Introspection of Thought Helps AI Agents

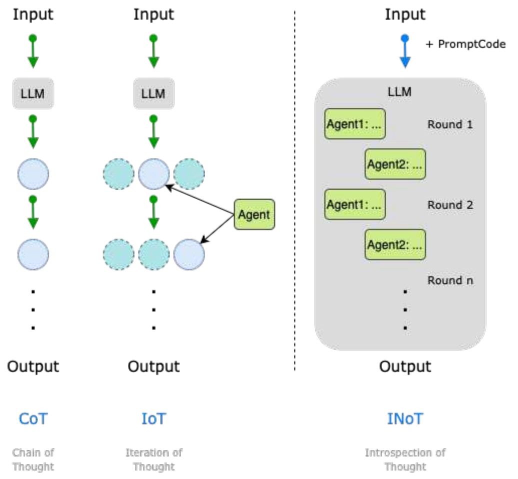
Haoran Sun[a,b,∗,1] **and Shaoning Zeng**[a,c,∗∗]

[a]Yangtze Delta Region Institute (Huzhou), University of Electronic and Science Technology of China, Huzhou, Zhejiang, China
[b]School of Information and Software Engineering, University of Electronic and Science Technology of China, Chengdu, Sichuan, China
[c]Zhejiang Chuangjiekedong Ltd., Huzhou, Zhejiang, China

Abstract. AI Agents rely on Large Language Models (LLMs) and Multimodal-LLMs (MLLMs) to perform interpretation and inference in text and image tasks without post-training, where LLMs and MLLMs play the most critical role and determine the initial ability and limitations of AI Agents. Usually, AI Agents utilize sophisticated prompt engineering and external reasoning framework to obtain a promising interaction with LLMs, e.g., Chain-of-Thought, Iteration of Thought and Image-of-Thought. However, they are still constrained by the inherent limitations of LLM in understanding natural language, and the iterative reasoning process will generate a large amount of inference cost. To this end, we propose a novel AI Agent Reasoning Framework with Introspection of Thought (INoT) by designing a new LLM-Read code in prompt. It enables LLM to execute programmatic dialogue reasoning processes following the code in prompt. Therefore, self-denial and reflection occur within LLM instead of outside LLM, which can reduce token cost effectively. Through our experiments on six benchmarks for three different tasks, the effectiveness of INoT is verified, with an average improvement of 7.95% in performance, exceeding the baselines. Furthermore, the token cost of INoT is lower on average than the best performing method at baseline by 58.3%. In addition, we demonstrate the versatility of INoT in image interpretation and inference through verification experiments.

1 Introduction

With the advancement of Large Language Models (LLM) and Multimodal LLMs (MLLM), from GPT-3 to modern models such as DeepSeek-V2.5 [6], GPT-4 [26], LLaVA [14] and Claude [1], multimedia reasoning tasks has undergone a significant transformation. LLMs are widely applied in various reasoning tasks [8, 37], particularly in mathematical reasoning [26], programming [8], text & image question answering [6, 37]. However, due to the limitations of the design and scale of LLMs themselves, they cannot handle very complex tasks alone [33]. AI Agent can help LLM solve more complex problems without post-training [12]. The basic core of AI Agent is the LLMs it constructs, which can conduct conversations, complete tasks, reason based on LLMs, and can demonstrate a certain degree of autonomous behavior [38]. The scale and design of the core LLM determine the initial capabilities and limitations of AI agents [33].



In order to improve the performance of LLM in AI Agent, prompt engineering plays an important role [28, 20, 24]. Prompt Engineering seamlessly integrates a pre-trained LLM into downstream tasks without updating model parameters by inputting designed instructions, and therefore guides the model to perform the expected behavior only through the given prompts [28]. Prompts can be natural language instructions that provide context to guide the model, or learning vector representations that activate relevant knowledge [22, 16]. AI Agents come with various built-in prompts to provide guidance, allowing LLM to perform more diverse reasoning for different tasks [17]. This enables AI Agents to complete more complex tasks. However, there are many ways to express the same operation in natural language. Different expressions can lead to bias in understanding in natural language, which can cause LLM not to think as expected [16].

To ensure that the final output of the AI Agent meets expectations, the reasoning frameworks are integrated into the AI Agent [12]. Based on the guidance of prompt engineering, they enhance the performance of an AI Agent through various iterative methods [12]. As shown in Figure **??**, the reasoning framework working in one single AI Agent, such as CoT, undergoes multiple chain iterative inferences in the AI Agent, reflecting and judging the results of each output of LLM until the results are considered reasonable [31]. Based on the inference of a single AI Agent, collaboration between multiple AI Agents performs better on various tasks and has become a promising method to enhance the capability of LLM, such as IoT, which allows multiple Agents to interact with each other and then

[∗] Email: 2022090916002@std.uestc.edu.cn.
[∗∗] Corresponding Author. Email: zeng@csj.uestc.edu.cn.
[1] Equal contribution.

iteratively generate results [24]. Through a continuous iterative dialogue, the results can be optimized. ProgCo utilizes a code-level self-correction mechanism to guide LLM for error detection and improvement through programmed feedback loops [29]. Although these methods have excellent performance in natural language tasks, they do not perform well in image interpretation and inference tasks [28]. Therefore, Image-of-Thought prompting guides the MLLMs to gradually perform visual inference by automatically designing image information extraction operations, in order to improve the accuracy and interpretability of MLLM in image inference tasks [37]. However, the above methods do not have great generality in multimedia inference tasks [16]. Meanwhile, although the reasoning framework improves the results of LLMs, this continuous iteration consumes a lot of token cost by constantly inputting and outputting to LLM [24, 20, 15].

To solve the above problem, we propose a new LLM-Read code in prompt (named as PromptCode), and a novel AI Agent reasoning framework based on this prompt, named as Introspection of Thought (INoT). INoT defines and designs the LLM-Read code PromptCode in the prompt, and then put the complete reasoning logic code written by PromptCode into the prompt and make LLM reason according to the code logic. By using PromptCode reasoning logic, a virtual multi-agent debate reasoning logic code is built and executed in LLM. Therefore, self-denial and reflection occur within LLM instead of outside LLM. Our main contributions are summarized as follows.

