Indian Institute of Technology Guwahati

Prompt engineering Techniques in Large Language Models for Solving Quantitative Problems

Ankit Varshney

210121069

Submitted to:

Dr. Girish Setlur

Department of Physics

Contents

Introduction

In the realm of artificial intelligence, the capabilities of Large Language Models (LLMs) have reached unprecedented heights, revolutionizing various domains, including natural language understanding and generation. However, while LLMs excel in processing and generating textual data, harnessing their potential for quantitative problem-solving demands a nuanced approach. This report delves into the innovative realm of prompt engineering techniques tailored specifically to enhance LLMs' aptitude in reasoning and tackling quantitative challenges.

Prompt engineering stands as a pivotal bridge between the inherent capabilities of LLMs and the diverse array of tasks they confront. By refining the prompts—inputs provided to LLMs—through strategic structuring and formulation, researchers and practitioners endeavor to bolster the models' proficiency in quantitative problem-solving. This entails leveraging a spectrum of techniques, ranging from crafting intricate chains of thought prompts to integrating Python coding within the prompts' framework.

Through an interdisciplinary lens, this report synthesizes insights from natural language processing, cognitive science, and computer programming to illuminate the multifaceted landscape of prompt engineering for quantitative problem-solving with LLMs. By elucidating the underlying principles, methodologies, and applications of these techniques, it seeks to equip researchers, developers, and enthusiasts with the knowledge and tools necessary to unlock the full potential of LLMs in the realm of quantitative analysis.

As we embark on this journey through the frontiers of prompt engineering, we embark on a quest to transcend the boundaries of traditional problem-solving paradigms, harnessing the transformative power of language models to tackle quantitative challenges with ingenuity and sophistication.

Basic Prompt design:

(on a sample problem) **Problem Prompt:**

You are given a physics problem involving a physical quantity Q, expressed in terms of other physical quantities such as mass m, length l, time t, etc. Follow these steps to solve it efficiently:

1. **Use symbolic representation than numerical representation** Better to use this approach to avoid tokenisation rounding error which misleads the final output
2. **Dimensionality Check:** Before diving into calculations, ensure dimensional consistency throughout the problem. Verify that the dimensions of the given quantities match the dimensions of the desired result. If they don't, revisit your equations and ensure that all terms have consistent dimensions.
3. **Limiting Case Evaluation:** Identify limiting cases where certain parameters approach extreme values (e.g., very large or very small). Evaluate the expression for these limiting cases to gain insights into its behavior and to verify the correctness of your solution. This step also helps in understanding the physical implications of the problem.
4. **Back Substitution:** After obtaining the final expression for the desired quantity, back-substitute known values to validate your solution. Use the given numerical values of relevant physical quantities to calculate the numerical value of the result. Compare this numerical value with your initial assumptions and ensure that they align.

By following these steps, we can efficiently approach and solve quantitative problems in physics, ensuring both accuracy and understanding of the underlying concepts.

Zero-shot prompting

Zero-shot prompting is a technique used in natural language processing where a model generates responses without being provided with any specific examples or prompts during training. Instead of being fine-tuned on a specific task or dataset, models are trained to generalize across various prompts and generate appropriate responses based on the input they receive. This approach allows models to generate relevant responses even for tasks or prompts they haven't seen before, making them more versatile and adaptable to new scenarios.

Results:-
The accuracy of this approach on GSM8k dataset using GPT4 is 87.1 % [1] while using this approach on Gemini

Ultra gave accuracy of 94.4 % . [2] GPT4 used for solving JEE Advanced Physics questions (JEEBench Dataset) gave 35.2 % accuracy (See apndix for information about Datasets)

Gemini Ultra is the best model for zero shot prompting on GSM8K dataset. **Limitations of Zero-shot Prompting in Reasoning and Quantitative Problems:**

- **Limited Logical Reasoning**: Zero-shot prompting may struggle with tasks requiring complex logical reasoning, such as inference or deduction, leading to inaccurate or incomplete responses.
- **Numerical Precision**: When dealing with quantitative problems, zero-shot models may not consistently provide precise numerical outputs, resulting in errors or inconsistencies.
- **Mathematical Complexity**: Complex mathematical expressions or operations may pose challenges for zero-shot models, impacting their ability to generate accurate solutions.
- **Domain-specific Concepts**: Tasks involving domain-specific knowledge, such as advanced mathematics or scientific principles, may exceed the general knowledge of zero-shot models, affecting the quality of their responses.
- **Interpretability**: The reasoning process of zero-shot models may not always be interpretable, making it difficult to understand how the model arrived at a particular conclusion, especially in quantitative contexts.
- **Evaluation in Reasoning**: Evaluating the logical consistency and coherence of responses generated by zero-shot models can be challenging, as it requires assessing not just the correctness of the answer but also the validity of the reasoning process.

Few Shot prompting

Few-shot prompting is a technique employed in natural language processing (NLP) to guide large language models (LLMs) towards generating specific outputs. It builds upon the concept of prompting, where a textual cue instructs the LLM on how to respond.

