ENHANCING GRAPH OF THOUGHT: ENHANCING PROMPTS WITH LLM RATIONALES AND DYNAMIC TEMPERATURE CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Enhancing Graph Of Thoughts (EGoT), a method designed to enhance the performance of large language models (LLMs) on complex reasoning tasks. EGoT automates the process of generating accurate responses using given data and a base prompt. The process consists of several steps: It obtains an initial response from the answering node using the base prompt. Evaluation node evaluates the response and generates reasoning for it, utilizing the score's probabilities to enhance evaluation accuracy. The reasoning from both the answering node and the evaluation node is aggregated to identify the problem in the response. This aggregated reasoning is incorporated into the base prompt to obtain an enhanced response. These steps are organized in a graph architecture, where the final leaf nodes are merged to produce a final response. As the graph descends, the temperature is lowered using Cosine Annealing and scoring, to explore diverse responses with earlier nodes and to focus on precise responses with later nodes. The minimum temperature in Cosine Annealing is adjusted based on scoring, ensuring that nodes with low scores continue to explore diverse responses, while those with high scores confirm accurate responses. In sorting 256 elements using GPT-4o mini, EGoT performs 88.31% accuracy, respectively, GoT (Graph Of Thought) performance has 84.37%. In the frozen lake problem using GPT-4o, EGoT averages 0.55 jumps or falls into the hole, while ToT (Tree of Thoughts) averages 0.89.

1 INTRODUCTION

In recent research, the performance of large language models (LLMs) has evolved incredibly rapidly, with applications in a variety of fields, including math problem (Shao et al., 2024), robotics (Park et al., 2023), medicine (Lee et al., 2024b; Kwon et al., 2024), and even programming (Wang et al., 2023a; Duong & Meng, 2024; McAleese et al., 2024). To further improve the performance of LLM, researchers are now actively exploring methods to significantly scale up the architecture of models, or optimize with distillation (Qu et al., 2024) and fine-tuning (Singh et al., 2024). These efforts are broadening the scope of LLM and enabling more innovative applications.

Training LLM directly requires a lot of time and GPU resources. To address such limitations, Prompt Engineering, which involves designing effective prompts rather than training the model directly, stands out. Prompt engineering is a technique that can improve the performance of LLM on specific tasks without requiring additional training. Examples of prompts include Chain of Thought (CoT) (Wei et al., 2022), Chain of Thought with Self-Consistency (CoT-SC) (Wang et al., 2023b), Tree of Thought (ToT) (Long, 2023; Yao et al., 2024), Exchange of Thought (EoT) (Yin et al., 2023), and Graph of Thought (GoT) (Besta et al., 2024). These approaches help LLM generate more accurate and useful results.

However, complex problems often decrease the rationale of LLM. When LLM provides a correct answer, its rationale steps are not always reliable (Hao et al., 2024). In addition, most architectures utilize external tools (Stechly et al., 2023; Gou et al., 2024) to improve performance, and prompts often require specific examples (Lee et al., 2024a). Since obtaining the valid rationale makes LLM's performance highly contributing (Yin et al., 2024), the technique of prompting LLM with a score to evaluate the performance of LLM (Valmeekam et al., 2023; Ren et al., 2023) is an ongoing research

area. There are also researches that utilize dynamic temperature control techniques (Cai et al., 2024; Nasir et al., 2024; Zhang et al., 2024; Zhu et al., 2024) to further enhance LLM's rationale ability.

Our approach, EGoT, is an architecture that can automatically generate the prompt and answer from LLM by only initializing the base prompt. In the process, log probability is utilized to evaluate the answers of LLM to increase the confidence. We also propose to dynamically adjust the temperature based on the progress and score of the answer, applying the cosine annealing (Loshchilov & Hutter, 2016) to set a high temperature at the beginning of the graph and a low temperature at the end. The minimum temperature is set as the inverse of the score, so that nodes with high scores consistently provide correct answers, while nodes with low scores explore a wide range of answers. This approach has the advantage of showing constant and consistent performance without the evaluation metric, and it does not need additional examples to avoid bias in the results. The approach focuses on addressing the challenges that emerge during the integration of multiple components, allowing for the optimization of trade-offs and the achievement of significant performance improvements. EGoT is to maximize LLM effectiveness without relying on fine-tuning.

To summarize, EGoT provides the following advantages:

- Dynamic Temperature Control with Cosine Annealing to propagate more accurate rationale to child node prompts.
- Continuously append rationale to the Base Prompt in graph architecture to generate high-quality final response.
- Increase confidence by utilizing the probability of LLM answers for scoring and avoid bias by not including specific examples.
- Directly repeating the input question in the same form once more to make it easier for LLM to understand the problem, and then utilize the previous repeats in the rationale.

2 EGoT Architecture

2.1 Overview

The EGoT structure of graph is shown in Figure 1. The overall node consists of three stages: (Stage 1, AnsweringNode) obtaining the answer to resolve the problem from the LLM, (Stage 2, EvaluationNode) asking the LLM to evaluate its response and assign a score, and (Stage 3, AggregateRationaleNode) collecting the LLM's rationale from both Steps 1 and 2 and forwarding it to the next node. MethodNode is executed only once at the beginning of the overall structure, and this step can be replaced with an expert's problem-solving approach.

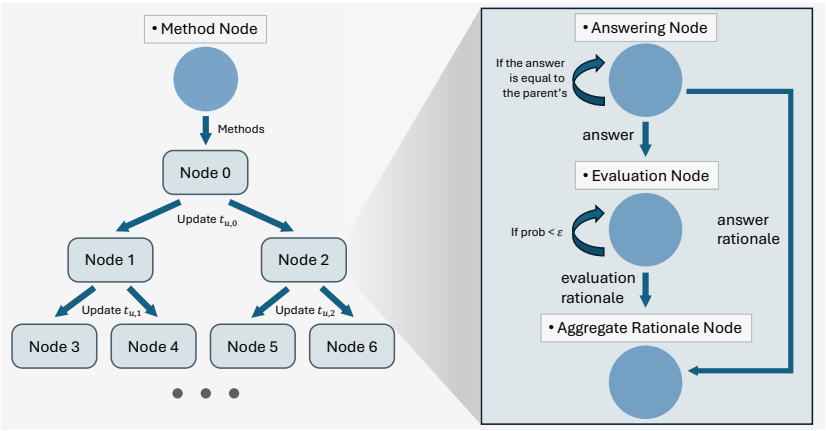

Figure 1: Framework of EGoT. The left side illustrates the overall graph architecture and dynamic temperature. The right side illustrates the internals of each Node N. Each Node contains AnsweringNode, EvaluationNode, and AggregateRationaleNode as sub-nodes. Each node generates a rationale and forwards the temperature to its child answering nodes.

