Buffer of Program: Enhancing Large Language Models Reasoning Through Retrieval of Trustworthy Programs

Anonymous ACL submission

Abstract

In the absence of external knowledge, large language models face difficulties in obtaining effective background information when dealing with complex reasoning tasks, which poses challenges to the models in understanding the task context and generating accurate solutions. We propose the Buffer of Program (BoP), an innovative framework that enhances LLM reasoning by storing and retrieving high-quality, validated program examples. BoP includes a dynamic buffer management system, a validation mechanism, and a retrieval-based problem context enhancement approach to provide models with robust contextual information and reasoning cues. Through extensive experiments on diverse reasoning datasets and mathematical datasets, BoP demonstrates significant performance improvements over existing state-of-the-art methods, with accuracy gains reaching as high as 120%. Moreover, BoP enhances model transparency and interpretability, offering a promising step toward trustworthy and systematic reasoning with LLMs.

1 Introduction

In recent years, Large Language Models (LLMs) have emerged as a transformative force in the field of artificial intelligence(Roberts et al., 2024), demonstrating unprecedented capabilities in the understanding and generation of natural language(Karanikolas et al., 2023). However, effective utilization of these models in complex reasoning tasks requires the development of sophisticated inference methods(Hao et al., 2024). These methods can be broadly classified into two categories: planning-based approaches and tool-invocation techniques. Planning-based methods, encompassing strategies such as Chain of Thought (CoT)(Wei et al., 2022), Buffer of Thought (BoT)(Yang et al., 2024b), Tree of Thought (ToT)(Yao et al., 2023), Graph of Thought (GoT)(Besta et al., 2024), Meta-Chain of Thought (Meta-CoT)(Wan et al., 2025), Least-to-Most (L2M)(Zhou et al., 2023), Self-Refine(Madaan et al., 2023), and Plan-and-Solver(Wang et al., 2023), focus on enhancing the internal reasoning mechanisms of LLMs by introducing structured and hierarchical processes(Yang et al., 2025). These approaches aim to improve the coherence, logical flow, and precision of reasoning by breaking down complex problems into manageable steps(Sodhi et al., 2023), dynamically adjusting reasoning paths(Wang et al., 2024a), and iteratively refining solutions(Madaan et al., 2023). In contrast, tool-invocation techniques, including Prompt Optimization for Tool Invocation (POT)(Chen et al., 2022), Program-Aided Language (PAL)(Gao et al., 2023), Chain of Code(Li et al., 2024), Codeplan(Wen et al., 2025), and Code Prompting Elicits Conditional Reasoning (Puerto et al., 2024), leverage the integration of external tools and resources to augment the model's capabilities(Zhuang et al., 2023). These methods improve the precision and efficiency of the inference process by using specialized tools and code snippets to perform specific tasks(Joel et al., 2024).

However, despite the significant advances made by these existing methods, they still face several limitations that hinder their effectiveness in complex reasoning tasks(Feng et al., 2025). Planning-based approaches, while effective in structuring the reasoning process, often suffer from the challenge of maintaining coherence and logical consistency over multiple steps(Wei et al., 2025), especially when dealing with highly complex and dynamic problems. The hierarchical nature of these methods can sometimes lead to rigid reasoning paths that are difficult to adapt to new or unexpected scenarios(Yang et al., 2025). Additionally, the iterative refinement process in methods such as Self-Refine(Madaan et al., 2023) and Plan-and-Solver(Wang et al., 2023) can be computationally expensive and time consuming, limiting their prac-

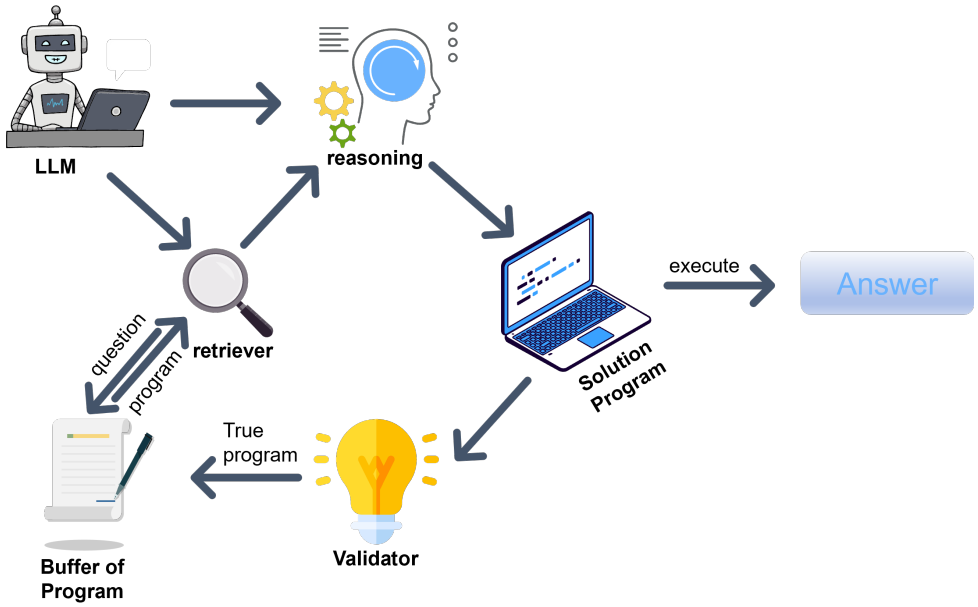

Figure 1: Buffer of Program(BoP)

ticality in real-time applications(Mirzadeh et al., 2024).