- INoT uses code-integrated prompt to optimize LLM performance in zero-shot reasoning tasks by making LLM reason according to the reasoning logic code in prompt.
- INoT puts a continuous iterative reflection process into the LLM internal through the guidance of PromptCode logic. Self-denial and reflection occur when reasoning within LLM, as shown in Figure **??**, so that the external does not need to interact with LLM multiple times, reducing the token cost of completing tasks using LLM.
- To evaluate INoT, Seven existing reasoning frameworks are selected as baselines and six datasets are selected as benchmarks in the areas of Math, Code, and QA tasks. Our INoT outperforms existing methods by an average of 11.6% on the benchmarks and the token cost of INoT is on average 58.3% lower than the best-performing method in baselines. In addition, we also demonstrate the versatility of INoT in image question answering tasks through validation experiments on three Image Q & A datasets.

2 Related Work

2.1 Prompt Engineering

Prompt engineering has become a key technique for enhancing the capability of LLM and MLLM [28]. While the scale and design of LLM define its initial abilities and limitations, prompts can refine the performance of an AI Agent without modifying model parameters [17, 31]. By strategically designing task-specific instructions, prompt engineering enables LLMs to generate targeted outputs, adapting to various tasks [28, 30]. Unlike traditional methods that require retraining or large-scale fine-tuning [7, 13], prompt engineering achieves task-specific improvements without extensive data or human input, offering a more efficient approach [13].

With the widespread application of AI Agents, research on prompt engineering continues to evolve, particularly in natural language reasoning and logic [28]. Initially, tasks were directly input into LLMs using the Input-Output (IO) approach, but due to the limitations of LLM, IO could only handle simple tasks. To address more complex tasks, Chain of Thought (CoT) prompts were introduced, mimicking human reasoning to guide LLMs [31]. However, CoT was effective only for the sufficiently advanced LLMs and lacked consistency between the reasoning process and the model predictions [30]. To improve this problem, Self-Consistent Chain-of-Thought Distillation (SCOTT) was developed, distilling a self-consistent CoT model from a much larger teacher model [30]. For even more complex reasoning, LogiCoT, an instruction tuning dataset, was introduced to enhance logical reasoning and trigger general reasoning abilities in LLMs [15]. Image-of-Thought prompt helps MLLMs better understand complex scenes by gradually extracting key visual information, optimizing the performance of MLLMs [37].

Furthermore, Iteration of Thought (IoT) framework was proposed to enhance the responsiveness of LLM by generating "thought-provoking" prompts for input queries and current iterative responses of LLMs [24]. Program of Thoughts (PoT) prompt significantly improves the accuracy and efficiency of numerical reasoning tasks by separating complex computational tasks and delegating them to external program interpreters, solving the problem of computational errors in traditional methods [4]. The Program-driven Self-Correction (ProgCo) method achieves self-correction of the model by automatically generating and executing verification programs, significantly improving the inference performance in complex tasks and reducing the misleading feedback of errors [29]. PoT and ProgCo combine programming ideas to optimize LLM, but they both generate real code and execute it in an external real environment, failing to make LLM truly programmatic inference. The above prompts are designed to optimize LLM or MLLM by constructing prompts for a certain task in natural language or images. There is no prompt that can universally optimize the performance of both LLM and MLLM. In addition, due to the limitations of understanding of natural language, LLM may not be able to strictly follow the guidance in natural language prompts for reasoning [28]. Therefore, in terms of guiding LLM reasoning, directly constructing the code for reasoning logic and putting it into the prompt, which will make the LLM strictly follow the code logic for reasoning and ensure the results more rigorous and accurate.

2.2 AI Agent

The advancement of LLMs has accelerated the progress of AI Agents [33, 12]. AI Agents extend beyond simple text generation by leveraging LLMs as their core computing engine, enabling dialogue, task execution, reasoning, and a degree of autonomy [18, 12]. Carefully designed prompts encode identity, instructions, authorization, and context to shape agent behavior [33]. While LLMs form the foundation of AI Agents, inference strategies and agent architectures are crucial [17]. Research has evolved from static, single-agent thought-chain prompts to dynamic, multi-agent dialogue systems [24, 31].

Some people focus on optimizing a prompt to enable LLMs to complete specific tasks, while others utilize different proxy architectures to optimize the performance of AI Agents. For example, an AI proxy library AgentLite was proposed, which simplified the innovation process of inference strategies, proxy architectures, and applications by providing a lightweight and user-friendly platform [17]. Instead of using static methods, a framework called Dynamic LLM-Powered Agent Network (DyLAN) was built, which collaborated on different tasks and domains with a dynamic communication structure for LLM-driven proxy collaboration, operating a two-stage