In contrast to zero-shot prompting, which provides only an instruction without examples, few-shot prompting offers a limited set of examples (typically 2-5) alongside the instruction. These examples function as demonstrations, illustrating the desired format and style of the output. By analyzing these examples, the LLM gains a better understanding of the task and can generate more accurate and relevant responses.

Here's a breakdown of the key aspects of few-shot prompting:

Limited examples: A small number of examples (2 or more) are presented to the LLM.
Demonstrative function: The examples serve as demonstrations, showcasing the expected output format and content.
Improved performance: Compared to zero-shot prompting, few-shot prompting leads to more precise and appropriate responses from the LLM.
Task-specific guidance: The examples guide the LLM towards completing a particular task effectively. Overall, few-shot prompting offers a powerful approach to enhance the performance of LLMs in various NLP applications by leveraging a combination of instruction and demonstration.
Prompt Example:-
We will be solving problems related to the motion of objects thrown at an angle. Remember, in projectile motion, the horizontal and vertical components of motion are independent.

Example :

A ball is thrown at an angle of 30 degrees with a speed of 20 m/s. Find the maximum height reached by the ball.

Solution:

We can solve this problem by separating the horizontal and vertical components of motion. The horizontal component (Vx) will remain constant throughout the motion (ignoring air resistance). The vertical component (Vy) will experience acceleration due to gravity ($-9.8m/s^2$).

$Vx = 20m/s * cos(30) = 17.32m/s$ (Horizontal component) $Vy = 20m/s * sin(30) = 10m/s$ (Initial vertical component)

The maximum height is reached when the vertical component of velocity becomes 0. We can use the formula $Vy^2 = u^2 + 2as$, where Vy is the final velocity (0 in this case), u is the initial velocity (10 m/s), a is the acceleration due to gravity ($-9.8m/s^2$), and s is the maximum height (unknown).

$0^2 = (10m/s)^2 + 2 * (-9.8m/s^2) * ss = (10m/s)^2/(2 * 9.8m/s^2)s = 5.1m(Maximumheight)$ A projectile is fired from horizontal ground with speed v and projection angle θ . When the acceleration due to gravity is g , the range of the projectile is d . If at the highest point in its trajectory, the projectile enters a different region where the effective acceleration due to gravity is $g/0.81$

, then the new range is . The value of is

Results:- Using this technique with GPT4 on GSM8k dataset gives 93 % accuracy(Though chain of thought with self consistency is also used)
Limitations of Few-shot Prompting:

- **Limited Generalization**: Few-shot prompting may struggle to generalize effectively to unseen prompts or tasks, especially when the training data is limited.
- **Sensitivity to Training Examples**: The performance of few-shot models heavily relies on the quality and relevance of the provided training examples, making them sensitive to the choice of data.
- **Difficulty in Handling Rare Scenarios**: Few-shot models may have difficulty handling rare or outlier scenarios due to the limited exposure to such cases during training.
- **Dependency on Task Similarity**: The effectiveness of few-shot prompting can vary depending on the similarity between the few-shot task and the tasks seen during training.
- **Challenge with Complex Tasks**: Few-shot models may struggle with complex tasks that require intricate reasoning or extensive domain knowledge beyond what can be captured in a few examples.
- **Difficulty in Adaptation**: Adapting few-shot models to new tasks or domains may require additional fine-tuning or training data, limiting their out-of-the-box applicability.

Chain-of-thought prompting

Chain-of-thought (CoT) prompting is a technique for large language models (LLMs) that goes beyond just getting an answer. It aims to encourage LLMs to reason by breaking down complex problems into smaller, easier-to-solve steps. Here's a breakdown of the concept:

Core Idea:

Traditional prompts ask LLMs for a direct answer. CoT prompting asks LLMs to show their work, explaining the steps they take to arrive at the answer. **How it Works:**

Provide Examples: You give the LLM a prompt with a question and the answer, but also showcase intermediate steps leading to the answer.
Learn by Example: By seeing how problems are solved step-by-step, the LLM learns to mimic this reasoning process for future tasks.
Improved Reasoning: This approach helps LLMs with complex tasks that require logical thinking, like math problems or common-sense reasoning.
Benefits:
More accurate and reliable responses compared to single-step prompting. Enhanced ability to solve problems that require multiple steps. Better understanding of the LLM's reasoning process. Think of it this way:

Imagine solving a math problem. You wouldn't just blurt out the answer. You'd show your work, explaining each step that leads to the solution. CoT prompting encourages LLMs to do the same kind of step-by-step reasoning.

Results:
This approach using GPT4 gave 33.5 % accuracy on JEE Advanced Physics questions on GSM8K it gave 93 %[3] using GPT-4 as well.