On the other hand, tool invocation techniques(Shen, 2024), although powerful in leveraging external resources(Fu et al., 2024), rely heavily on the availability and quality of these tools(Chen et al., 2024b). The integration of external tools can introduce additional complexity and potential errors(Sun et al., 2024), especially if the tools are not well aligned with the specific requirements of the reasoning task(Chen et al., 2024a). Moreover, these methods often focus on improving the precision of specific tasks rather than improving the overall reasoning capabilities of the model(Yuan et al., 2024), which can limit their applicability to a wider range of problems(Xu et al., 2025).

To overcome the limitations of these methods, we propose the Program Buffer, a high-quality program example enhanced reasoning framework. It stores the correct program code and its description generated by large models during the reasoning process, and retrieves the most relevant problems along with their program code and descriptions as contextual examples in subsequent reasoning to enhance the reasoning capabilities of the models. Specifically, our method has the following features:

- We propose a novel program - enhanced reasoning framework, buffer of program (BoP), which automatically fills high - quality program examples during the reasoning process of large models.
- We ensure the high quality of the programs in the buffer through validation of the programs generated during the reasoning process.
- Extensive experiments on key reasoning benchmarks show that our framework has been significantly improved. The accuracy has been increased, and the reasoning process has become clearer and more interpretable.

2 Related Work

Planning Based Approaches Planning-based approaches have garnered significant attention in recent years due to their ability to enhance the reasoning capabilities of Large Language Models (LLMs) through structured and hierarchical processes. The Chain of Thought (CoT) method(Wei et al., 2022) has been widely studied, in which LLMs are urged to generate step-by-step reasoning chains to solve complex problems. This approach has been shown to improve the logical coherence and accuracy of reasoning by breaking down problems into smaller, more manageable steps. Building on CoT, the Buffer of Thought (BoT)(Yang et al., 2024b) introduces a buffer mechanism to dynamically adjust reasoning paths, allowing the model to explore multiple solution paths concurrently. The Tree of Thought (ToT)(Yao et al., 2023) further extends this concept by constructing a tree-like structure of reasoning paths, allowing the model to explore a broader solution space and select the most

Question: Let's play a game called 24. You'll be given four integers, and your objective is to use each number only once, combined with any of the four arithmetic operations (addition, subtraction, multiplication, and division) and parentheses, to achieve a total of 24. Input: 4 7 8 8

PoT without examples:

```
from itertools import permutations, product
def find_solution(numbers):
    ops = ['+', '-', '*', '/']
    for nums in permutations(numbers):
        for op1, op2, op3 in product(ops, repeat=3):
            expr = f"(({nums[0]} {op1} {nums[1]} {op2} {nums[2]} {op3} {nums[3]}))"
            try:
                if eval(expr) == 24:
                    return expr
            except ZeroDivisionError:
                continue

    return 'No solution found.'
numbers = [4, 7, 8, 8]
print(find_solution(numbers))
```

BoP with a correct program example:

```
from itertools import permutations, product

def find_expression(nums):
    operations = ['+', '-', '*', '/']
    for perm in permutations(nums):
        for ops in product(operations, repeat=3):
            exprs = [
                f"(({perm[0]}{ops[0]}{perm[1]}){ops[1]}({perm[2]}{ops[2]}{perm[3]}))",
                f"(({perm[0]}{ops[0]}{perm[1]}){ops[1]}{perm[2]}){ops[2]}{perm[3]}",
                f"{perm[0]}{ops[0]}(({perm[1]}{ops[1]}{perm[2]}){ops[2]}{perm[3]}))"
            ]
            for expr in exprs:
                try:
                    result = eval(expr)
                    if abs(result - 24) < 1e-9:
                        return f"Expression: {expr} = {result}"
                except ZeroDivisionError:
                    continue

    return "No valid expression found"

nums = [1, 3, 7, 10]
print(find_expression(nums))
```