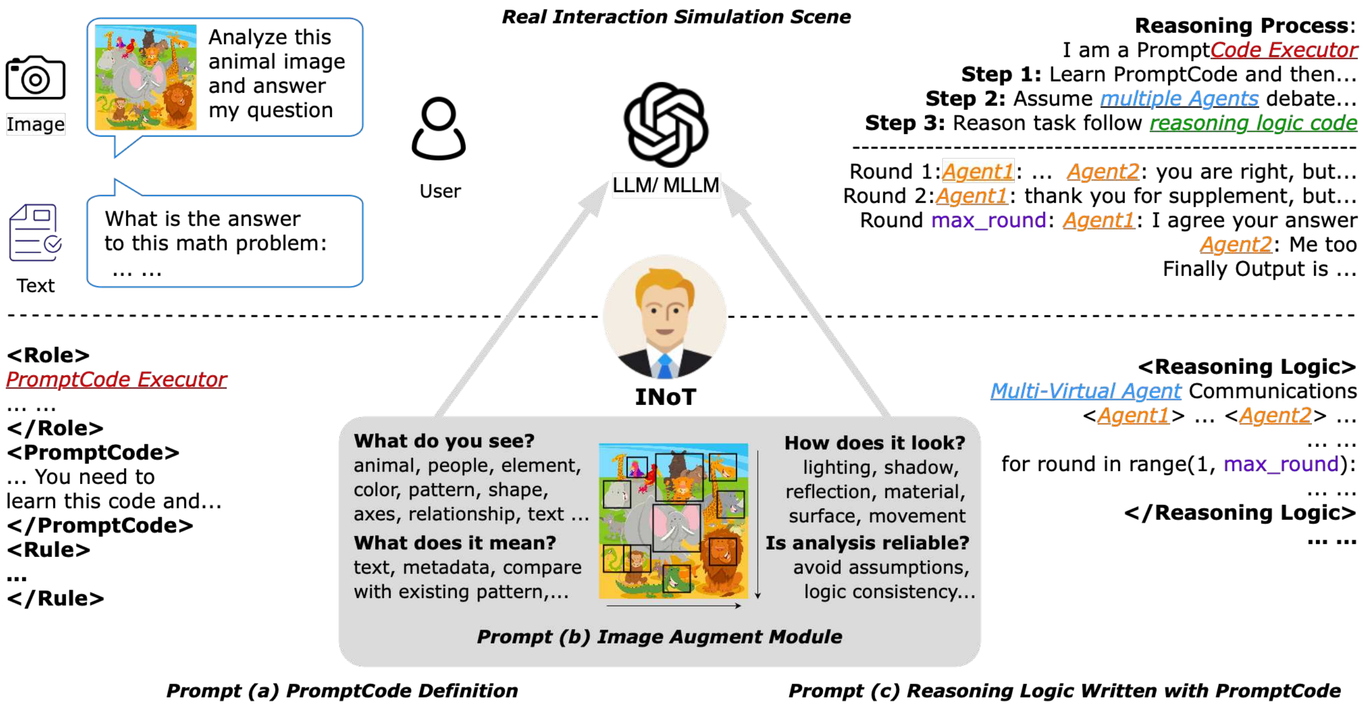

Real Interaction Simulation Scene
Image
Analyze this animal image and answer my question
Text
What is the answer to this math problem: … …
User
LLM/ MLLM
Reasoning Process:
I am a PromptCode Executor
Step 1: Learn PromptCode and then…
Step 2: Assume multiple Agents debate…
Step 3: Reason task follow reasoning logic code
Round 1:Agent1: … Agent2: you are right, but…
Round 2:Agent1: thank you for supplement, but…
Round max_round: Agent1: I agree your answer
Agent2: Me too
Finally Output is …
INoT
<Role>
PromptCode Executor
… …
</Role>
<PromptCode>
… You need to learn this code and…
</PromptCode>
<Rule>
…
</Rule>
What do you see?
animal, people, element, color, pattern, shape, axes, relationship, text …
What does it mean?
text, metadata, compare with existing pattern,…
How does it look?
lighting, shadow, reflection, material, surface, movement
Is analysis reliable?
avoid assumptions, logic consistency…
Prompt (b) Image Augment Module
<Reasoning Logic>
Multi-Virtual Agent Communications
<Agent1> … <Agent2> …
… …
for round in range(1, max_round):
… …
</Reasoning Logic>
… …
Prompt (a) PromptCode Definition
Prompt (c) Reasoning Logic Written with PromptCode

Figure 1: A multimedia inference task is combined with INoT Prompt and passed to LLM, then LLM internally processes the task according to the guidance of reasoning logic code in prompt, enabling self-denial and reflection occur within LLM instead of outside LLM.

paradigm: (1) team optimization, and (2) task resolution [18].

All the above methods interact with only one single LLM. In addition, multiple agents have been developed to communicate with each other for reasoning. For example, Iteration of Thought (IoT) framework, which enhanced the responsiveness of LLM by generating thought-provoking prompts based on input queries and the current iterative response of LLM [24]. However, the token cost of current methods designed for the Agent architecture was extremely high. The final result depended on iterative optimization outside the LLM or dialogue between multiple LLMs [24, 17, 20]. These reflection processes required continuous inputs and outputs from LLMs, which generated a mass of inference costs [12]. If there is a method that can incorporate external iterative and dialogue processes into the LLM, it will save a lot of token costs.

3 Method

We propose the Introspection of Thought (INoT) framework, which defines a LLM-read code, PromptCode, in the prompt, as shown in Figure 1. We use PromptCode to write reasoning logic code, and put the complete definition of PromptCode and reasoning logic code into the prompt, guiding LLM to strictly follow the code logic in the prompt for inference. This transfers the self-denial and reflection process of conventional AI Agents from outside LLM to inside LLM, thereby improving LLM inference performance while reducing token cost.

After investigating each prompt structure, we choose XML as the overall designing framework for prompts in INoT, using the XML structure to construct prompt. XML uses a hierarchical tree structure to organize data, dividing it into different levels and nodes through nested tags. This structure can clearly express the logical relationship between each part of prompt, providing LLM with an intuitive and easy-to-parse prompt framework, improving parsing efficiency of LLM, and reducing misunderstandings caused by non-standard or ambiguous prompt formats. At the same time, XML tags are self-descriptive, and tag names are usually defined based on the content or meaning of the section, which makes the section itself has strong semantic expression ability. When LLM processes prompt, it can quickly understand the meaning of each prompt fragment without relying on additional contextual information, thus significantly improving the accuracy and efficiency of its semantic understanding.

There are three modules in the INoT prompt, the PromptCode Definition Module, the Augment Module, and the Reasoning Module.

Listing 1: PromptCode Definition (XML as the Language)

```
<Role>
RoleName: PromptCode Executor
RoleDesc: You need to learn PromptCode,
a structured reasoning code defined below.
Follow the rules and execute reasoning logic
code strictly as written in <Reasoning Logic>.
</Role>
<PromptCode>
PromptCode is a structured reasoning code
 that explicitly defines logical steps
 to solve a given task. It is a hybrid of
 Python programming and natural language.
</PromptCode>
<Rule>
 Purpose of each module designed below:
<Image Augment>:Follow this when analyse
image, if no image, ignore this.
<Reasoning Logic>: The most important part,
 must reason task follow this line by line!
</Rule>
```

3.1 PromptCode Definition Module

PromptCode is a new program language defined in INoT prompt, specifically designed for LLM to read and understand. In order to better enable LLM to learn and understand, PromptCode is composed of a mixture of Python programming language and natural language. Unlike pseudocode, pseudocode is mainly designed for human understanding, while PromptCode is designed specifically to cater to semantic parsing ability and generation logic of LLM, in order to facilitate LLM reading and understanding.