2.2 METHOD NODE

The METHODNODE inquires about the method to solve the question and the methods for evaluating the answer. Although these methodologies can be formulated by humans, in this paper, heuristic methods are requested from LLM and utilized. m_a denotes the method for obtaining the answer to the question, and m_e denotes the method for evaluating the answer. t denotes the temperature of LLM.

$$m_a, m_e = \text{METHODNODE}(Prompt, t = 0) \tag{1}$$

2.3 ANSWERING NODE

ANSWERINGNODE finds the answer to the problem. The top root node solves the problem with the rules. The child node solves the problem using the rationale from the previous nodes. ANSWERINGNODE outputs the answer to the problem and the rationale for the answer. a and r_a are the answer and the rationale regarding the response provided by LLM, and r_{pr} denotes the rationales of the previous nodes. t denotes the temperature, in case of the root node is set to 1, and t_u is the temperature updated on the parent node.

$$a, r_a = \begin{cases} \text{ANSWERINGNODE}(Prompt(m_a, \cdot), t = 1), & \text{if} \quad \text{Node = Root Node} \\ \text{ANSWERINGNODE}(Prompt(m_a, r_{pr}), t = t_u), & \text{else} \quad \text{Node} \neq \text{Root Node} \end{cases} \tag{2}$$

2.4 EVALUATION NODE

EVALUATIONNODE evaluates the answer provided by ANSWERINGNODE. LLM outputs the accuracy of the answer and the rationale for why the accuracy score is given. If the probability of the score provided by LLM is lower than the threshold, EVALUATIONNODE is performed once more. s and r_s are the score and the rationale regarding the response provided by LLM and $\Pr(s)$ is the probability of the score. t denotes the temperature of LLM. We request a score range of 0-100 from LLM to better represent the scores as percentages.

$$s, r_s, \Pr(s) = \text{EVALUATIONNODE}(Prompt(m_e, a), t = 0) \tag{3}$$

2.5 AGGREGATE RATIONALE NODE

AGGREGATERATIONALENODE integrates the rationales provided from ANSWERINGNODE and EVALUATIONNODE. LLM outputs the aggregated rationale and the information considered inaccurate. AGGREGATERATIONALENODE aggregates the information from the two input rationales, emphasizing the incorrect encountered during the reasoning while omitting details related to successful outputs. This concept is similar to the state evaluator in ToT (Yao et al., 2024). The difference with ToT is that it provides a rationale for finding flaws without providing a question and answer. The inaccurate information is the elements that LLM needs to recheck where there is a conflict between the two input rationales. It arises from LLM misinterpretation of the question and can lead to hallucinations and incorrect reasoning in the responses. This information derived from AGGREGATERATIONALENODE is subsequently incorporated into the prompt of the child's ANSWERINGNODE. r_{pr} denotes the aggregate rationale and the incorrect information.

$$r_{pr} = \text{AGGREGATERATIONALENODE}(Prompt(r_a, r_s), t = 0) \tag{4}$$

3 METHODOLOGY

3.1 ENHANCING RESPONSE

This section describes the methods to obtain enhancing responses from LLM. Two methods are used: exploring varied answers to obtain enhancing responses and utilizing the probability of answers to obtain more accurate scoring.

3.1.1 EXPLORING VARIED ANSWER

To explore different answers, multiple root nodes are utilized in the architecture. Since the temperature decreases as the node travels down, multiple graphs are used to explore different answers. There are cases where the node gives the same answer as the node travels down. To solve these cases, if a child ANSWERINGNODE gives the same answer as the parent ANSWERINGNODE, a question is only asked once more. This is because it cannot be determined exactly whether it is the correct answer while the graph is in progress.

3.1.2 ENHANCING SCORE

To enhance the score, the probability that LLM predicts the score token is used to answer the score. If the probability does not exceed the threshold, it asks for the score one more time. The probability threshold is set high for evaluation scores of 0 or 100 to ensure these extreme values are assigned only when the model is highly confident. For other scores, from 1 to 99, the threshold is set low to exclude nonsense answers. It is important to consider the order in which LLM is asked for the score and the rationale for the score. If LLM is asked for the rationale first and then the score, LLM thinks that it has a basis in the previous rationale. Therefore, a score of 0 or 100 is often returned regardless of whether the answer is correct or not, with a probability close to 1. For this reason, the score is asked for before the rationale, and the score is obtained with a variety of scores. The setting for the thresholds is explained later in each experiment.

3.2 TEMPERATURE CONTROL

LLM basically sets temperature to 1.0 for creative answers. Whereas when creativity is not required, it sets temperature closer to 0 for consistent answers. However, setting temperature to 0 from the start can lead to fixed answers and errors.

To gradually decrease temperature as the graph progresses, cosine annealing is used. When the answer is well guessed, the temperature is lowered to generate a fixed answer, and when the answer is ambiguous, the temperature is kept high to explore different answers. The reason for evaluating answers in EVALUATIONNODE is not only to create a rationale, but also to control the temperature. If the score is high, it means that the rationale of that ANSWERINGNODE is correct, and this rationale is forwarded to the child nodes, which are expected to generate good answers. On the other hand, if the score is low, the answer needs to be revised, and the rationale of ANSWERINGNODE also needs to improve, requiring various explorations until it is correct.