BoT of the thought templates extracted from incorrect answers:

```
from itertools import permutations, product
import operator
def evaluate_expression(a, b, op):
    if op == 'div' and b == 0:
        return None
    return operator.__dict__[op](a, b)
def find_solution(nums):
    ops = {
        'plus': operator.add,
        'minus': operator.sub,
        'mul': operator.mul,
        'div': operator.truediv
    }
    for perm in permutations(nums):
        for ops_combination in product(ops.keys(), repeat=3):
            expr = f"(((perm[0]} {ops_combination[0]} {perm[1]}) {ops_combination[1]} ({perm[2]} {ops_combination[2]} {perm[3]}))"
            try:
                result = evaluate_expression(
                    evaluate_expression(
                        evaluate_expression(perm[0], perm[1], ops_combination[0]),
                        perm[2],
                        ops_combination[1]
                    ),
                    perm[3],
                    ops_combination[2]
                )
                if abs(result - 24) < 1e-6:
                    return expr
            except (ZeroDivisionError, TypeError):
                continue
    return None

nums = [4, 7, 8, 8]
solution = find_solution(nums)
if solution:
    print(f"A solution is: {solution}")
else:
    print("No solution found.")
```

Figure 2: Buffer of Program can dynamically acquire and utilize trustworthy program templates.

promising paths. Similarly, the Graph of Thought (GoT)(Besta et al., 2024) utilizes a graph structure to represent reasoning processes, facilitating more flexible and interconnected reasoning. The Meta-Chain of Thought (Meta-CoT)(Wan et al., 2025) incorporates meta-learning techniques to adapt reasoning strategies based on the problem context, enhancing the model's generalizability. The Least-to-Most (L2M) approach(Zhou et al., 2023) focuses on incrementally refining solutions by starting with simpler subproblems and gradually addressing more complex aspects. The Self-Refine method(Madaan et al., 2023) iteratively refines the reasoning process by evaluating and correcting the previous steps, thus improving the general accuracy. Lastly, the Plan-and-Solver approach(Wang et al., 2023) separates the reasoning process into planning and execution phases, where the model first generates a plan and then executes it to solve the problem. These planning-based methods collectively aim to enhance the internal reasoning mechanisms of LLMs, making them more effective in handling complex reasoning tasks.

Tool Invocation Techniques Tool invocation techniques have emerged as a powerful strategy to enhance the reasoning capabilities of LLMs by integrating external tools and resources. These methods leverage the strengths of specialized tools and executable code snippets to perform tasks beyond the native capabilities of LLMs. For instance, Prompt Optimization for Tool Invocation (PoT)(Chen et al., 2022) focuses on optimizing the prompts used to invoke external tools, ensuring their effective and efficient use. Program-Aided Language (PAL)(Gao et al., 2023) integrates programming languages and code snippets to improve the precision and efficiency of the inference process. The Chain of Code approach (Li et al., 2024) generates chains of code to solve complex problems step by step, while Codeplan (Wen et al., 2025) introduces a planning mechanism for code generation. Additionally, code prompting elicits conditional reasoning(Puerto et al., 2024), which explores the use of conditional prompts to elicit specific reasoning abilities from text and code models. However, these methods often face limitations such as the non-executable nature of pseudocode, which restricts access to the core computational and tool-invocation capabilities of code (Wang et al., 2024b). Other works generate executable code by fine-tuning (Gou et al., 2024; Lu et al., 2024) or prompting (Chen et al., 2022; Gao et al., 2023; Xiao et al., 2024) for interpreter-based answering, but remain confined to narrow domains due to the challenges in disentangling implicit reasoning signals from noisy, ambiguous information for abstract, complex input problems. Some approaches combine LLMs with logic programming (LP) languages (Pan et al., 2023; Ye et al., 2023; Zhou et al., 2024), but these are limited by the unique syntax and interaction

protocols of each LP environment.

3 Methodology

In this section, we detail our Buffer of Program (BoP) method. BoP aims to enhance the reasoning capabilities of large language models (LLMs) by storing and retrieving high-quality program code. Specifically, BoP first retrieves the most relevant problems and their corresponding programs and descriptions from the program buffer as contextual examples to aid reasoning when processing each specific task. Subsequently, the new programs generated by the model during the reasoning process are validated, and if they pass the validation, they are stored in the buffer along with the problem descriptions for future use.

3.1 Problem Retrieval

Most of the existing prompting methods for complex problems fail to automatically provide the model with effective external knowledge, which poses challenges to the model when handling these tasks. To overcome this limitation, BoP introduces a retrieval-based problem context enhancement mechanism. Specifically, BoP employs a retriever that calculates the similarity between the current task and the problems stored in the buffer to retrieve the most relevant problems and their corresponding programs and descriptions. The retrieved problems and their programs are integrated into the model's input prompt as contextual examples, thereby providing the model with additional background information and reasoning cues. This method not only helps the model better understand the context of the task but also guides the model in the right direction, thereby improving the accuracy and efficiency of reasoning.

$$p^* = \arg\max_{p_i \in P} S(q, p_i)$$

Let p^* denote the problem retrieved from the problem library that is most similar to the current problem, p_i denote the i-th problem in the problem library, P represent the set of problems in the buffer, q represent the current problem to be solved and S represent the similarity function.