Python features a concise syntax and clear code structure, enabling the efficient expression of complex task logic. Its code inherently contains rich semantic information, such as variable names, function names, and comments, which provides LLMs with additional contextual cues to better infer code intent. Meanwhile, natural language offers an intuitive and flexible means of conveying task background, goals, and requirements, allowing LLMs to grasp the overall task intention more effectively. Due to the linguistic proximity between Python and natural language, their integration is seamless and mutually reinforcing. While Python provides precise logical instructions, natural language supplements them with rich semantic context, enhancing the model's understanding of task details. In scenarios involving complex task descriptions or dynamic logic adjustments, natural language can often replace lengthy code blocks, expressing the same logic in a single sentence. This significantly improves the conciseness and interpretability of reasoning prompts for LLMs.

Integrating natural language into Python can further enhance the capabilities of LLMs. In prompt, natural language can describe some functions that are difficult to directly implement in Python in reality, which often have high abstraction, dynamism, or involve complex logical reasoning. Through the description of natural language, LLM can break through the limitations of traditional programming languages when understanding tasks, thereby generating more creative and adaptive solutions. Python code usually requires clear logic and structure to implement specific functions, but may face difficulties in dealing with highly abstract concepts. For instance, non-functional requirements like "scalability" and "robustness," or stylistic attributes of images, are challenging to encode directly in code. However, by expressing these concepts in natural language, LLMs can better grasp the core requirements and generate outputs aligned with them.

In PromptCode Definition, Rule enables LLM to understand the entire prompt clearly and efficiently. Rule provides a brief introduction to all modules in the prompt in advance, as shown in Listing 1, which can construct a clear reasoning framework and execution specifications in prompt, guiding LLM to follow specific logical structures for reasoning and decision-making in different task scenarios. It ensures that LLM can think systematically when dealing with complex tasks, avoid arbitrary speculation, and improve the accuracy, consistency, and controllability of reasoning.

Listing 2: Image Augment

```
<Image Augment>
Follow this module to analyze images.
<Basic Visual Understanding>
(What do you see?)
- Identify people, animals, key elements.
- Recognize colors, shapes, patterns, axes,
legends and spatial relationships.
- Detect text within images (if present)
</Basic Visual Understanding>
<Advanced Visual Analysis>
(How does it look?)
- Observe lighting, shadows, and reflections.
- Analyze textures, materials, and surfaces.
- Consider movement cues (if applicable)
</Advanced Visual Analysis>
<Context Awareness>
(What does it mean?)
- Use any accompanying text, metadata, or
task instructions to refine understanding.
- Compare the image with known patterns or
knowledge to improve accuracy.
- Recognize symbolic or cultural references
that may impact interpretation.
</Context Awareness>
<Inference and Verification>
(Is the analysis reliable?)
- Cross-check image details with available
textual descriptions.
- Avoid assumptions if something is unclear,
acknowledge uncertainty instead of guessing.
- Maintain logical consistency: ensure your
visual interpretation aligns with input task.
</Inference and Verification>
</Image Augment>
```

3.2 *Image Augment Module*

Image Augment module, as part of the INoT prompt, aims to guide MLLM to follow a systematic visual analysis framework when processing tasks that involve images to ensure precision, consistency, and logic of understanding, as shown in Listing 2. When the input task contains images, the Image Augment Module will play a role. This module uses clear instructions to enable MLLM to perform basic visual perception when parsing images, identify key elements, colors, shapes, patterns, axes, legends, spatial relationships, and textual information, thereby establishing a preliminary understanding of the image content. Subsequently, the prompt further requires LLM to conduct a more in-depth visual analysis, focusing on environmental factors such as lighting, shadows, and reflections, in order to infer time, material, and physical characteristics, and combine dynamic features to infer possible time-series information. At the same time, this module clearly instructs LLM to combine contextual information, including task descriptions, text annotations, and existing knowledge, to improve recognition accuracy, understand possible cultural or symbolic meanings, and avoid one-sided reasoning. In addition, this prompt module requires LLM to conduct strict logical verification during the reasoning process to ensure that the analysis results conform to the context and task objectives, avoid arbitrary guessing, and express appropriate uncertainty when the information is uncertain. Through this series of precise guidelines, the Image Augment module, as part of the INoT prompt, not only standardizes the way LLM processes images, but also enhances its depth of visual understanding, enabling it to provide more reliable and reasonable reasoning results in various task scenarios.

3.3 *Reasoning Module*

Reasoning Logic module in INoT prompt is designed to structure a multi-agent debate mechanism within the reasoning process of LLM, ensuring that responses are reflected and refined iteratively through critical evaluation and self-denial. The reasoning framework assumes the presence of two independent agents, Agent_A and Agent_B, who engage in a structured debate to iteratively refine their responses. The debate proceeds through several structured phases: initial reasoning, argument presentation, critique, rebuttal, and adjustment, ultimately leading to supervised validation and output of the final response.

Listing 3: Reasoning Code

```
<ReasoningLogic>
if image in task: import <Image Augment>
#Assume agents with independent reasoning.
Agent_A/B = DebateAgent(task)
result_A, thought_A = Agent_A.reason()
result_B, thought_B = Agent_B.reason()
# Set debate parameters
MaxRounds=10,Counter=0,agreement=False
While not agreement or Counter < MaxRounds:
Counter += 1
At each step, you must think like Agent_A
and then like Agent_B, responding as if
two intelligent debaters are reflecting.
#Step 1:Agents present their arguments
argument_A = Agent_A.reason()
argument_B = Agent_B.reason()
#Step 2: Critique each other's reasoning
(Debate Phase)
critique_A = Agent_A.critique(argument_B)
# A critiques B's reasoning
critique_B = Agent_B.critique(argument_A)
# B critiques A's reasoning
#Step 3: Agents respond to critiques
(Rebuttal Phase)
rebuttal_A = Agent_A.rebut(critique_B)
# A rebuts B's critique
rebuttal_B = Agent_B.rebut(critique_A)
# B rebuts A's critique
#Step 4: Adjust based on rebuttals
(Adjustment Phase)
result_A,thought_A=Agent_A.adjust(rebuttal_B)
# A refines based on B's rebuttal
result_B,thought_B=Agent_B.adjust(rebuttal_A)
# B refines based on A's rebuttal
#Step 5: Agreement Check
agreement = (result_A == result_B)
final_result = result_A if agreement
Output final_result without explanation.
</ReasoningLogic>
```