Prompt example:- Two inductors L_1 (inductance 1mH, internal resistance 3Ω) and L_2 (inductance 2mH, internal resistance 4Ω), and a resistor R (resistance 12Ω) are all connected in parallel across a 5 V battery. The circuit is

switched on at time $t = 0$. What is the ratio of the maximum to the minimum current ($I_{\max}/I_{\min}$) drawn from the battery? Solve this question step by step solving the problem by dividing into subproblem

Limitations of Chain of Thought:

- **Limited Contextual Understanding**: Chain of Thought may struggle to maintain coherent and contextually relevant responses over longer sequences, leading to drift or loss of coherence.
- **Difficulty in Long-term Dependency**: Maintaining relevance and coherence in responses over multiple turns of interaction can be challenging for Chain of Thought, especially with complex or abstract topics.
- **Sensitivity to Input Quality**: The quality of input prompts or user interactions heavily influences the quality of responses generated by Chain of Thought, making it sensitive to errors or ambiguities in input.
- **Lack of Grounded Knowledge**: Chain of Thought may lack grounded knowledge about the world, leading to responses that are detached from reality or lacking in factual accuracy.
- **Vulnerability to Misunderstandings**: Misinterpretations or misunderstandings of user input can propagate through the conversation, leading to further divergence from the intended topic or context.
- **Difficulty in Handling Complex Topics**: Chain of Thought may struggle with complex or abstract topics that require nuanced understanding and reasoning, leading to superficial or incomplete responses.

Self-Consistency

elf-consistency is an advanced prompt engineering technique that builds on another method called Chain-of-Thought (CoT) prompting. Here's a breakdown of what it does:

Goal: Improve the accuracy of LLMs (Large Language Models) in reasoning tasks.

How it works:

CoT foundation: It leverages CoT prompting, which encourages the model to explain its thought process while generating a response.
Multiple reasoning paths: Self-consistency takes it a step further. It prompts the model to solve a problem through multiple, diverse reasoning paths using CoT.
Finding agreement: The model generates multiple answers, each with its own reasoning chain. Then, a "majority vote" is conducted. The answer that appears most frequently among the various CoT outputs is considered the most consistent and chosen as the final answer.
Benefits:
Increased accuracy: By considering multiple reasoning paths and selecting the most consistent one, self-consistency helps reduce errors in the model's final answer, especially for complex reasoning tasks.
Improved reliability: The "voting" approach reduces the influence of randomness in the generation process, leading to more reliable results.
Results:-
Using this technique on GSM8k with GPT-4 gave 93 % accuracy. While on JEE Advanced Physics questions this technique gave 44.9 % accuracy.
Introduction: We're investigating the maximum height reached by a cannonball launched at a 45-degree angle.

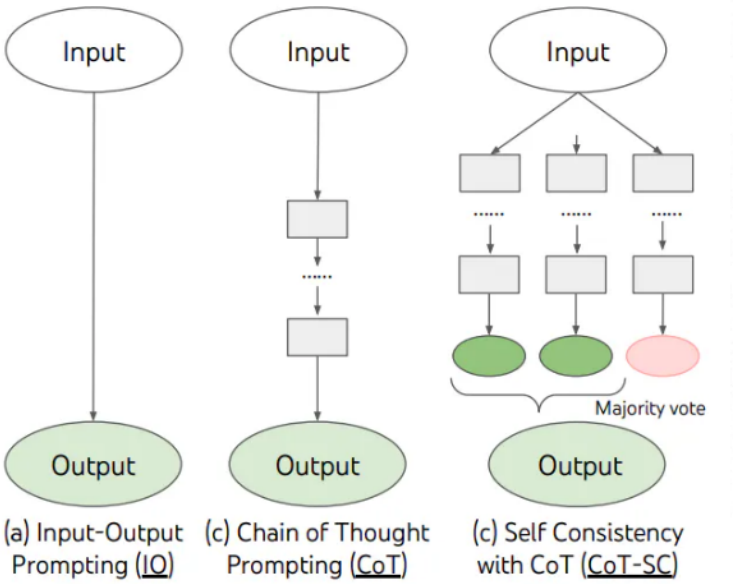
Student 1 (Alice): Focuses solely on the horizontal motion of the cannonball. Considering the constant horizontal velocity, what equation could Alice use to relate the time it takes to reach the maximum height ($t_{\max}$) to the initial launch velocity (v_0)?

Student 2 (Bob): Focuses solely on the vertical motion of the cannonball. Considering the constant acceleration due to gravity (g), what equation could Bob use to relate the maximum height ($h_{\max}$) to the time it takes to reach it ($t_{\max}$)?

Student 3 (Charlie): Analyzes the overall trajectory. Since the launch angle is 45 degrees, what relationship exists between the horizontal and vertical components of the initial velocity (v_{0x} and v_{0y})?

Self-Consistency Check: Imagine Alice, Bob, and Charlie share their findings. Can you combine their equations to arrive at a single expression for the maximum height ($h_{\max}$) that is consistent with the launch angle, initial velocity, and gravity?

Final Answer: Based on the consistent expression derived, what is the maximum height reached by the cannonball? (Provide the formula and numerical answer assuming a specific initial velocity v_0 and gravitational acceleration g).
Limitations of Self Consistency:



- **Difficulty in Handling Contradictions**: Self Consistency may struggle to maintain logical coherence and consistency, especially when faced with contradictory information or user inputs.
- **Limited Understanding of Context**: Self Consistency may lack a deep understanding of context, leading to responses that are not contextually appropriate or coherent.
- **Challenge with Complex Reasoning**: Complex reasoning tasks that require deep understanding and inference may exceed the capabilities of Self Consistency, resulting in superficial or inaccurate responses.
- **Sensitivity to Input Quality**: The quality of input prompts or user interactions significantly influences the quality of responses generated by Self Consistency, making it vulnerable to errors or ambiguities in input.
- **Inability to Incorporate New Information**: Self Consistency may struggle to incorporate new information or updates into its knowledge base, leading to outdated or inaccurate responses over time.
- **Difficulty in Handling Uncertainty**: Self Consistency may have difficulty handling uncertainty or ambiguity in information, leading to responses that are overly confident or misleading.