3.2 Program-Enhanced Reasoning

The retrieved problems and their programs are integrated into the model's input prompt as contextual examples to aid reasoning. This method allows the model to refer to the program logic and structure of previously solved similar problems when tackling new tasks. Specifically, when generating a new program, the model can draw on the key steps and logical structures in the retrieved programs, thereby improving the quality and accuracy of the generated program. Moreover, this context-enhancing mechanism also helps the model better understand the requirements and constraints of the task, thereby generating program code that is more in line with the task's needs. During the reasoning process, the model first parses the contextual examples in the input prompt to extract key information and logical structures. Then, the model combines this information with the specific requirements of the current task to generate program code for the current task. In this way, the model can not only use previous experience to solve new tasks but also make appropriate adjustments and optimizations according to the characteristics of the current task. This program-enhanced reasoning method not only improves the model's reasoning ability but also enhances the model's adaptability and flexibility, enabling it to better handle a variety of complex tasks. To generate the solution code c_{gen} for the current task q , we combine the current task q with the most relevant retrieved problem p^*, its description D^*, and its program C^* to construct the input prompt. The solution code is then generated by the language model M as follows:

$$c_{\text{gen}} = M\left(\text{Prompt}(q, (p^*, D^*, C^*))\right)$$

Here, C^* denotes the retrieved program, D^* denotes the description of the retrieved program, and$M(\cdot)$ represents the language model that generates new program code based on the provided context.

3.3 Program Validation

Program validation is a crucial component of the BoP method, ensuring that newly generated programs meet high standards of accuracy and reliability before being stored in the buffer. In this approach, we utilize a secondary large language model (LLM), known as the Validator Model, to conduct a comprehensive assessment of the generated programs, including syntactic correctness and logical consistency.

The Validator Model operates by receiving the newly generated executable program as input and performing a series of evaluations. These evalua-

tions are designed to assess the program's syntactic structure and logical coherence. The Validator Model then outputs a validation score that reflects the overall quality of the program. Only programs that achieve a validation score above a predefined threshold are considered valid and subsequently stored in the buffer for future use.

The validation function $\text{Validate}(p)$ is defined as:

$$\text{Validate}(p) = \text{Executable}(P) \quad \text{and} \quad (\text{Score}(P) > \delta)$$

where $\text{Executable}(\cdot)$ indicates whether the program is executable, $\text{Score}(\cdot)$ represents the model's assessment score for the program's syntax and logic, and δ denotes the predefined threshold.

Through the validation function, the Validator Model generates a comprehensive validation score that reflects the overall quality of the program. This comprehensive validation approach ensures that only high-quality and reliable programs are added to the buffer, thereby continuously enhancing the reasoning capabilities of the BoP method. This mechanism ensures that the programs stored in the buffer are of high quality and reliability, providing robust contextual examples for future tasks and enhancing the overall reasoning performance of the BoP method.

3.4 Buffer Management

The buffer update mechanism is dynamic, constantly updating and expanding based on the resolution of new tasks. Whenever the model successfully solves a new task and generates a validated program, the program and its corresponding problem description are stored in the buffer. This dynamic update mechanism not only ensures that the programs stored in the buffer are always of high quality and reliable but also allows the buffer to continuously expand, providing the model with more contextual examples and reasoning cues. Through this approach, BoP can continuously enhance the model's reasoning capabilities, enabling it to perform more effectively when handling various complex tasks.

The algorithmic formula for the buffer update mechanism is as follows. This formula primarily describes how the buffer dynamically updates:

$$P_{\text{new}} = P \cup \{p_{\text{new}} \mid \text{Validate}(p_{\text{new}}) = \text{True}\}$$

Here, P_{new} represents the updated program buffer; P represents the current program buffer; p_{new} represents the new program generated by the model during the current inference process; $\text{Validate}(p_{\text{new}})$ is a validation function used to determine whether the program p_{new} meets the criteria for inclusion in the buffer.

By integrating problem retrieval, program-enhanced reasoning, and buffer management, the BoP method effectively enhances the reasoning capabilities of LLMs. This approach not only provides the model with rich contextual information and reasoning cues but also ensures that the model maintains high standards of quality and reliability when generating new programs. Through this method, BoP offers an effective solution for improving the reasoning performance of LLMs on complex tasks.