The whole reasoning logic code is shown in Listing 3. The process begins by incorporating an Image Augment module to enhance interpretability before reasoning. Then, LLM begins to initialize Agent_A and Agent_B, both tasked with reasoning independently on the given problem. Each agent produces an initial result along with the corresponding thought process. The debate mechanism is regulated through a maximum number of rounds (MaxRounds = 10) to prevent infinite loops, while a Counter ensures structured iteration. The core of the process lies in the debate, where in each round, both agents present their arguments, engage in mutual critique, respond through rebuttals, and adjust their reasoning accordingly. This iterative debate-rebuttal-adjustment cycle enables both agents to refine their thoughts based on opposing viewpoints, thereby reducing reasoning errors and inconsistencies. If both agents reach an agreement, their refined results converge and the result is finalized. Otherwise, the debate continues until either an agreement is reached or the iteration limit is met. The final response is then presented without additional explanations, reinforcing the structured reasoning approach and leading to more reliable output.

4 Experiments

4.1 Setup

Datasets. We select three types of tasks to evaluate the performance of INoT, including QA, Code and Math domains. Six public benchmarks are used for these tasks for our experiments. According to established practices for addressing these three tasks, the data is divided into validation and test sets using a 1 : 4 ratio [34]. Specifically, in QA domain, HotpotQA [32] and SQuAD 1.1 [25] are selected, where we randomly select 1,000 samples each. In Code domain, we use the Python dataset for HumanEval [3] and full dataset for MBPP [2]. In terms of Math, we use the entire dataset for GSM8K [5] to refer to existing work [31], and for MATH [10], 600 random problems with difficulty levels of 4 and 5 are selected from three types of problems (Combinatorics & Probability, Pre-algebra, Pre-calculus). Furthermore, in multimedia inference versatility verification, three image QA datasets are selected as benchmarks to refer to existing work [11], which are ScienceQA-IMG [21], LLaVA-Bench (COCO) [14], and LLaVA-Bench (In-the-Wild) [14].

Implementation Details. INoT utilizes different LLMs for tasks. LLMs including DeepSeek-V2.5 [6], DeepSeek-Chat in DeepSeek-V2 [6], Claude-3.5-sonnet [1], Gemma 2 [9], Qwen2.5-coder [23] and Llama3.2 [19] are employed as core LLMs. In addition, LLaVA [14], LLaMA-Adapter [35], and Multimodal Chain-of-Thought (MM-CoT) [36] are selected in multimedia inference versatility verification. All models are accessed via APIs. We set the temperature at 1 for DeepSeek-V2.5 and 0 for the other models. We set iteration rounds to 10 in INoT prompt. The environment configuration for experiment is: Pytorch 2.1.2 + cuda 12.1 as frame, Intel ® Core ™ i9-14900KF as CPU and NVIDIA GeForce RTX 4090 as GPU.

Baselines. In the experiments, we compare the results generated by INoT on the three types of tasks with other reasoning prompt engineering methods [28]. These methods include IO (direct LLM invocation), Chain-of-Thought (CoT) [31], Self-Consistency Chain-of-Thought (SCCOT) [30], Logical Cain-of-Thought (LogiCoT) [15], Tree-of-Thought (ToT) [20], guided iteration of thought (GIoT) and autonomous iteration of thought (AIoT) [24]. In these methods, LLMs are uniformly implemented using DeepSeek-V2.5 [6]. Additionally, in order to verify the universality and effectiveness of INoT in image inference tasks, we compared the results of four methods on benchmarks. These methods are IO, CoT, INoT without Image Augment Module (w/o IAM), and complete INoT on benchmarks.

Evaluation Metrics. For the QA tasks, we report the F1 Score as the metric. For the Code tasks, we report the pass@1 metric following established practices [27] to evaluate the accuracy of the generated code. For the Math tasks, we report the Solve Rate (%). All the final result is the average of five experiments. Moreover, we construct a pareto chart to analyze the cost of executing tasks on each test set by tracking token usage. This approach visually demonstrates the performance-cost trade-offs across different methods.

Methods	HumanEval	MBPP	MATH	GSM8K	HotpotQA	SQuAD	Avg.
	pass@1	pass@1	Solve Rate(%)	Solve Rate(%)	F1 Score	F1 Score	
IO	88.2	72.2	48.2	90.0	65.2	84.2	74.7
CoT	89.3	72.2	49.2	89.2	63.7	85.3	74.8
SCCOT	89.7	73.3	48.9	90.8	65.1	86.7	75.8
LogiCoT	89.7	73.7	48.7	90.2	64.3	85.2	75.3
ToT	90.6	74.2	50.2	90.4	66.5	86.2	76.4
GIoT	90.2	72.2	47.5	90.4	63.3	85.1	74.8
AIoT	90.4	72.8	47.2	91.2	64.3	86.3	75.4
Ours	**95.9**	**84.3**	**53.4**	**93.3**	**72.2**	**88.8**	**81.3**

Table 1: Comparative analysis of performance of different methods and INoT on divided test set of Code, Math, QA tasks and average performance of each methods. The methods in this experiment uniformly use DeepSeek-V2.5. In the table, the best-performing results on the same dataset test set are highlighted in bold.