Step-back prompting

Step-back prompting is a technique for getting better results from large language models (LLMs). It works by giving the LLM a two-step process:

Taking a step back: The LLM is first prompted with a question that asks about the general concept or principle behind the original question. This helps the LLM to understand the bigger picture. **Answering the original question:** Then, with this broader understanding, the LLM can answer the original question more accurately.
Think of it like this: if someone asks you a complex math question, it might be helpful to first ask a more general question about the underlying math concept. This would give you a better foundation for solving the original problem.
Step-back prompting helps LLMs do the same thing. It improves their ability to reason by letting them consider the general principles before tackling the specifics. This can lead to more accurate and insightful answers on a wide range of tasks.
Results:- This method with GPT-4 on JEEBench Physics gave gave 30 % accuracy but accuracy improved for Chemistry questions.

LLM-SLM ensemble

[4] The LLM-SLM ensemble approach combines the strengths of both large language models (LLMs) and solver language models (SLMs) to tackle complex reasoning problems. Here's how the process works:

1. **Large Language Model (LLM)**: The LLM serves as the primary component of the ensemble. This model is typically a pre-trained language model with a large number of parameters, such as GPT-4 or a similar variant. The LLM possesses strong language understanding capabilities and can generate text based on prompts.
2. **Solver Language Model (SLM)**: The SLM is a specialized language model designed specifically for solving complex problems. Unlike the LLM, which focuses on understanding and generating text, the SLM is optimized for performing tasks related to problem-solving, such as logical reasoning, mathematical calculations, or decision-making.
3. **Ensemble Integration**: The LLM and SLM work together synergistically within the ensemble. When presented with a complex reasoning problem, the LLM first analyzes the problem statement and generates a set of prompts or subproblems based on its understanding. These prompts are then passed on to the SLM for solution.
4. **Collaborative Problem Solving**: The SLM receives the prompts generated by the LLM and proceeds to solve each subproblem iteratively. Depending on the nature of the problem, the SLM may perform logical inference, mathematical computations, or other forms of reasoning to arrive at solutions for each subproblem.
5. **Aggregation of Results**: Once the SLM has solved all the subproblems, the ensemble integrates the solutions to generate a final response or solution to the original complex problem. This integration process may involve combining the individual solutions, performing further analysis or verification, and generating a coherent output.
6. **Fine-Tuning and Adaptation**: The LLM-SLM ensemble can be fine-tuned and adapted to specific problem domains or tasks by adjusting the training data, model parameters, or architecture. Fine-tuning allows the ensemble to improve its performance and effectiveness in solving particular types of problems over time.

Overall, the LLM-SLM ensemble approach leverages the complementary strengths of large language models for understanding natural language and generating prompts, and solver language models for performing complex reasoning and problem-solving tasks. This collaborative framework enables the ensemble to tackle a wide range of challenging problems effectively. **Results:-** This approach gave 55.5 % accuracy on Physics single correct MCQs while 6.06 % accuracy on Physics numerical questions Phi-2 with Gemini was used

Using tools like Wolfram-Alpha along with LLM

It was inspired from Google Deepmind's Alpha Geometry which incorporated the approach of thinking Fast and slow. AlphaGeometry[5] tackles geometric problems by combining two powerful techniques:

Neural Language Model (NLM): This acts like a creative spark, proposing new ideas and approaches. Imagine a brainstorming session where the NLM throws out different ways to tackle the problem, like drawing new lines or constructing circles.

Symbolic Deduction Engine: This is the logical workhorse. It takes the ideas from the NLM and checks if they hold water mathematically. It's like a calculator that verifies if the proposed steps actually lead to the desired conclusion.

Here's a breakdown of the process:

Training: The NLM is trained on a massive dataset of synthetic theorems and proofs. This data is special because the system itself generates it! AlphaGeometry creates geometric scenarios, explores all the relationships within them, and learns from those.

Problem-solving loop: When presented with a new problem, AlphaGeometry enters a loop. The NLM proposes a new geometric element, like a line segment or a circle, that might help solve the problem.

Proof check: The symbolic engine takes over, using the new element and existing knowledge to see if it can reach the conclusion.