4 Experiments

Datasets To evaluate the efficacy of our proposed approach, we selected six datasets: Game of 24(Yao et al., 2023), Checkmate in One(Srivastava et al., 2023), Word Sorting(Suzgun et al., 2022), Date Understanding(Suzgun et al., 2022), GSM8K(Cobbe et al., 2021), and GSM-hard(Gao et al., 2023). (1) The Game of 24 requires forming an arithmetic expression equaling 24 using four given numbers exactly once, assessing numerical reasoning and combinatorial thinking. (2) Checkmate-in-One involves identifying a single move to achieve checkmate in a given chessboard configuration, testing strategic thinking and spatial reasoning. (3) Word Sorting tasks participants with arranging words based on specific criteria, such as alphabetical order or semantic categories, evaluating lexical manipulation and categorization abilities. (4) DateUnderstanding is a task that involves inferring dates from natural language descriptions, performing arithmetic operations on dates, and utilizing global knowledge such as the number of days in February, assessing temporal reasoning and knowledge integration. (5) GSM8K is a dataset of grade-school math word problems, requiring the model to understand and solve complex arithmetic problems, evaluating mathematical reasoning and problem-solving skills. (6) GSM-hard(Gao et al., 2023) is a harder version of GSM8K(Cobbe et al., 2021) with larger numbers that are less common, further challenging the model's mathematical reasoning and computational capabilities. These datasets collectively

Model	Logical Reasoning				Mathematical Reasoning	
	Game of 24	Checkmate In One	Word Sorting	Data Understand	GSM8K	GSM_Hard
Qwen2.5-7B	6.1%	3.2%	13.2%	52.3%	91.6%	59.3%
CoT	8.2%	4.4%	15.6%	53.5%	90.2%	54.5%
Self-Refine	11.2%	46.8%	85.6%	58.2%	90.9%	57.3%
Plan-and-Solve	31.6%	3.6%	22.4%	53.3%	90.1%	58.0%
PoT	32.6%	1.4%	91.6%	41.7%	88.7%	71.0%
BoT	41.2%	78.0%	84.4%	47.4%	89.7%	65.5%
BoP	90.8%	83.5%	95.4%	64.8%	93.5%	75.5%
(Relative Gain)	120.4%	7.1%	4.1%	11.1%	2.1%	6.3%

Table 1: Performance of different methods on the dataset and the improvement of BoP over other best methods.

provide a comprehensive evaluation framework, covering mathematical reasoning, logical deduction, linguistic processing, temporal reasoning, and knowledge integration.

Base Models and Baselines We implement BoP on two model scales: qwen2.5-7B-instruct(Yang et al., 2024a) and qwen2.5-1.5B-instruct(Yang et al., 2024a). To comprehensively evaluate our proposed method, we selected a range of baselines that reflect various reasoning paradigms: (1) Zero-Shot, which utilizes LLMs without any prompting or training on the evaluation benchmarks; (2) Standard Chain-of-Thought (CoT)(Wei et al., 2022), guiding LLMs to analyze natural language problems and generate intermediate reasoning steps; (3) Plan-and-Solve(Wang et al., 2023), a two-stage approach that first formulates a high-level plan and subsequently generates responses; (4) Self-Refine(Madaan et al., 2023), an iterative strategy that alternates between generating answers and refining them based on self-assessment feedback; (5) Program of Thought(PoT)(Chen et al., 2022), employing LLMs to read natural language problems and generate programs as intermediate steps; and (6) Buffer of Thought (BoT)(Yang et al., 2024b), which enhances reasoning by maintaining a buffer of previously generated thoughts or intermediate results. All methods were assessed in a few-shot setting (1–3 shots) without training on the evaluation benchmarks, ensuring consistent configurations across all baselines.

4.1 Comprehensive Analysis of Experimental Results

Experimental Results Comparison When comparing the performance of different methods, we focus on the comparison between the BoP method and other best-performing methods. As shown in Table 1.

On the reasoning datasets, BoP shows improvement over other best methods, especially on the Game of 24 dataset. on the Game of 24 dataset, the accuracy of the BoP method reaches 90.8%, which is significantly higher than the best-performing PoT method (32.6%) among other methods, with a relative gain of 120.4%. On the Checkmate in One dataset, the BoP method achieves an accuracy of 83.5%, compared to the best-performing Self-Refine method (46.8%) among other methods, showing a noticeable performance improvement with a relative gain of 7.1%. On the Word Sorting dataset, the BoP method accuracy is 92.8%, compared to the best-performing Self-Refine method (85.6%) among other methods, with a more moderate performance improvement and a relative gain of 1.3%. On the Date Understanding dataset, the BoP method achieves an accuracy of 64.8%, compared to the best-performing Self-Refine method (58.2%) among other methods, showing a more significant performance improvement with a relative gain of 11.1%.

The problems within reasoning datasets often exhibit a degree of correlation. Consequently, when the model successfully infers a correct program for a given problem, this program is subsequently employed as a contextual example to facilitate reasoning for subsequent problems. The inherent correlation among problems enables the correct program to serve as an effective heuristic, efficiently guiding the model's reasoning process for related tasks. This, in turn, enhances the model's overall accuracy by leveraging previously acquired knowl-

edge and solution patterns to inform and refine its subsequent inferences.