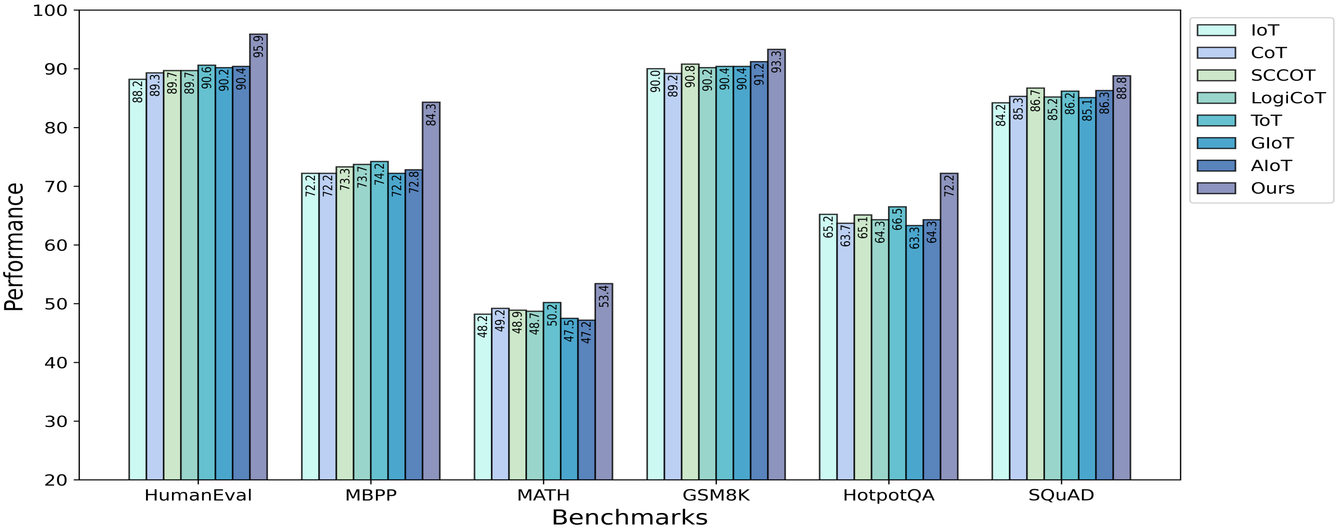


Figure 2: **Performance comparison with baselines.** Experiments are conducted on six datasets representing Math, Code, and QA tasks. INoT outperforms baselines on each dataset.

4.2 Results and Analysis

Comparison to Baselines. Experimental results on six datasets in QA, Code, Math domains, as shown in Table 1, reflect the effectiveness and applicability of INoT compared to the baseline methods. INoT achieves an average performance of 81.3, surpassing all baseline methods by an average margin of 7.95%. This consistent improvement highlights INoT's ability to enhance model reasoning and problem-solving across diverse domains. Specifically, INoT outperforms the best-performing baseline, ToT, by 6.41 percentage points in overall performance. INoT exhibits substantial gains in code generation tasks, achieving a pass@1 score of 95.9 on HumanEval and 84.3 on MBPP, significantly higher than the best baseline scores of 90.6 and 74.2, respectively. Similarly, in mathematical problem-solving, INoT attains a solve rate of 53.4 on MATH and 93.3 on GSM8K, surpassing the best baseline performance of 50.2 and 91.2. For QA benchmarks, INoT achieves an F1 score of 75.2 on HotpotQA and 87.8 on SQuAD, representing notable improvements over the best baseline performances of 66.5 and 86.7, respectively. These results underscore INoT's robustness and adaptability across different problem types, affirming its advantage in text task reasoning and structured problem-solving scenarios.

Comparative Study Among Different LLMs. As shown in Table 2, the performance of INoT across different LLMs demonstrates its model-agnostic nature. We conduct experiments with INoT using different LLMs on benchmarks. On the same datasets, the performance variation of INoT across different models does not exceed 5%. The results indicate that INoT shows stable performance and is not significantly affected by the variations in LLMs.

Cost Analysis. As the Figure 3 shown, we compare the performance and token cost between the baselines and INoT. The range of token cost ranges from a minimum of 6.1 to a maximum of 84.7, with significant differences. The token cost of INoT in all systems remains at a low level, usually around 26, far below the average level of other methods. Token cost of INoT is 58.3% lower on average than the best performing method in baselines. From the comprehensive comparison of token cost and performance score, the "Ours" method shows significant advantages. It not only has the lowest token cost, but also has the highest performance score, showing the best balance between resource efficiency and task effectiveness.

Multimedia Versatility Analysis. Experimental results on three image QA benchmarks, as shown in Table 3, demonstrate that INoT has excellent performance compared to various base lines. Under different settings and benchmarks, the proposed method consistently outperforms the baselines, proving the effectiveness and versatility of INoT in image inference tasks. Compared with the base line methods (IO and CoT), the proposed method has an average accuracy improvement of about 2.7% on SQA-IMG; an average improvement of about 4.5% on LLaVA-Bench (COCO); and an average improvement of about 6.0% on LLaVA-Bench (In-the-Wild). These improvements indicate that the proposed method can effectively enhance the performance of the model in

Models	HumanEval	MBPP	MATH	GSM8K	HotpotQA	SQuAD
	pass@1	pass@1	Solve Rate(%)	Solve Rate(%)	F1 Score	F1 Score
DeepSeek-V2.5	95.9	84.3	53.4	93.3	72.2	88.8
DeepSeek-V2	93.4	82.4	52.8	90.2	71.6	87.4
Claude-3.5-sonnet	96.2	84.5	53.2	93.8	72.1	86.7
Gemma 2	93.2	82.5	52.2	90.3	72.0	84.8
Qwen2.5-coder	92.7	81.7	51.7	91.2	70.3	84.2
Llama3.2	93.3	79.4	51.2	90.4	71.5	84.3

Table 2: Comparative analysis of performance of INoT when utilizing different LLMs as the core LLM.