In cosine annealing, max temperature (t_{max}) is fixed at 0.7 and min temperature is set to the inverse of accuracy so that the higher the accuracy, the lower the temperature. Total epoch is set to the total number of nodes ($node_t$) and the current epoch is defined as the progress of the nodes ($node_c$).

$$t_u = t_{min} + \frac{1}{2}(t_{max} - t_{min})(1 + cos(\frac{node_c}{node_t})), t_{min} = 1 - \sqrt{1-(c-1)^2}, c = s \cdot \Pr(s)^{\frac{1}{e}} \quad (5)$$

c represents the confidence of ANSWERINGNODE. If the answer is scored high and the probability that LLM predicted the score is also high, the confidence is high. If the answer is scored low or the probability that LLM predicted the score is low, the confidence is low. c and t_{min} are between 0 and 1. The probability is used in t_{min} to differentiate between high and low probability cases when LLM answers the score.

3.3 EXAMPLE USE CASE

This section explains the content of section 2 and 3 with a practical example. Figure 2 shows the results of the Frozen lake experiment, one of the experimental results that demonstrates the advantages of EGoT. The blue background represents the hole and the light blue represents the frozen. The two black points on the top left (0, 0) and bottom right (4, 4) represent the start and end. The green line is the route that LLM predicts the answer, the orange square is what EVALUATIONNODE rationale explains as incorrect because it is a hole, and the brown triangle is the position that AGGREGATERATIONALENODE aggregates because the rationale from ANSWERINGNODE and EVALUATIONNODE conflict with each other.

Before the graph starts, METHODNODE is invoked once. The information responded from the METHODNODE is utilized by all subsequent nodes in Figure 2, from Node 0 to Node Final. The graph experiment starts with 3 nodes. In ANSWERINGNODE, Node 0 passes through the holes (2, 1), (3, 1), and Node 1 and Node 2 pass through the holes (2, 4), (3, 4). At EVALUATIONNODE, Node 0 observes the hole at (2, 1) and Node 1 observes the hole at (2, 4), and ANSWERINGNODE states that the answer is incorrect, lowering Node 0 and Node 1's confidence. Conversely, Node 2 has a high confidence in EVALUATIONNODE, because it doesn't find anything wrong. Since it is the first round, temperature remains close to 0.7, regardless of confidence. The node updates the temperature of its two child nodes. Node 3 and Node 4 update the temperature by Node 0.

Because depth 0 informs the coordinates (2, 1) and (2, 4) are holes, depth 1 nodes recognize this as a hole and do not traverse these coordinates. Still, nodes 4, 5, and 8 are unsure of the correct answer because the propagated rationale confuses the information about frozen tile and hole. Depth 1 also can't make a confident decision and answers incorrectly that (3, 3) is a hole. Since one depth is passed, nodes with higher confidence have a lower temperature to update to their child. In the middle, omitted part of the figure, if a node gives an incorrect answer, the temperature increases again, and it explores for coordinate (3, 2). When the final node responds to the answer by incorporating aggregate rationales from the leaf nodes, LLM explores the correct answer, avoiding [2, 1], [3, 2].

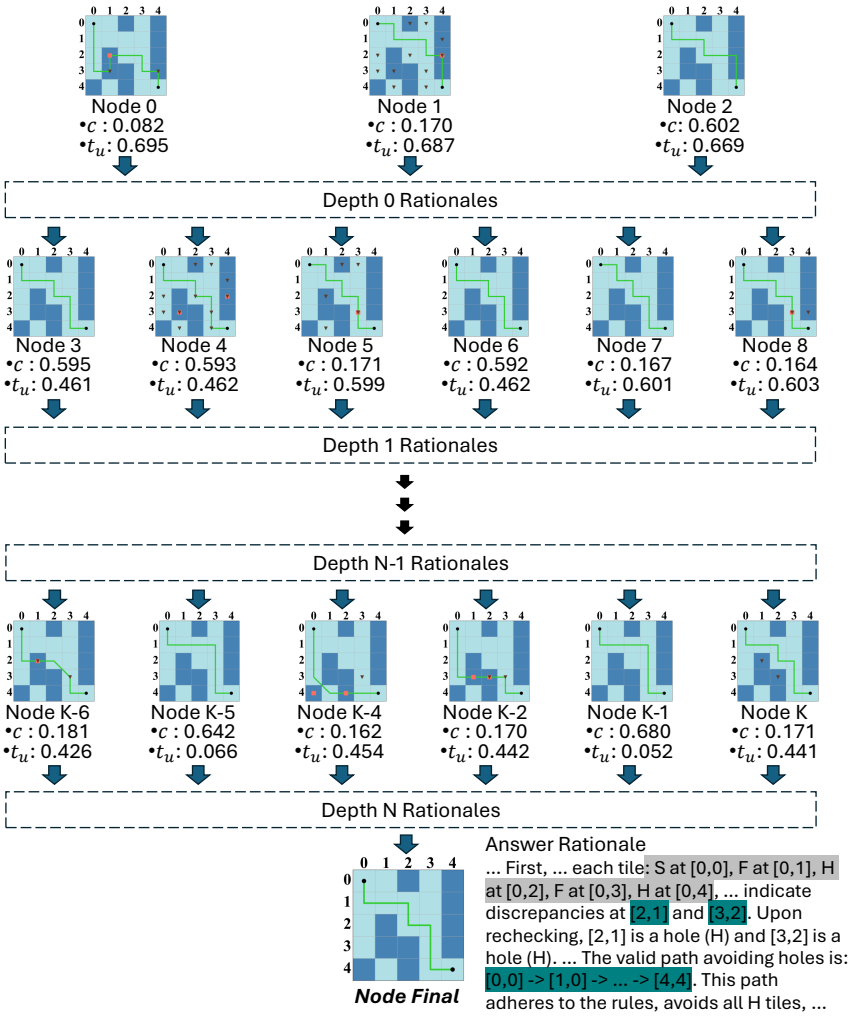

Figure 2: In the Frozen lake example, the temperature decreases as it progresses down the graph, various positions are explored and the graph finds the correct answer using the information.