On the mathematical datasets, BoP also shows improvement compared to other methods. On the GSM8K dataset, accuracy of the BoP method is 93.5%, compared to the best-performing Self-Refine method (90.9%) among other methods, with a more moderate performance improvement and a relative gain of 2.1%. On the GSM-Hard dataset, the BoP method achieves an accuracy of 75.5%, compared to the best-performing PoT method (71.0%) among other methods, showing a more significant performance improvement with a relative gain of 6.3%. These results indicate that the BoP method demonstrates a significant performance advantage in handling various complex tasks, especially in tasks such as Game of 24 and Checkmate in One, where the performance improvement of the BoP method is particularly remarkable.

The performance improvement of BoP on mathematical datasets further demonstrates its generalizability. Even when there is a certain degree of discrepancy between the examples and the target problems, the correct program examples still exert a positive influence on the model. This suggests that BoP can effectively leverage the underlying patterns and structures from the examples to enhance the model's reasoning, despite the variations between the examples and the specific problems at hand.

Comparative experiments of models with different sizes The evaluation of the Buffer of Program (BoP) enhancement on the Game of 24 dataset provides compelling evidence of its transformative impact on the reasoning capabilities of Large Language Models (LLMs). Specifically, the results highlight the significant performance gains achieved by integrating BoP into the model architecture, as demonstrated in Figure 3.

The baseline models, qwen2.5-1.5b and qwen2.5-7b, achieved accuracies of 1.1% and 6.1%, respectively. These results underscore the inherent limitations of large language models in complex numerical reasoning tasks without any enhancements. The introduction of the Buffer of Thought (BoT) provided a notable improvement. However, the most substantial performance gains were observed when the Buffer of Program (BoP) was applied.

The qwen2.5-1.5b + BoP model achieved an accuracy of 50.0%, representing a significant

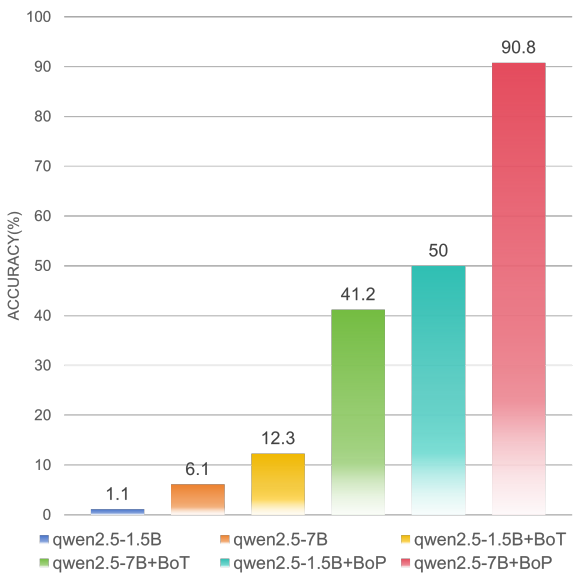

Figure 3: Evaluation of different benchmarks and models on the Game of 24 dataset

improvement over both the baseline and BoT-enhanced models. More impressively, the qwen2.5-7b + BoP model attained an accuracy of 90.8%, demonstrating the versatility of BoP across different model sizes.

The significant performance boost achieved by BoP can be attributed to its unique approach of validating and admitting high-quality program examples. By storing and retrieving verified program examples, BoP provides the model with a rich set of contextual information and reasoning cues, enabling it to generate more accurate and efficient solutions. This structured approach not only enhances the model's ability to solve complex numerical reasoning tasks but also shows its potential for generalization across a broader range of reasoning tasks.

Ablation study When the buffer is removed, the BoP method degrades to program reasoning without prompts, which causes the model to lose support for contextual examples when handling tasks. In this case, the accuracy of the model drops significantly. For example, on the Game of 24 dataset, the accuracy rate drops from 90.8% to 32.6%; on the Checkmate In One dataset, the accuracy rate drops from 83.5% to 1.4%. This shows that the program examples in the buffer play a crucial role in the model's reasoning, especially when dealing with complex tasks, where contextual examples can significantly improve the model's accuracy.