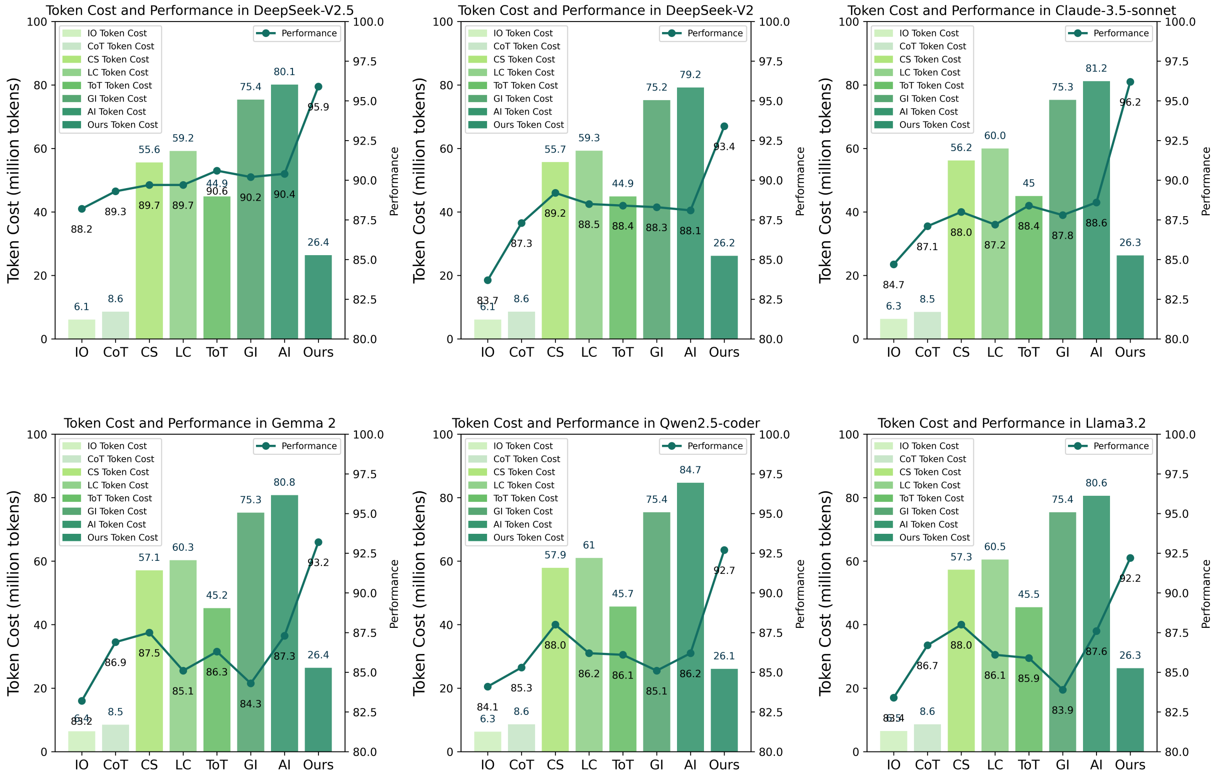


Figure 3: **Analysis of token cost.** Comparing the token cost and performance of baselines and INoT on the HumanEval dataset test set utilizing different LLMs. In this chart, CS represents SCCOT, LC represents LogiCoT, GI represents GIoT and AI represents AIoT. The line chart shows the performance, and the bar chart shows the token cost.

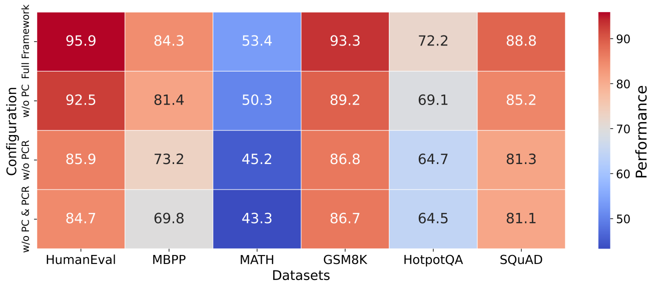


image QA tasks, especially on the more challenging LLaVA-Bench (In-the-Wild) dataset, where the improvement is more significant. The LLaMA-Adaptor and MM-CoT models have also achieved significant improvements in their respective benchmark tests, further demonstrating the generalization and adaptability of the method. In addition, the results also verify the usefulness of the Image Augment Module in INoT prompt. INoT without Image Augment Module shows lower accuracy on all benchmarks of each model than the full INoT. For example, in LLaVA, the accuracy of INoT without Image Augment Module on SQA-IMG is 88.9%, the accuracy on LLaVA-Bench (COCO) is 81.2%, and the accuracy on LLaVA-Bench (In-the-Wild) is 69.8%, which is 1.3%, 2.2%, and 2.6% lower than INoT, respectively.

Methods	Setting	SQA-IMG	LLaVA^{qa}	LLaVA^{w}
LLaVA	IO	88.0	85.1	67.3
	CoT	88.2	80.2	68.2
	ours(w/o IAM)	88.9	81.2	69.8
	ours	**90.2**	**83.4**	**72.4**
LLaMA-Adaptor	IO	80.3	76.7	62.4
	CoT	81.2	77.2	62.8
	ours(w/o IAM)	81.8	78.6	63.9
	ours	**83.9**	**80.2**	**70.1**
MM-CoT	IO	82.9	77.6	63.9
	CoT	83.2	78.1	64.2
	ours(w/o IAM)	83.8	79.2	64.7
	ours	**85.7**	**80.4**	**68.7**

Table 3: Accuracy (%) on three image QA benchmarks. SQA-IMG: ScienceQA-IMG (zero-shot); LLaVA^{qa}: LLaVA-Bench (COCO); LLaVA^{w}: LLaVA-Bench (In-the-Wild). INoT outperforms baselines on each benchmarks. Additionally, Image Augment Module in INoT prompt is verified to be useful.

Ablation Study. In order to verify whether the custom PromptCode in the code-integrated prompt really works, we conduct ablation experiments on benchmarks. As shown in Figure **??**, these are the experimental results after removing the custom code of PromptCode.

On the same dataset, the performance of INoT has a certain degree of decline. Additionally, it can be seen that the impact of the execution of the PromptCode logic is greater than that of the design of PromptCode and PromptComplier, indicating that running PromptCode to specify the reasoning logic plays a significant role in the optimal performance of LLMs.

5 Conclusion and Future Work

We propose a novel AI Agent Reasoning Framework with Introspection of Thought (INoT) by designing a new LLM-Read code in prompt. It constructs a virtual multi-agent debate reasoning framework within LLM. Therefore, self-denial and reflection occur within LLM instead of outside LLM. Benchmark experiments are conducted on Math, Code, and QA tasks. The performance of INoT exceeds baselines, demonstrating its effectiveness in completing natural language tasks. In addition, putting the iterative reflection and optimization of the conventional agent reasoning framework into LLM effectively reduces the cost of completing natural language tasks. Furthermore, we demonstrate the versatility of INoT in image interpretation and inference through verification experiments on three image QA benchmarks. For future work, we would like to continue to optimize the code-integrated prompts, enhance the internal reasoning logic of LLM. Also, we will apply the INoT reasoning framework to more diverse tasks.

References

[1] Anthropic. *Claude-3.2-sonnet*. 2024. URL https://www.anthropic.com/news/claude-3-5-sonnet.

[2] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, and C. Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

[3] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

[4] W. Chen, X. Ma, X. Wang, and W. W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023. URL https://arxiv.org/abs/2211.12588.

[5] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems, 2021. URL https://arxiv.org/abs/2110.14168.