4 EXPERIMENTS

We use langgraph library to construct the graph, and "gpt-4o-mini-2024-07-18" & "gpt-4o-2024-08-06" for LLM models. The graph structure starts with three root nodes, and when solving a problem, LLM responds with prompts that include all of the rationale information before one depth. At the end of the graph, the answer is aggregated into one, using the response from ANSWERINGNODE with the prompt that incorporates all the aggregate rationales from the leaf nodes.

EGoT evaluates three experiments: document merging, number sorting, and frozen lake. Document merging and number sorting use the graph with a depth of 3, and frozen lake uses the graph with a depth of 4. We experiment with TOT (Long, 2023) that appends the incorrect answer rather than evaluating and exploring each element because in the experiments, the number of nodes increases exponentially to explore each case. GoT only picks the best performing node to evaluate the graph, however, it is changed to select a medium value to compare only structural performance. Solving problems with evaluation metrics is not considered to be a structural advantage, therefore, to fully automate LLM, the evaluation of nodes is assumed to be randomized and the medium value is used as the expectation. Experiments are performed multiple times with the same data. To compare the impact of temperature, the experiment is performed with temperature fixed to 1, named EGoT*.

4.1 DOCUMENT MERGING

We experiment with the dataset provided by GoT for doc merge, and evaluate the result at the end with GoT's scoring prompt. The evaluation compares non-redundancy and retained harmonic mean. The performance is 75.96%, **77.79%**, 76.74%, 76.43%, 76.01%, 74.98%, in the following order: IO, **CoT**, ToT, GoT, EGoT, EGoT*. This experiment motivates the idea that scoring with LLM should not simply be evaluated. The experiment shows that autonomous evaluation by LLM does not have the logical and structural advantages of well-known CoT and ToT. It provides a rationale for the idea that the scoring should be better evaluated.

4.2 NUMBER SORTING

This experiment is a sorting problem with random numbers as input. LLM is able to sort small list lengths successfully, however, LLM performs inaccurately when sorting long lists of numbers. To evaluate the sorting problem, two metrics are utilized: accuracy and number of errors (NOE). Accuracy is the intersection divided by union to measure how similar the two lists are. The number of errors is the number of elements that ascend rather than descend. The higher the accuracy, the better, the lower the number of errors, the better. All nodes except ANSWERINGNODE set the temperature to 0. The threshold for score probability is set to 0.99 for 100 and 0, and 0.5 for others.

The experiment is performed with 100 lists of 128 elements and 100 lists of 256 elements. 128 elements are randomly selected from the numbers 1 to 1000, allowing for duplicates. 256 elements are randomly selected from the numbers 1 to 1500, allowing for duplicates, because GPT 4o's tokenizer splits numbers over 1000 into two tokens. In this experiment, to show the effectiveness of rationale in stating the problem once more, CoT is performed in two ways. CoT1 utilizes the rationale to sort the entire list in three steps: divide the list into four parts, sort each part, and then combine them. CoT2 is the method where the rationale writes the input one more time to understand it, and then sorts the corresponding numbers written in the previous step.

4.3 FROZEN LAKE

A frozen lake is a problem of finding a route to a destination avoiding the hole. To find the correct route in a frozen lake, it is necessary to know the exact location of the holes and understand the rules of the frozen lake. To evaluate the frozen lake problem, two metrics are utilized: Accuracy and number of errors (NOE). Accuracy is the number of successful routes found correctly divided by the total number of attempts. The number of errors is the number of times falling into a hole plus the distance of the jump. All nodes except ANSWERINGNODE set the temperature to 0. The threshold for score probability is set to 0.95 for 100 and 0, and 0.5 for other scores. This experiment is performed on 5 by 5 size lake with 20 examples of 8 holes and 20 examples of 10 holes. GPT-4o and GPT-4o mini are utilized in the experiment.

Table 1: Results of the Number Sorting experiment (GPT-mini 4o)

128 Elements	**IO**	**CoT1**	**CoT2**	**ToT**	**GoT**	**EGoT**	**EGoT***
Accuracy	90.25%	72.13%	90.41%	**92.28%**	90.98%	92.09%	91.70%
Number of Errors	14.07	38.79	13.87	**10.87**	11.88	10.94	11.45
256 Elements							
Accuracy	70.71%	49.50%	83.17%	75.58%	84.37%	**88.31%**	87.94%
Number of Errors	119.51	154.57	49.25	65.74	40.93	**34.54**	35.19

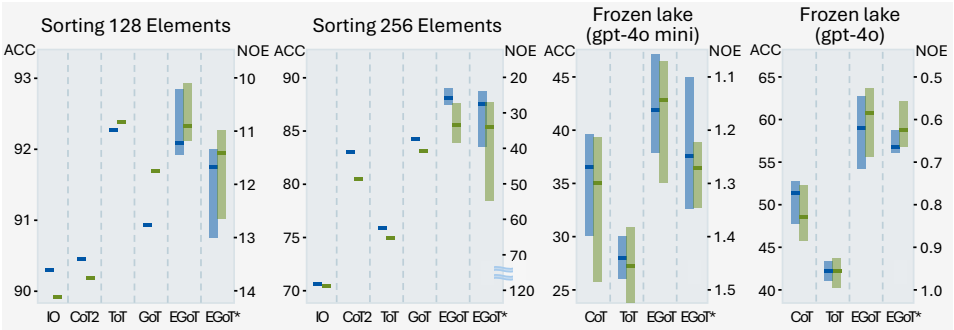

Figure 3: Figure shows Min, Max, and Average for multiple experiments. The blue line on the left of each graph represents accuracy and the green line on the right represents number of errors (NOE). The bars represent the maximum and minimum values, and the darker color in the middle of the bars represents the average. In the sorting problem, IO, CoT2, ToT, and GoT architectures are validated architectures, we experiment only one time. The higher ACC, the better, the lower NOE, the better.

5 Evaluation

5.1 Number Sorting

Table 1 is the experiment result of number sorting. ToT has the best performance for sorting 128 elements, followed by the proposed EGoT. When sorting 256 elements, the proposed EGoT outperformed the other architectures. EGoT* also performs similarly to EGoT, however, it is slightly lower. Five experiments were conducted to verify the consistent performance of EGoT for 128 elements and 256 elements, the results are shown in Figure 3 and it shows consistent performance in general.