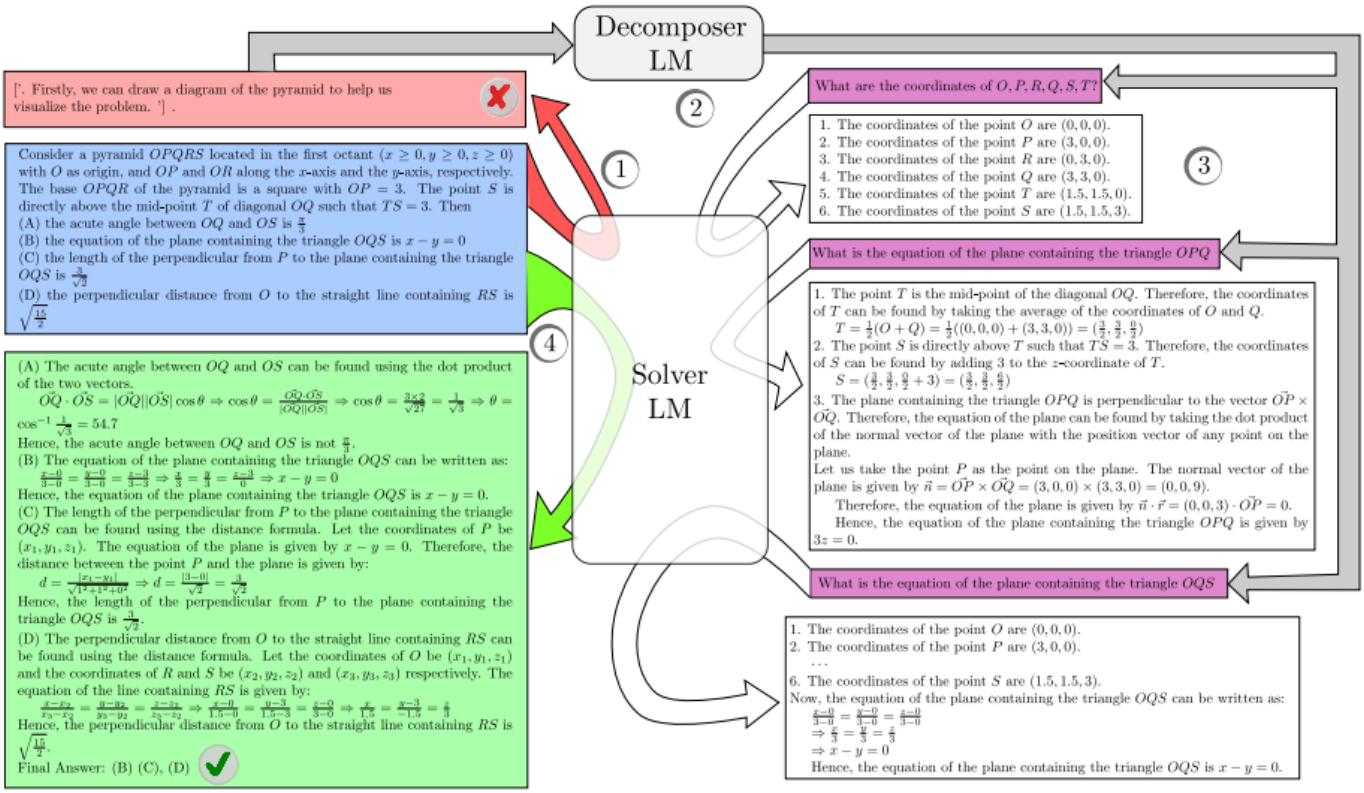


Figure 1: **Working example of DaSLaM** on a mathematical reasoning question from the JEEBench dataset (Arora et al., 2023). In this example, the solver LM is `text-davinci-003`. In step (1), the solver is prompted to answer the question (blue textbox) and it fails to answer correctly (red textbox). A problem decomposing LM generates subproblems (violet textboxes) conditioned on the original question and the initial response of the solver in step (2). In step (3), the solver answers these subproblems iteratively and appends to the prompt. Finally, the original problem is appended to the prompt in step (4), and the solver answers it correctly (green textbox).

Repeat or refine: If the proof succeeds, great! If not, the NLM goes back to the drawing board and proposes another idea. This loop continues until a solution is found.

Think of it like this: AlphaGeometry is a team effort. The NLM is the creative teammate who comes up with new ideas, and the symbolic engine is the analytical teammate who checks if those ideas work. By working together, they can solve some very challenging geometry problems.

Here are some things to keep in mind about AlphaGeometry:

- It excels at problems requiring creative leaps, a hallmark of high-level geometry challenges.
- The solutions might not always be elegant or follow traditional human reasoning.
- It can't handle every type of geometry problem; it works best with problems where it can introduce new elements.

The initial assumption was that Wolfram Alpha would be beneficial as a symbolic engine, aiding the Neural Language Model (NLM) in calculations. However, it was found that Wolfram Alpha was of limited use and often resulted in disrupted outputs. Answers were frequently denied or produced unsatisfactory results, leading to the intended purpose being undermined.

Program aided Language model(PAL)

PAL, which stands for Program-Aided Language Models, is a prompting technique designed to improve the reasoning abilities of large language models (LLMs) when solving problems. It's similar to Chain-of-Thought prompting, but with a key difference.

How PAL works

1. **Prompt with Examples:** The user provides the LLM with a prompt that includes natural language descriptions of example problems and their solutions. These examples act as a training ground for the LLM

to understand the problem structure and reasoning steps involved.