When the validation module is removed, the

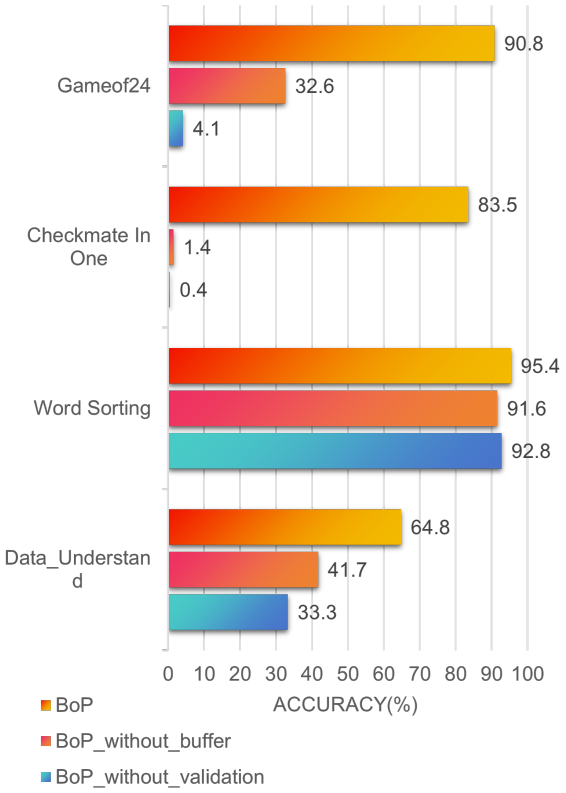

Figure 4: Ablation studies on four reasoning datasets using the qwen2.5-7B model regarding the removal of the buffer module and the validation module.

quality of the programs stored in the buffer drops significantly. Due to the lack of a validation mechanism, the programs stored in the buffer may contain errors or low-quality solutions, and these inaccurate program examples can have a negative impact on the model's reasoning when retrieved. For example, on the Game of 24 dataset, after removing the validation module, the accuracy rate drops from 90.8% to 4.1%; on the Checkmate In One dataset, the accuracy rate drops from 83.5% to 0.4%. This shows that the validation module plays a key role in ensuring the quality of the programs in the buffer, and high-quality program examples can effectively guide the model's reasoning process.

In addition, as the difficulty of the problems increases, the help that the retrieved program examples provide to the model's reasoning gradually weakens. On the Word Sorting dataset, even after removing the validation module, the accuracy of the model rate remains high (92.8%), which may be because this task is relatively simple and the model can solve the problem with its own reasoning ability. However, in more complex tasks (such as Game of 24 and Checkmate In One), the lack of high-quality program examples can cause a significant drop in the model's accuracy rate. This shows that in dealing with complex tasks, high-quality contextual examples are crucial for the model's reasoning, and the collaborative effect of the validation module and the buffer can significantly improve the model's reasoning performance.

5 Discussion

Limitations When the model encounters overly difficult tasks at the initial stage, it may fail to generate program code that meets the task requirements, and thus cannot effectively guide the subsequent reasoning process. Therefore, the improvement of BoP's reasoning ability is somewhat limited when dealing with highly complex tasks. Additionally, when the relevance between different tasks is low, even if there are a large number of program examples in the buffer, the model may not be able to retrieve highly relevant program examples, causing the algorithm to degrade into example-free program reasoning.

Future Directions Future research can optimize the reasoning model and the validation model in a dual-layer manner to enhance the reasoning model's semantic alignment capability with the high-quality program examples screened by the validation model. By matching the semantic information of these program examples with the current task requirements, the accuracy of reasoning can be improved.

Conclusion In this study, we introduce the Buffer of Program (BoP) method, a novel framework designed to enhance the reasoning capabilities of Large Language Models (LLMs) by leveraging high-quality program examples. BoP integrates a dynamic buffer management system, a validation mechanism, and a retrieval-based problem context enhancement approach to provide models with robust contextual information and reasoning cues. Through extensive experiments on diverse reasoning datasets, BoP has demonstrated significant performance improvements over existing state-of-the-art methods, representing a major advancement in enhancing the reasoning abilities of LLMs and offering a scalable solution to improve reasoning performance.

References

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna

Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and 1 others. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.

Sijia Chen, Yibo Wang, Yi-Feng Wu, Qingguo Chen, Zhao Xu, Weihua Luo, Kaifu Zhang, and Lijun Zhang. 2024a. Advancing tool-augmented large language models: Integrating insights from errors in inference trees. *Advances in Neural Information Processing Systems*, 37:106555–106581.

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

Yanfei Chen, Jinsung Yoon, Devendra Singh Sachan, Qingze Wang, Vincent Cohen-Addad, Mohammadhossein Bateni, Chen-Yu Lee, and Tomas Pfister. 2024b. Re-invoke: Tool invocation rewriting for zero-shot tool retrieval. *arXiv preprint arXiv:2408.01875*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Sicheng Feng, Gongfan Fang, Xinyin Ma, and Xinchao Wang. 2025. Efficient reasoning models: A survey. *arXiv preprint arXiv:2504.10903*.

Xiaohan Fu, Shuheng Li, Zihan Wang, Yihao Liu, Rajesh K Gupta, Taylor Berg-Kirkpatrick, and Earlence Fernandes. 2024. Imprompter: Tricking llm agents into improper tool use. *arXiv preprint arXiv:2410.14923*.