[6] DeepSeek. *DeepSeek-V2.5*. 2024. URL https://huggingface.co/deepseek-ai/DeepSeek-V2.5.

[7] N. Ding, Y. Qin, G. Yang, F. Wei, Z. Yang, Y. Su, S. Hu, Y. Chen, C.-M. Chan, W. Chen, et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235, 2023.

[8] Y. Dong, X. Jiang, Z. Jin, and G. Li. Self-collaboration code generation via chatgpt. *ACM Transactions on Software Engineering and Methodology*, 33(7):1–38, 2024.

[9] T. M. Gemma Team, C. Hardin, R. Dadashi, S. Bhupatiraju, L. Sifre, M. Rivière, M. S. Kale, J. Love, P. Tafti, L. Hussenot, and et al. Gemma. 2024. doi: 10.34740/KAGGLE/M/3301. URL https://www.kaggle.com/m/3301.

[10] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset, 2021. URL https://arxiv.org/abs/2103.03874.

[11] W. Hu, H. Liu, L. Chen, F. Zhou, C. Xiao, Q. Yang, and C. Zhang. Socratic questioning: Learn to self-guide multimodal reasoning in the wild, 2025. URL https://arxiv.org/abs/2501.02964.

[12] X. Huang, W. Liu, X. Chen, X. Wang, H. Wang, D. Lian, Y. Wang, R. Tang, and E. Chen. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*, 2024.

[13] E. Latif and X. Zhai. Fine-tuning chatgpt for automatic scoring. *Computers and Education: Artificial Intelligence*, 6:100210, 2024.

[14] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning, 2023. URL https://arxiv.org/abs/2304.08485.

[15] H. Liu, Z. Teng, L. Cui, C. Zhang, Q. Zhou, and Y. Zhang. Logicot: Logical chain-of-thought instruction-tuning, 2023. URL https://arxiv.org/abs/2305.12147.

[16] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35, 2023.

[17] Z. Liu, W. Yao, J. Zhang, L. Yang, Z. Liu, J. Tan, P. K. Choubey, T. Lan, J. Wu, H. Wang, et al. Agentlite: A lightweight library for building and advancing task-oriented llm agent system. *arXiv preprint arXiv:2402.15538*, 2024.

[18] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang. A dynamic llm-powered agent network for task-oriented agent collaboration, 2024. URL https://arxiv.org/abs/2310.02170.

[19] Llama. *Llama3.2*. 2024. URL https://ollama.com/library/llama3.2.

[20] J. Long. Large language model guided tree-of-thought, 2023. URL https://arxiv.org/abs/2305.08291.

[21] P. Lu, S. Mishra, T. Xia, L. Qiu, K.-W. Chang, S.-C. Zhu, O. Tafjord, P. Clark, and A. Kalyan. Learn to explain: Multimodal reasoning via thought chains for science question answering. *Advances in Neural Information Processing Systems*, 35:2507–2521, 2022.

[22] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende. Prompt engineering in large language models. In *International conference on data intelligence and cognitive informatics*, pages 387–402. Springer, 2023.

[23] Qwen. *Qwen2.5-coder*. 2024. URL https://ollama.com/library/qwen2.5-coder.

[24] S. K. Radha, Y. N. Jelyani, A. Ghukasyan, and O. Goktas. Iteration of thought: Leveraging inner dialogue for autonomous large language model reasoning, 2024. URL https://arxiv.org/abs/2409.12618.

[25] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ questions for machine comprehension of text. In J. Su, K. Duh, and X. Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, Nov. 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

[26] K. I. Roumeliotis and N. D. Tselikas. Chatgpt and open-ai models: A preliminary review. *Future Internet*, 15(6):192, 2023.

[27] J. Saad-Falcon, A. G. Lafuente, S. Natarajan, N. Maru, H. Todorov, E. Guha, E. K. Buchanan, M. Chen, N. Guha, C. Ré, et al. Archon: An architecture search framework for inference-time techniques. *arXiv preprint arXiv:2409.15254*, 2024.

[28] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha. A systematic survey of prompt engineering in large language models: Techniques and applications, 2024. URL https://arxiv.org/abs/2402.07927.

[29] X. Song, Y. Wu, W. Wang, J. Liu, W. Su, and B. Zheng. Progco: Program helps self-correction of large language models, 2025. URL https://arxiv.org/abs/2501.01264.

[30] P. Wang, Z. Wang, Z. Li, Y. Gao, B. Yin, and X. Ren. Scott: Self-consistent chain-of-thought distillation, 2023. URL https://arxiv.org/abs/2305.01879.

[31] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf.

[32] Z. Yang, P. Qi, S. Zhang, Y. Bengio, W. W. Cohen, R. Salakhutdinov, and C. D. Manning. Hotpotqa: A dataset for diverse, explainable multi-hop question answering, 2018. URL https://arxiv.org/abs/1809.09600.

[33] Y. Yao, J. Duan, K. Xu, Y. Cai, Z. Sun, and Y. Zhang. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, page 100211, 2024.

[34] J. Zhang, J. Xiang, Z. Yu, F. Teng, X. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, B. Zheng, B. Liu, Y. Luo, and C. Wu. Aflow: Automating agentic workflow generation, 2024. URL https://arxiv.org/abs/2410.10762.

[35] R. Zhang, J. Han, C. Liu, P. Gao, A. Zhou, X. Hu, S. Yan, P. Lu, H. Li, and Y. Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *arXiv preprint arXiv:2303.16199*, 2023.

[36] Z. Zhang, A. Zhang, M. Li, H. Zhao, G. Karypis, and A. Smola. Multimodal chain-of-thought reasoning in language models, 2024. URL https://arxiv.org/abs/2302.00923.

[37] Q. Zhou, R. Zhou, Z. Hu, P. Lu, S. Gao, and Y. Zhang. Image-of-thought prompting for visual reasoning refinement in multimodal large language models. *arXiv preprint arXiv:2405.13872*, 2024.

[38] W. Zhou, Y. E. Jiang, L. Li, J. Wu, T. Wang, S. Qiu, J. Zhang, J. Chen, R. Wu, S. Wang, et al. Agents: An open-source framework for autonomous language agents. *arXiv preprint arXiv:2309.07870*, 2023.