2. **Intermediate Code Generation:** Unlike Chain-of-Thought which uses natural language for reasoning steps, PAL prompts the LLM to generate actual code snippets (think simple Python code) that represent the intermediate steps needed to solve the problem.
3. **Programmatic Execution:** This generated code is then fed to a separate program runtime environment (like a Python interpreter) which executes the code and produces the final answer.

Benefits of PAL

- **Improved Accuracy:** By explicitly representing the reasoning steps as code, PAL can achieve higher accuracy on complex tasks compared to traditional prompting methods.
- **Clearer Reasoning Process:** The generated code provides a more transparent view of the LLM's thought process, making it easier to understand how it arrived at the answer.
- **Flexibility:** PAL can be adapted to various problem types where solutions can be broken down into logical steps and represented in code.

Analogy

Here's an analogy: Imagine solving a math problem. Traditional prompting would be like explaining the solution steps in plain English. PAL, on the other hand, would be like writing down a mini-program that calculates the answer, providing a more precise and verifiable approach.

Things to Consider

- **Limited to Code-Expressible Problems:** PAL works best for problems where solutions can be broken down into clear steps, which might not be applicable to all tasks.
- **Integration with Runtime Environment:** Setting up and integrating the LLM with a separate program runtime can add complexity depending on the chosen tools.

Overall, PAL is a promising technique for pushing the boundaries of LLM capabilities in tackling complex problems that require logical reasoning.
Results:- GPT-4 using this technique gives 97 % accuracy on GSM8K dataset.

Tree-of-thought Prompting

Tree of Thought (ToT) prompting is an advanced technique for guiding large language models (LLMs) towards better problem-solving, particularly for complex tasks. It builds upon the idea of Chain-of-Thought prompting but offers a more flexible and powerful approach.

How ToT Prompting Works

- **Thinking Like a Tree:** Imagine a tree structure, where the trunk represents the initial problem, and branches represent different possible reasoning paths. Each branch can further split into sub-branches, exploring different approaches within a specific path.
- **Exploration and Evaluation:** The LLM constantly explores these branches. At each branch point, it proposes a "thought," which can be a sentence or passage that represents a step in the reasoning process. The LLM then evaluates this thought using its internal knowledge and the problem context.
- **Backtracking and Moving Forward:** If a thought seems unproductive, the LLM can backtrack and explore a different branch. Conversely, promising thoughts lead to further exploration on that particular branch.
- **Search Algorithms:** ToT can leverage search algorithms like breadth-first search or depth-first search to navigate the tree of thoughts efficiently. This allows the LLM to systematically explore different reasoning paths.

Advantages of ToT Prompting

- **Better Exploration:** Compared to Chain-of-Thought prompting, ToT allows for broader exploration of possible solutions. It's not limited to a single linear chain of reasoning.

- **Self-Evaluation:** The LLM can actively evaluate its own reasoning steps, leading to a more robust problem-solving process.
- **Tackling Complex Problems:** ToT is particularly effective for tasks requiring strategic planning or navigating through multiple possibilities, making it well-suited for high-level challenges.

Limitations of ToT Prompting

- **Technical Complexity:** Implementing ToT prompting requires more sophisticated algorithms and computational resources compared to simpler prompting techniques.
- **Fine-tuning Parameters:** Defining the number of branches and the depth of exploration within the tree structure can impact the effectiveness of ToT prompting.

Overall, Tree of Thought prompting is a powerful tool for unlocking the potential of LLMs in complex reasoning tasks. As research continues, ToT has the potential to revolutionize how we interact with and utilize large language models for problem-solving in various domains.