The result of CoT1 and CoT2 is the one to focus on here. While there is a relatively slight performance difference when sorting 128 numbers, there is a huge performance difference when sorting 256 numbers. The reason for the difference is that in the first step of CoT1's rationale, when dividing the list into 4 lists, many numbers are missing, and in the last step of the rationale, when merging the 4 lists, it sometimes returns only the numbers from the first list without merging. For the reason, the performance of CoT1 is significantly lower compared to the other experiments. Conversely, CoT2's first step of rationale, which is to repeat elements one more time, is not a difficult task for LLM, and it outputs relatively few missing numbers. Subsequently, when prompting for sorting with the previously mentioned numbers, LLM performs the sorting without difficulty. The tradeoff is an increase in time and the number of output tokens because the rationale process requires more outputs.

We also compare the performance of various LLMs instead of GPT. Since EGoT requires the probability to evaluate the answer, we utilize the Llama 3.1 405B model and the Mixtral 8×22B model, both provided by the Fireworks platform. The Anthropic Claude 3 Haiku model does not provide the probability of the answer, therefore, we fix the probability to 1. The experiments are conducted using 10 samples for sorting 256 data. During the evaluation, both Llama and Mixtral, in contrast to GPT-4o mini, consistently assign a score of 100 in EvaluationNode. In such cases, we request the LLM for the score again. The results of these experiments are presented in Table 2.

Table 2: Results of the 256 Number Sorting experiment using various LLMs

Llama 3.1 405B	**CoT**	**ToT**	**GoT**	**EGoT**
Accuracy	91.59%	92.05%	94.09%	**95.85%**
Number of Errors	22.53	21.3	16.4	**11.5**
Mixtral 8×22B				
Accuracy	82.91%	71.91%	83.85%	**89.05%**
Number of Errors	73.63	83.6	44.6	**30.67**
Claude 3 Haiku				
Accuracy	92.10%	**97.62%**	94.38%	95.00%
Number of Errors	20.4	**6.2**	14.6	12.9

Table 3: Results of the Frozen Lake experiment (GPT-4o mini)

5 by 5 with 8 holes	**CoT**	**ToT**	**EGoT**	**EGoT***
Accuracy	36%	28.1%	**43%**	41%
Number of Errors	1.33	1.38	**1.13**	1.14
5 by 5 with 10 holes				
Accuracy	36.3%	27.6%	**41.0%**	34.0%
Number of Errors	1.28	1.54	**1.15**	1.43

Table 4: Results of the Frozen Lake experiment (GPT-4o)

5 by 5 with 8 holes	**CoT**	**ToT**	**EGoT**	**EGoT***
Accuracy	50.8%	39.7%	**58.8%**	53.3%
Number of Errors	0.83	1.03	0.64	**0.62**
5 by 5 with 10 holes				
Accuracy	51.7%	44.4%	59.0%	**60.3%**
Number of Errors	0.80	0.89	**0.55**	0.60

5.2 Frozen Lake

Table 3 and Table 4 are the experiment results for the frozen lake. In the experiment, EGoT and EGoT* perform better than other architectures. To evaluate the consistent performance, 5 experiments are performed on the GPT-4o mini, and 3 experiments are performed on the GPT-4o. The results can be shown in Figure 3. GoT is applicable only when the problem can be divided into subproblems, whereas Frozen Lake cannot be broken down into smaller parts. Therefore, we cannot compare GoT in this experiment. When the rationale simply requests to understand the position of the holes and tiles, LLM confuses a lot, whereas when LLM writes the coordinate next to the input and then requests to understand the position of the holes and tiles, it performs better.

5.3 EGoT's Advantages

EGoT shows the benefits of utilizing rationale information in prompt engineering instead of focusing only on LLM answer, and proposes a structure to improve prompt engineering performance by utilizing the effect of LLM's temperature. Therefore, EGoT has two main advantages.

First, EGoT generalizes the problem by generating the prompts to enhance the basis prompt. The basis prompt contains only the rule and rationale step of the problem, and the child node enhances the prompt by appending only the parent's rationale output. In all experiments, EGoT performs high performance, showing that the enhancing prompt is effective.

Second, dynamic temperature control and requesting the score from LLM to increase the confidence of the answer through the score and the probability of the corresponding token. Cosine Annealing is used to control the temperature, therefore, it is possible to explore various answers and find various rationales at the beginning. Obtaining a variety of rationales helps to understand the problems of the problem and fix the prompt engineering easily. In the end, the low temperature allows us to focus on more accurate answers rather than diversity.

5.4 Differences Between EGoT and EGoT*

EGoT* does not include dynamic temperature control, which leads to a continual exploration of diverse solutions. This exploration helps maintain greater diversity in the responses. In tasks where the LLM performs well, EGoT can identify the correct answer, rather than focusing on maintaining response diversity. However, in tasks where LLM performance is lower, exploration may lead to better answers. EGoT can sometimes exhibit lower performance when it utilizes only a portion of the provided rationale instead of considering all of it. For this reason, EGoT* performs similarly to EGoT on average when solving frozen lake problems. On the other hand, when solving sorting problems, EGoT showed better performance.

5.5 Differences with Other Architectures

Since EGoT depends on LLM to evaluate, it doesn't need a tool to check that the answer is correct. Math problems are simple to evaluate for correctness with the assistance of tools, however, general questions are not simple to evaluate for correctness with the assistance of tools. EGoTs do not require splitting the problem. GoT is a useful architecture if the problem can be divided hierarchically, however it is difficult to apply to general problems where the problem cannot be partitioned. ToT is a similar implementation of BFS or A* in LLM, and it is difficult to solve with BFS or A* when there are many elements to evaluate, such as number sorting. CoT-SC focuses on the answer, not the rationale, when voting for the final answer, which is efficient if the answer is a scalar. However, when the answer is a list or vector, such as in experiments like sorting or frozen lake, it is not as applicable as ToT. EGoT emphasize the importance of rationale and propose that it can have the same effect as voting by aggregating rationales continuously, discarding incorrect rationales and allowing correct rationales. The disadvantage of EGoT compared to other architectures is that it requires more time and credits due to having many nodes. Since EGoT utilizes three nodes (Answering, Evaluation, and Aggregate Rationale) to obtain one answer, it takes three times more time and credits to obtain the same number of answers.

