 times in Hard Hotpot. However, CoM is comparable to the fine-tuned model, which generates longer reasoning traces with a lower speed per token.

Table 7 compares the number of prompts made by CoM with those made by Workflow. Note that CoM prompts twice per methodology-reasoning iteration. The results show that although we set the maximum iteration $K = 8$, CoM stops at a smaller number of steps on average, generates more reasoning steps for the harder AIME problems, and a smaller number of steps for the easier GSM8K problems.

4.8 Summary of Experiments

The experiments on complex mathematical problems (AIME and GSM8K) and multi-hop question answering (HotpotQA) evaluate the effectiveness of CoM in methodology selection and guided reasoning using a 72B, a 7B LLM, and a fine-tuned LLM for structured reasoning.

Results show that our CoM is effective in improving the performance of two challenging tasks over baselines that embody recent prompt engineering approaches. This result supports our hypothesis that we can use a training-free solution that integrates human methodological insights to enhance the performance of LLMs in complex reasoning tasks.

Table 7: Average Number of Prompts

	AIME	GSM8K	Hard Hotpot
CoM	2×5.76	2×3.99	2×5.98
Workflow	4	4	3

Methodology selection patterns reveal that CoM effectively generates reasonable methodology sequences, which guide its reasoning in the right direction. Error analysis identifies that common LLM issues contribute to the majority of errors made by CoM. Finally, the ablation study confirms that the methodologies we employed are critical for solving complex reasoning tasks.

5 Conclusion and Future Work

In this paper, we introduced the Chain of Methodologies (CoM), an innovative iterative prompting framework aimed at enhancing the reasoning capabilities of LLMs for complex tasks by simulating metacognitive processes. Our approach enables LLMs to utilize user-defined methodologies, allowing for more effective navigation of complex reasoning tasks without extensive retraining. Through rigorous experimentation, we demonstrated CoM's effectiveness and adaptability across various applications, including mathematical reasoning and retrieval-augmented generation.

An interesting area for future work will be to investigate the automatic search for optimal methodologies and reasoning chains via explorations and evaluations similar to (Yang et al., 2024b), from which we elicit sets of methodologies and their application examples to construct prompts for better reasoning performance.

Limitations

Due to constraints in computational resources, we conducted experiments using a few LLMs with free API calls on three applications across two domains: mathematical reasoning and retrieval-augmented reasoning. Future work can expand on this foundation with additional LLMs and applications.

The methodologies included in our framework are not exhaustive. This presents an opportunity for future research to enhance the list by incorporating diverse strategies to improve the CoM framework's adaptability.

Ethical Statement

This work fully complies with the ACL Ethics Policy. We declare that there are no ethical issues in this paper, to the best of our knowledge.

References

https://siliconflow.cn/.

Anonymous. 2024a. AFlow: Automating agentic workflow generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

Anonymous. 2024b. Chain of ideas: Revolutionizing research in idea development with LLM agents. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

Anonymous. 2025. Inference scaling for long-context retrieval augmented generation. In *The Thirteenth International Conference on Learning Representations (ICLR)*.

Shulin Cao, Jiajie Zhang, Jiaxin Shi, Xin Lv, Zijun Yao, Qi Tian, Lei Hou, and Juanzi Li. 2023. Probabilistic tree-of-thought reasoning for answering knowledge-intensive complex questions. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12541–12560. Association for Computational Linguistics.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

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

Aniket Rajiv Didolkar, Anirudh Goyal, Nan Rosemary Ke, Siyuan Guo, Michal Valko, Timothy P Lillicrap, Danilo Jimenez Rezende, Yoshua Bengio, Michael Curtis Mozer, and Sanjeev Arora. 2024. Metacognitive capabilities of LLMs: An exploration in mathematical problem solving. In *AI for Math Workshop @ ICML 2024*.

Marah Abdin et al. 2024. Phi-3 technical report: A highly capable language model locally on your phone. *Preprint*, arXiv:2404.14219.

Mingyu Jin, Weidi Luo, Sitao Cheng, Xinyi Wang, Wenyue Hua, Ruixiang Tang, William Yang Wang, and Yongfeng Zhang. 2024. Disentangling memory and reasoning ability in large language models. *Preprint*, arXiv:2411.13504.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329, Nusa Dua, Bali. Association for Computational Linguistics.

OpenAI. 2024. Learning to Reason with LLMs.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. In *Arxiv*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *Preprint*, arXiv:2408.03314.

O. Team. 2024. https://github.com/open-source-o1/open-o1.

Jianing Wang, Qiushi Sun, Xiang Li, and Ming Gao. 2024. Boosting language models reasoning with chain-of-knowledge prompting. In *The 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4958–4981, Bangkok, Thailand. Association for Computational Linguistics.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023a. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Yuqing Wang and Yun Zhao. 2024. Metacognitive prompting improves understanding in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1914–1926, Mexico City, Mexico. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Jundong Xu, Hao Fei, Liangming Pan, Qian Liu, Mong-Li Lee, and Wynne Hsu. 2024a. Faithful logical reasoning via symbolic chain-of-thought. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13326–13365, Bangkok, Thailand. Association for Computational Linguistics.

Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. 2024b. Benchmarking benchmark leakage in large language models. *arXiv preprint arXiv:2404.18824*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zhihao Fan. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2024b. Large language models as optimizers. In *The Twelfth International Conference on Learning Representations*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. 2024c. Buffer of thoughts: Thought-augmented reasoning with large language models. *arXiv preprint arXiv:2406.04271*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488. Curran Associates, Inc.

Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. 2024. Rest-mcts*: Llm self-training via process reward guided tree search. Thirty-eighth Conference on Neural Information Processing Systems (NeurIPS).

Xingyuan Zhang, Philip Becker-Ehmck, Patrick van der Smagt, and Maximilian Karl. 2023a. Action inference by maximising evidence: Zero-shot imitation from observation with world models. In *Thirty-seventh Conference on Neural Information Processing Systems (NeurIPS)*.

Zhebin Zhang, Xinyu Zhang, Yuanhang Ren, Saijiang Shi, Meng Han, Yongkang Wu, Ruofei Lai, and Zhao Cao. 2023b. IAG: Induction-augmented generation framework for answering reasoning questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1–14. Association for Computational Linguistics.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. 2024. Marco-o1: Towards open reasoning models for open-ended solutions. *Preprint*, arXiv:2411.14405.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Appendix

Our list of methodologies is displayed in Figure 3, and our methodology-selection and methodology-based reasoning prompts are listed in Figure 4.

Analysis
- When: In step 1.
- What: Analyze the category and solution type of the question, list the key facts, variables, relations, constraints with their associated values, and clarify the required output format. Break down complex problems into simpler steps while maintaining critical context. Propose a sequence of methodologies necessary to tackle the remaining reasoning steps iteratively and explain how they are related to the final result.

Retrieval
- When: Fact-based information from the internet is needed.
- What: Write 1-3 line(s) of Python function call(s) `search([information],topk=3)` for each information needed to retrieve. The function `search` has been defined and imported for you, which returns a text summary for the argument `information`. Place your code in a single ```python\n...``` code-block. Finally, accurately simulated the retrieved output by yourself.

Coding
- When: Coding is necessary.
- What: A standalone Python snippet with structural and systematic reasoning in comments, using the **print** function to output the result. Place your code in a single ```python\n...``` code-block. Finally, accurately simulated the output of your Python snippet by yourself without using a computer.

Validation
- When: A temporary or a final the result is resulting from a previous reasoning step.
- What: Identify the result, and analyze its correctness from a different angle. You may write a test-case function that prints True/False to validate the result and then simulate its output.

Reflection
- When: An error is detected or validation fails.
- What: Analyze the reasoning steps to identify errors and provide constructive self-critic or feedback for improvement.

Flexibility
- When: The previous step fails to obtains the expected result or when a reflective or critic feedback is available.
- What: Adjust the approach based on insights gained and propose alternative strategies for the next steps.

Conclusion
- When: A confident final answer is available, or in the last step.
- What: Clarify the output format required by the question. Compile the reasoning process and generate the answer in the required format.

Figure 3: Our list of methodologies.

````
# List of Existing Methodologies

{example_methodologies}

{historical_reasoning_steps}

# Question
- {question}

# Step Instruction
- You are currently on {cur_step} out of a total of a maximum of {max_steps} steps to solve the question. Please select the most suitable methodology from the list, considering the reasoning history and the timing for applying each methodology. Try to diversify your choices to advance the resolution of the question, and avoid redundant or repeated methodologies. Select and output a single methodology succinctly in the following format, without any additional text:

## [Methodology name]
- When: [The timing for applying the methodology]
- What: [The characteristics and details of the methodology]
````

````
# List of Existing Methodologies

{example_methodologies}

{historical_reasoning_steps}

# Question
- {question}

# Selected Methodology

{selected_methodology}

# Step Instruction
- You are currently on {cur_step} out of a total of a maximum of {max_steps} steps to solve the question. Please conduct an innovative next reasoning step for the question, using the selected methodology "{selected_methodology_name}" based on the current reasoning history. Please output your reasoning step succinctly in the following format, without any additional text:

- Methodology: [Name of the selected methodology]
- Reasoning/Code: [A structural and systematic reasoning step that uses the selected methodology and improves on your past reasoning steps. Or a Python snippet with such reasoning inside its comments in a ```python\n...``` code-block.]
- Result: [A short result of your reasoning, or an accurate simulated outpout from the code.]
````

Figure 4: Our methodology-selection prompt (left) and methodology-based reasoning prompt (right).