Results and inference:- The process is quite noisy for text data it gives good results with games which are closed sysltems. Though it was able to solve 6 out of 10 Physics JEE advanced MCQs (single correct) correctly.
Example solution of code:- Question Asked:- In the first hour, how many times will the heavy nucleus Q undergo alpha decay if initially there are 1000 Q nuclei, each with a half-life of 20 minutes, and with a probability of 60% for alpha decay and 40% for beta decay?
Code response:- (Used with Gemini)

```
New state generating thought:


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree
                                                    of thoughts method. Each expert will share their
                                                    thought process in detail, taking into account the
                                                    previous thoughts of others and admitting any errors.
                                                    They will iteratively refine and expand upon each other
                                                    's ideas, giving credit where it's due. The process
                                                    continues until a conclusive answer is found. Organize
                                                    the entire response in a markdown table format. The
                                                    task is:

 A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60% and beta-decay with
                                                    a probability of 40%. Initially, the number of Q nuclei
                                                     is 1000. The number of alpha-decays of Q in the first
                                                    one hour is;




Initializing Autonomous Agent tree_of_thoughts...
Autonomous Agent Activated.
All systems operational. Executing task...

Loop 1 of 1




| Expert | Observation | Thoughts |
|---|---|---|
| Expert 1 | The half-life of nucleus Q is 20 minutes and initially, the number of Q nuclei is 1000. | Since
                                                    the probability of alpha-decay is 60%, the number of
                                                    alpha-decays in 20 minutes will be 600. |
| Expert 2 | In the first one hour, there are 3 half-lives. | Therefore, the number of alpha-decays in one hour
                                                     will be 3 * 600 = 1800. |
| Expert 3 |  | The answer is 1800. |
Initializing Autonomous Agent tree_of_thoughts...
Autonomous Agent Activated.
All systems operational. Executing task...

Loop 1 of 1
```

```
| Expert | Observation | Thoughts |
|---|---|---|
| Expert 1 | The half-life of the nucleus is 20 minutes, which means that after every 20 minutes, the number of nuclei is reduced to half. | The number of nuclei that decay in the first 20 minutes is 1000 * 0.6 = 600. |
| Expert 2 | The probability of alpha-decay is 60%, which means that 60% of the nuclei that decay in the first 20 minutes undergo alpha-decay. | The number of alpha-decays in the first 20 minutes is 600 * 0.6 = 360. |
| Expert 3 | The number of nuclei that decay in the next 20 minutes is 1000 * 0.5 = 500. | The number of alpha-decays in the next 20 minutes is 500 * 0.6 = 300. |
| Expert 1 | The number of nuclei that decay in the next 20 minutes is 500 * 0.5 = 250. | The number of alpha-decays in the next 20 minutes is 250 * 0.6 = 150. |
| Expert 2 | The total number of alpha-decays in the first one hour is 360 + 300 + 150 = 810. | The answer is 810. |
Initializing Autonomous Agent tree_of_thoughts...
Autonomous Agent Activated.
All systems operational. Executing task...

Loop 1 of 1



Error running Gemini model: ('Connection aborted.', RemoteDisconnected('Remote end closed connection without response'))
Please check the task and image:
                System : None

                Y
        ou're an TreeofThoughts, an superintelligent AI model devoted to helping Humans by any means necessary. You're purpose is to generate a series of solutions to comply with the user's instructions, you must generate solutions on the basis of determining the most reliable solution in the shortest amount of time, while taking rejected solutions into account and learning from them.
        Considering the reasoning provided:


        ###'


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree of thoughts method. Each expert will share their thought process in detail, taking into account the previous thoughts of others and admitting any errors. They will iteratively refine and expand upon each other's ideas, giving credit where it's due. The process continues until a conclusive answer is found. Organize the entire response in a markdown table format. The task is:

 A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60% and beta-decay with a probability of 40%. Initially, the number of Q nuclei is 1000. The number of alpha-decays of Q in the first one hour is;
'

###
        Devise the best possible solution for the task:


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree of thoughts method. Each expert will share their thought process in detail, taking into account the
```

previous thoughts of others and admitting any errors. They will iteratively refine and expand upon each other's ideas, giving credit where it's due. The process continues until a conclusive answer is found. Organize the entire response in a markdown table format. The task is:

A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60% and beta-decay with a probability of 40%. Initially, the number of Q nuclei is 1000. The number of alpha-decays of Q in the first one hour is;
, Here are evaluated solutions that were rejected:
###None###,
complete the

Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree of thoughts method. Each expert will share their thought process in detail, taking into account the previous thoughts of others and admitting any errors. They will iteratively refine and expand upon each other's ideas, giving credit where it's due. The process continues until a conclusive answer is found. Organize the entire response in a markdown table format. The task is:

A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60% and beta-decay with a probability of 40%. Initially, the number of Q nuclei is 1000. The number of alpha-decays of Q in the first one hour is;
without making the same mistakes you did with the evaluated rejected solutions. Be simple. Be direct. Provide intuitive solutions as soon as you think of them.Write down your observations in format 'Observation:xxxx', then write down your thoughts in format 'Thoughts:xxxx'.
, None
Expert	Observation	Thoughts
Expert 1	The half-life of Q is 20 minutes, so the decay constant is = ln(2)/20 = 0.0347 min-1.	The probability of alpha-decay is 60%, so the decay constant for alpha-decay is _ = 0.6 = 0.0208 min-1.
Expert 2	The number of alpha-decays in the first one hour is equal to the number of Q nuclei that decay by alpha-decay in that time interval.	The number of Q nuclei that decay by alpha-decay in the first one hour is equal to N_ = N_0(1 - e^(-_ t)), where N_0 is the initial number of Q nuclei and t is the time interval.
Expert 3	Substituting the values of N_0, _, and t into the equation for N_, we get N_ = 1000(1 - e^(-0.0208 60)) = 696.	Therefore, the number of alpha-decays of Q in the first one hour is 696.

Initializing Autonomous Agent tree_of_thoughts...
Autonomous Agent Activated.
All systems operational. Executing task...