6 Related Work

6.1 Chaining Architecture and Rationale Step

There are several Prompt engineering architectures, including CoT (Wei et al., 2022), CoT-SC (Wang et al., 2023b), ToT (Long, 2023; Yao et al., 2024), EoT (Yin et al., 2023), and GoT (Besta et al., 2024). Various methods of evolving CoT and voting on the results of CoT are proposed. There are papers that emphasize the correct answer and others that emphasize the rationale. EGoT utilizes to construct the architecture as known EOT and Determlr (Sun et al., 2024).

CoT emphasized the importance of rationale and CoT-SC, on the contrary, focused on the correct answer rather than rationale. The importance of providing rationale steps in prompts is widely recognized, and this leads to research on which rationale steps to include (Xu et al., 2024). Generally, it summarizes the input (Zhang et al., 2023), separates the steps and gives the feedback in the input (Yuan et al., 2024; Madaan et al., 2024), or provides an explanation of the input (Yugeswardeenoo et al., 2024). Villarreal-Haro et al. (2024) and Yin et al. (2024) show that utilizing a rationale with negative information and evaluating the rationale with probability increases the performance of the rationale, and shows the validity of the EGoT rationale step.

6.2 TEMPERATURE CONTROL AND EVALUATION LLM RESPONSE

Temperature increases LLM's response diversity, and temperature affects the performance of answers. Zhu et al. (2024) shows the performance increase by adapting temperature with token confidence. To evaluate LLM response, voting (Li et al., 2022; Du et al., 2024), debating (Liang et al., 2023; Xiong et al., 2023) and scoring (Lee et al., 2024a) are utilized. Since evaluating LLM response affects the performance of the architecture significantly, external tools (Gou et al., 2024) are used to evaluate the confidence level of LLM response (Zhu et al., 2023). Motivated by these methods, we utilize debating to obtain the answer by providing the rationale of the parent node to LLM to infer the correct answer, and we perform self-evaluation on a single token by requesting the score from LLM first instead of utilizing all the response in the EVALUATIONNODE. We defined confidence by utilizing the score and the probability of the token responded by LLM to self evaluate.

7 CONCLUSION

Prompt engineering is an area of study that is key to effectively utilizing LLM, maximizing the advantage of LLM: the applicability of the model to a wide variety of problems without training. Fine-tuning is essential when an LLM needs to acquire specialized skills for certain tasks, however, it can reduce generalization capabilities and tends to be costly and resource-intensive. Chain of Thought (CoT) approach enhanced the ability to reason in general situations, recently various architectures evolved methodologies that are more effective for special cases.

We emphasize that LLM performance is already enough to enable automated solutions for intuitive problems, and use the simple rationale approach of repeating the question entered when LLM prints an answer because it is effective. Different people use different reasoning forms for the problem and requirement, however, EGoT approach is generally applicable to a wide range of situations and leads to improved performance. Our work reemphasizes the importance of rationale, and its concise architecture suggests the possibility of prompt engineering for a wide variety of problems.

Improving the performance of the LLM is also important, obviously. We tried to compare chess puzzles to verify the performance of EGoT architecture. However, despite adding a rule in the prompts that no piece except the knight can jump, GPT-4o mini thinks it can jump over a piece in the middle of a move. As a result, no architectures can find a move that captures the opponent's piece and checkmates, and the performance is not enough to compare results. Therefore, we hope that prompt engineering techniques improve with LLM performance improvement.

REFERENCES

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

Chengkun Cai, Xu Zhao, Yucheng Du, Haoliang Liu, and Lei Li. T2 of thoughts: Temperature tree elicits reasoning in large language models. *arXiv preprint arXiv:2405.14075*, 2024.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=zj7YuTE4t8`.

Ta Nguyen Binh Duong and Chai Yi Meng. Automatic grading of short answers using large language models in software engineering courses. In *2024 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1–10. IEEE, 2024.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=Sx038qxjek`.

Shibo Hao, Yi Gu, Haotian Luo, Tianyang Liu, Xiyan Shao, Xinyuan Wang, Shuhua Xie, Haodi Ma, Adithya Samavedhi, Qiyue Gao, Zhen Wang, and Zhiting Hu. LLM reasoners: New evaluation, library, and analysis of step-by-step reasoning with large language models. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*, 2024. URL https://openreview.net/forum?id=h1mvwbQiXR.

Taeyoon Kwon, Kai Tzu-iunn Ong, Dongjin Kang, Seungjun Moon, Jeong Ryong Lee, Dosik Hwang, Beomseok Sohn, Yongsik Sim, Dongha Lee, and Jinyoung Yeo. Large language models are clinical reasoners: Reasoning-aware diagnosis framework with prompt-generated rationales. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 18417–18425, 2024.

Gyeong-Geon Lee, Ehsan Latif, Xuansheng Wu, Ninghao Liu, and Xiaoming Zhai. Applying large language models and chain-of-thought for automatic scoring. *Computers and Education: Artificial Intelligence*, 6:100213, 2024a.

Suhyeon Lee, Won Jun Kim, Jinho Chang, and Jong Chul Ye. LLM-CXR: Instruction-finetuned LLM for CXR image understanding and generation. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=BqHaLnans2.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*, 2022.

Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Zhaopeng Tu, and Shuming Shi. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.

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

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

Nat McAleese, Rai Michael Pokorny, Juan Felipe Ceron Uribe, Evgenia Nitishinskaya, Maja Trebacz, and Jan Leike. Llm critics help catch llm bugs. *arXiv preprint arXiv:2407.00215*, 2024.

Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. Llmatic: neural architecture search via large language models and quality diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1110–1118, 2024.