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

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. Tora: A tool-integrated reasoning agent for mathematical problem solving. *In The Eleventh International Conference on Learning Representations*.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, and 1 others. 2024. Llm reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. *arXiv preprint arXiv:2404.05221*.

Sathvik Joel, Jie JW Wu, and Fatemeh H Fard. 2024. A survey on llm-based code generation for low-resource and domain-specific programming languages. *arXiv preprint arXiv:2410.03981*.

Nikitas Karanikolas, Eirini Manga, Nikoletta Samaridi, Eleni Tousidou, and Michael Vassilakopoulos. 2023. Large language models versus natural language understanding and generation. In *Proceedings of the 27th Pan-Hellenic Conference on Progress in Computing and Informatics*, pages 278–290.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. 2024. Chain of code: Reasoning with a language model-augmented code emulator. *International Conference on Machine Learning*.

Zimu Lu, Aojun Zhou, Ke Wang, Houxing Ren, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2024. Mathcoder2: Better math reasoning from continued pretraining on model-translated mathematical code. *arXiv preprint arXiv:2410.08196*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2024. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. Logic-lm: Empowering large language models with symbolic solvers for faithful logical reasoning. *arXiv preprint arXiv:2305.12295*.

Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in text+ code llms. *arXiv preprint arXiv:2401.10065*.

John Roberts, Max Baker, and Jane Andrew. 2024. Artificial intelligence and qualitative research: The promise and perils of large language model (llm)'assistance'. *Critical Perspectives on Accounting*, 99:102722.

Zhuocheng Shen. 2024. Llm with tools: A survey. *arXiv preprint arXiv:2409.18807*.

Paloma Sodhi, SRK Branavan, Yoav Artzi, and Ryan McDonald. 2023. Step: Stacked llm policies for web actions. *arXiv preprint arXiv:2310.03720*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adri Garriga-Alonso, and 1 others. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on machine learning research*.

Jimin Sun, So Yeon Min, Yingshan Chang, and Yonatan Bisk. 2024. Tools fail: Detecting silent errors in faulty tools. *arXiv preprint arXiv:2406.19228*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, and 1 others. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Ziyu Wan, Yunxiang Li, Yan Song, Hanjing Wang, Linyi Yang, Mark Schmidt, Jun Wang, Weinan Zhang, Shuyue Hu, and Ying Wen. 2025. Rema: Learning to meta-think for llms with multi-agent reinforcement learning. *arXiv preprint arXiv:2503.09501*.

Jiapu Wang, Sun Kai, Linhao Luo, Wei Wei, Yongli Hu, Alan Wee-Chung Liew, Shirui Pan, and Baocai Yin. 2024a. Large language models-guided dynamic adaptation for temporal knowledge graph reasoning. *Advances in Neural Information Processing Systems*, 37:8384–8410.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*.

Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024b. Executable code actions elicit better llm agents. In *Forty-first International Conference on Machine Learning*.

Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. Plangenllms: A modern survey of llm planning capabilities. *arXiv preprint arXiv:2502.11221*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Jiaxin Wen, Jian Guan, Hongning Wang, Wei Wu, and Minlie Huang. 2025. Codeplan: Unlocking reasoning potential in large language models by scaling code-form planning. In *The Thirteenth International Conference on Learning Representations*.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, and 1 others. 2024. Chain-of-experts: When llms meet complex operations research problems. *In The Eleventh International Conference on Learning Representations*.

Hongshen Xu, Zihan Wang, Zichen Zhu, Lei Pan, Xingyu Chen, Lu Chen, and Kai Yu. 2025. Alignment for efficient tool calling of large language models. *arXiv preprint arXiv:2503.06708*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. 2025. Reasonflux: Hierarchical llm reasoning via scaling thought templates. *arXiv preprint arXiv:2502.06772*.

Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E Gonzalez, and Bin Cui. 2024b. Buffer of thoughts: Thought-augmented reasoning with large language models. *Advances in Neural Information Processing Systems*, 37:113519–113544.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822.

Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. 2023. Satlm: Satisfiability-aided language models using declarative prompting. *Advances in Neural Information Processing Systems*, 36:45548–45580.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Ren Kan, Dongsheng Li, and Deqing Yang. 2024. Easytool: Enhancing llm-based agents with concise tool instruction. *arXiv preprint arXiv:2401.06201*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2023. Least-to-most prompting enables complex reasoning in large language models. *In The Eleventh International Conference on Learning Representations*.

Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q Weinberger, and Yuhuai Wu. 2024. Don't trust: Verify–grounding llm quantitative reasoning with autoformalization. *In The Eleventh International Conference on Learning Representations*.

Yuchen Zhuang, Yue Yu, Kuan Wang, Haotian Sun, and Chao Zhang. 2023. Toolqa: A dataset for llm question answering with external tools. *Advances in Neural Information Processing Systems*, 36:50117–50143.