Jeongeun Park, Seungwon Lim, Joonhyung Lee, Sangbeom Park, Minsuk Chang, Youngjae Yu, and Sungjoon Choi. Clara: classifying and disambiguating user commands for reliable interactive robotic agents. *IEEE Robotics and Automation Letters*, 2023.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching foundation model agents how to self-improve. In *Automated Reinforcement Learning: Exploring Meta-Learning, AutoML, and LLMs*, 2024.

Jie Ren, Yao Zhao, Tu Vu, Peter J Liu, and Balaji Lakshminarayanan. Self-evaluation improves selective generation in large language models. In *Proceedings on*, pp. 49–64. PMLR, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, YK Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

Kaya Stechly, Matthew Marquez, and Subbarao Kambhampati. GPT-4 doesn't know it's wrong: An analysis of iterative prompting for reasoning problems. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=PMtZjDYB68.

Hongda Sun, Weikai Xu, Wei Liu, Jian Luan, Bin Wang, Shuo Shang, Ji-Rong Wen, and Rui Yan. Determlr: Augmenting llm-based logical reasoning from indeterminacy to determinacy. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9828–9862, 2024.

Karthik Valmeekam, Matthew Marquez, and Subbarao Kambhampati. Investigating the effectiveness of self-critiquing in LLMs solving planning tasks. In *NeurIPS 2023 Foundation Models for Decision Making Workshop*, 2023. URL https://openreview.net/forum?id=gGQfkyb0KL.

Kapioma Villarreal-Haro, Fernando Sánchez-Vega, Alejandro Rosales-Pérez, and Adrián Pastor López-Monroy. Stacked reflective reasoning in large neural language models. *Working Notes of CLEF*, 2024.

Ruocheng Wang, Eric Zelikman, Gabriel Poesia, Yewen Pu, Nick Haber, and Noah D Goodman. Hypothesis search: Inductive reasoning with language models. *arXiv preprint arXiv:2309.05660*, 2023a.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=1PL1NIMMrw.

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

Kai Xiong, Xiao Ding, Yixin Cao, Ting Liu, and Bing Qin. Examining inter-consistency of large language models collaboration: An in-depth analysis via debate. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 7572–7590, 2023.

Tianyang Xu, Shujin Wu, Shizhe Diao, Xiaoze Liu, Xingyao Wang, Yangyi Chen, and Jing Gao. Sayself: Teaching llms to express confidence with self-reflective rationales. *arXiv preprint arXiv:2405.20974*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Zhangyue Yin, Qiushi Sun, Cheng Chang, Qipeng Guo, Junqi Dai, Xuan-Jing Huang, and Xipeng Qiu. Exchange-of-thought: Enhancing large language model capabilities through cross-model communication. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 15135–15153, 2023.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Zhiyuan Zeng, Xiaonan Li, Junqi Dai, Qinyuan Cheng, Xuan-Jing Huang, and Xipeng Qiu. Reasoning in flux: Enhancing large language models reasoning through uncertainty-aware adaptive guidance. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2401–2416, 2024.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. Advancing LLM reasoning generalists with preference trees. In *AI for Math Workshop @ ICML 2024*, 2024. URL https://openreview.net/forum?id=2Y1iiCqM5y.

Dharunish Yugeswardeenoo, Kevin Zhu, and Sean O'Brien. Question-analysis prompting improves llm performance in reasoning tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 543–554, 2024.

Shimao Zhang, Yu Bao, and Shujian Huang. Edt: Improving large language models' generation by entropy-based dynamic temperature sampling. *CoRR*, abs/2403.14541, 2024. URL https://doi.org/10.48550/arXiv.2403.14541.

Zhuosheng Zhang, Yao Yao, Aston Zhang, Xiangru Tang, Xinbei Ma, Zhiwei He, Yiming Wang, Mark Gerstein, Rui Wang, Gongshen Liu, et al. Igniting language intelligence: The hitchhiker's guide from chain-of-thought reasoning to language agents. *arXiv preprint arXiv:2311.11797*, 2023.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Yongfeng Huang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. Solving math word problems via cooperative reasoning induced language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4471–4485, 2023.

Yuqi Zhu, Jia Li, Ge Li, YunFei Zhao, Zhi Jin, and Hong Mei. Hot or cold? adaptive temperature sampling for code generation with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 437–445, 2024.

A APPENDIX

A.1 EVALUATING COSTS: A COMPARISON TO GOT

Table 5: Cost Comparison for solving one problem (GPT-4o mini) in USD

128 Elements	**Average**	**Minimum**	**Maximum**
EGoT	0.03844	0.02922	0.04404
GoT	**0.03246**	**0.02815**	**0.03658**
256 Elements			
EGoT	**0.04302**	0.03484	0.06855
GoT	0.05289	**0.03353**	**0.05852**

The number of nodes in the experiment is adjusted through iterative testing to ensure that the cost difference from GoT remains minimal. As a result, while the number of subnodes and tokens per node in EGoT is higher compared to GoT, the total number of nodes in the graph is reduced, ensuring that the overall cost difference is not substantial. For the sorting 128 elements, GoT generally exhibits approximately 20% lower cost; however, it does not hold for the sorting 256 elements. This discrepancy can be attributed to EGoT placing greater emphasis on rationale within the prompt and requesting more detailed responses. Consequently, even as the problem and correct answer length increases, the total input/output tokens increases only marginally. In contrast, GoT focuses primarily on the answer, leading to a proportional increase in input/output tokens as the problem length grows.

Mathematically, the total output tokens can be represented as $A + R$, where A denotes the number of answer tokens and R represents the number of rationale tokens. EGoT generally has $R > A$, so even with an increase in A, the rate of increase in the total token count $A + R$ is relatively smaller. On the other hand, GoT typically has $R < A$, meaning that as A increases, the rate of increase in the total token count is greater. For sorting 256 elements, EGoT has a lower average cost, while GoT has lower costs in both minimum and maximum. When LLM solves difficult problems, it tends to generate uncertain or additional responses, which increases the total number of tokens. Conversely, for simpler problems, more concise answer is produced, resulting in fewer tokens.

The reason for the cost difference in maximum is that EGoT has conditional statements. EGoT queries again for uncertain answers to increase the confidence of the answer at the ANSWERINGNODE and EVALUATIONNODE. If this process occurs at every node, it would require a higher cost. There is a higher cost increase for problems that are difficult for LLM to solve. However, in general, EGoT performs better than GoT even though its cost is lower.

A.2 METHOD NODE PROMPT EXAMPLE (SORTING PROBLEM)

Human Message
<Goal> - When a list of numbers is entered, answer the question about how to sort them. - Answer the questions about how to evaluate whether the sorted numbers found by this method is correct. </Goal> <Instruction> - Answers the method based on keywords. - The evaluation methods should be done by LLM, so only print heuristic answers. - Answer without being abstract, only considering the method with the best performance. </Instruction>
LLM Response
Method: - QuickSort Evaluation Method: - Check if the sorted list is in ascending order by comparing each element with the next one. - Count the number of inversions in the original list and the sorted list; the number of inversions should decrease. - Verify that the sorted list contains the same elements as the original list by checking for duplicates and ensuring all elements are present.

Table 6: METHODNODE Prompt Example.

Table 6 is an example of a real prompt we used. The method and evaluation method may not be effective, however, we used it to minimize human intervention.

A.3 ANSWERING NODE PROMPT EXAMPLE (SORTING PROBLEM)

Human Message
... [Provide the method and data to resolve the problem.] ... <Rationale_Step> - Rationalize through 4 processes and must provide a step-by-step explanation in detail. - First, accurately determine and write the all numbers in Input_Data. - Be sure to sorting the numbers and write down the rationale in detail using the information provided in the previous step. - Make sure you explain why the sorting is correct with rationale the entire process. - Finally, answer the process of deducing the correct answer by synthesizing the previous process. </Rationale_Step> ... [Provide the information from the upper nodes.] ... {% if aggregate_rationale %} <Hint> - Refer to the hint, but they are not always accurate, so verify the hint with checking the input data and use them. {% for idx in range(aggregate_rationale|length) %} - Hint_{{ idx }}: {aggregate_rationale[{{ idx }}]}{% endfor %} </Hint> {% endif %}
LLM Response
Answer: - 6, 20, 20, 25, ... Answer Rationale: - First, I will identify all the numbers in the input list, which are: [1348, 147, 1454, 1111, ... Next, using the QuickSort algorithm, I will sort this list in ascending order. Using QuickSort, I will choose a pivot and divide the list into smaller sub-lists of numbers less than or greater than the pivot, then further sort those sub-lists until the entire list is sorted. The resulting sorted list is: [6, 20, 20, 25 ...

Table 7: ANSWERINGNODE Prompt Example. (Jinja2 template)

Table 7 is the prompt that requests the LLM to answer the question. Because we emphasize the importance of rationale, we set four rationale steps and gather effective rationale. The top node does not have aggregate rationale information, while the lower nodes have aggregate rationale information received from the upper nodes. Therefore, the prompt utilizes jinja2 template and forwards this information to the lower nodes at the bottom of the prompt with the Hint tag.

A.4 EVALUATION NODE PROMPT EXAMPLE (SORTING PROBLEM)

Human Message
... [Provide the question, the answer from ANSWERINGNODE, and request a score.] ... <Evaluation_Rationale_Step> - Rationalize through 4 processes and must provide a step-by-step explanation in detail. - First, accurately determine and write the all the numbers in Input_Data. - Be sure to score each number included in the sorted data and write down the rationale in detail using the information provided in the previous step. - Make sure you explain why the sorting is correct with rationale the entire process. - Finally, answer the process of deducing the correct answer by synthesizing the previous process. </Evaluation_Rationale_Step> <Evaluate_Methods> {% for idx in range(evaluate_methods|length) %} - {evaluate_methods[{{ idx }}]}{% endfor %} </Evaluate_Methods>
LLM Response
Accuracy: - 80 Evaluation Rationale: ...For instance, the number '680' appears in the sorted data but is not present in the input data. Additionally, the number '994' is also present in the sorted data but is missing from the input. ...Therefore, while the majority of the numbers are sorted correctly, the presence of these discrepancies warrants a score below 90.

Table 8: EVALUATIONNODE Prompt Example. (Jinja2 template)

Table 8 is the prompt that requests the LLM to evaluate the response. The Evaluation Rationale Step is almost identical to table 7. However, only the content of step 2 changes slightly because it needs to evaluate the score. By parsing the log probability of the number "80" in the LLM response, the confidence is calculated as 0.607.

A.5 AGGREGATE RATIONALE NODE PROMPT EXAMPLE (SORTING PROBLEM)

Human Message
<Input_Rationales> - Evaluation_Rationale: {evaluation_rationale} - Answer_Rationale: {answer_rationale} </Input_Rationales> <Instruction> <Aggregate_Rationale> - Don't print the successes or valid, just print the incorrectly missing numbers. - Integrate the flaws that are problematic or invalid. - Be sure to consider the previous two steps, aggregate the Input_Rationales. </Aggregate_Rationale> <Inaccurate_Info> - Inaccurate_Info is information to recheck regarding the missing numbers, as it seems to be inaccurate. - If the number information in two rationales is different, the number should be included in the inaccurate information. </Inaccurate_Info> </Instruction>
LLM Response
Aggregate Rationale: - The sorted list contains discrepancies, including numbers that are not present in the original input data, such as '680' and '994'. . . . Inaccurate Info: - The numbers '680' and '994' are present in the sorted data but are missing from the input list. Additionally, the number '1496' appears twice in the input list but is only represented once in the sorted list, indicating a potential error in the sorting process.

Table 9: AGGREGATERATIONALENODE Prompt Example. (Jinja2 template)

Table 9 is the prompt that summarizes the rationale generated by the two nodes ANSWERINGNODE and EVALUATIONNODE. AGGREGATERATIONALENODE provides the rationale information by summarizing the key points. Additionally, it extracts negative information and propagates this to the lower nodes. In table 8, LLM informs that 680 and 994 are present in the input, however, the sorted result doesn't contain these numbers, therefore table 9 aggregates this information. Misinformation like 1496 also propagates, though the misinformation gradually vanishes as the graph progresses.