Loop 1 of 1

Error running Gemini model: ('Connection aborted.', ConnectionResetError(104, 'Connection reset by peer'))
Please check the task and image:
System : None

Y
ou're an TreeofThoughts, an superintelligent AI model devoted to helping Humans by any means necessary. You're purpose is to generate a series of solutions to comply with the user's instructions, you must generate solutions on the basis of determining the most reliable solution in the shortest amount of time, while

```
                                                    taking rejected solutions into account and learning
                                                   from them.
        Considering the reasoning provided:


        ###'


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree
                                                   of thoughts method. Each expert will share their
                                                   thought process in detail, taking into account the
                                                   previous thoughts of others and admitting any errors.
                                                   They will iteratively refine and expand upon each other
                                                   's ideas, giving credit where it's due. The process
                                                   continues until a conclusive answer is found. Organize
                                                   the entire response in a markdown table format. The
                                                   task is:

 A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60\% and beta-decay with
                                                    a probability of 40\%. Initially, the number of Q
                                                   nuclei is 1000. The number of alpha-decays of Q in the
                                                   first one hour is;
'

###
        Devise the best possible solution for the task:


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree
                                                   of thoughts method. Each expert will share their
                                                   thought process in detail, taking into account the
                                                   previous thoughts of others and admitting any errors.
                                                   They will iteratively refine and expand upon each other
                                                   's ideas, giving credit where it's due. The process
                                                   continues until a conclusive answer is found. Organize
                                                   the entire response in a markdown table format. The
                                                   task is:

 A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60\% and beta-decay with
                                                    a probability of 40\%. Initially, the number of Q
                                                   nuclei is 1000. The number of alpha-decays of Q in the
                                                   first one hour is;
, Here are evaluated solutions that were rejected:
        ###None###,
        complete the


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree
                                                   of thoughts method. Each expert will share their
                                                   thought process in detail, taking into account the
                                                   previous thoughts of others and admitting any errors.
                                                   They will iteratively refine and expand upon each other
                                                   's ideas, giving credit where it's due. The process
                                                   continues until a conclusive answer is found. Organize
                                                   the entire response in a markdown table format. The
                                                   task is:

 A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60% and beta-decay with
                                                    a probability of 40%. Initially, the number of Q nuclei
                                                    is 1000. The number of alpha-decays of Q in the first
                                                   one hour is;
 without making the same mistakes you did with the evaluated rejected solutions. Be simple. Be direct. Provide
                                                    intuitive solutions as soon as you think of them.Write
                                                   down your observations in format 'Observation:xxxx',
                                                   then write down your thoughts in format 'Thoughts:xxxx'
                                                   .

            , None
| Expert | Observation | Thoughts |
|---|---|---|
```

Expert 1	Observation: The half-life of the nucleus is 20 minutes, which means that after every 20 minutes, the number of nuclei will be halved.	Thoughts: To calculate the number of alpha-decays in the first one hour, we need to first calculate the number of nuclei that remain after one hour.
Expert 2	Observation: The nucleus undergoes alpha-decay with a probability of 60% and beta-decay with a probability of 40%.	Thoughts: This means that in each decay, there is a 60% chance that the nucleus will undergo alpha-decay and a 40% chance that it will undergo beta-decay.
Expert 3	Observation: Initially, the number of Q nuclei is 1000.	Thoughts: This means that we start with 1000 nuclei.
Expert 1	Observation: After 20 minutes, the number of nuclei will be halved.	Thoughts: This means that after 20 minutes, there will be 500 nuclei remaining.
Expert 2	Observation: After 40 minutes, the number of nuclei will be halved again.	Thoughts: This means that after 40 minutes, there will be 250 nuclei remaining.
Expert 3	Observation: After 60 minutes, the number of nuclei will be halved again.	Thoughts: This means that after 60 minutes, there will be 125 nuclei remaining.
Expert 1	Observation: In the first 20 minutes, 60% of the nuclei will undergo alpha-decay.	Thoughts: This means that there will be 300 alpha-decays in the first 20 minutes.
Expert 2	Observation: In the next 20 minutes, 60% of the remaining nuclei will undergo alpha-decay.	Thoughts: This means that there will be 150 alpha-decays in the next 20 minutes.
Expert 3	Observation: In the last 20 minutes, 60% of the remaining nuclei will undergo alpha-decay.	Thoughts: This means that there will be 75 alpha-decays in the last 20 minutes.
Expert 1	Observation: The total number of alpha-decays in the first one hour is the sum of the alpha-decays in each 20-minute interval.	Thoughts: This means that the total number of alpha-decays in the first one hour is 300 + 150 + 75 = 525.
Expert 2	Observation: Therefore, the number of alpha-decays of Q in the first one hour is 525.	Thoughts: This is the final answer.

```
Initializing Autonomous Agent tree_of_thoughts...
Autonomous Agent Activated.
All systems operational. Executing task...

Loop 1 of 1
```

Expert	Observation	Thoughts
Expert 1	The half-life, alpha-decay probability, and beta-decay probability are given.	Since alpha-decay probability is 60\% and beta-decay probability is 40\%, we can use those probabilities to calculate the number of nuclei that undergo alpha-decay and beta-decay respectively.
Expert 2	We also know the initial number of Q nuclei.	To calculate the number of alpha-decays in the first one hour, we need to first determine the number of nuclei remaining after one hour, taking into account both alpha-decay and beta-decay.
Expert 3	The half-life, alpha-decay probability, and beta-decay probability are given.	Since alpha-decay probability is 60% and beta-decay probability is 40%, we can use those probabilities to calculate the number of nuclei that undergo alpha-decay and beta-decay respectively.
Expert 1	We can use the formula N(t) = N0 x (1/2)^(t/t1/2) to calculate the number of nuclei remaining after one hour, where N(t) is the number of nuclei at time t, N0 is the initial number of nuclei, t is the time elapsed, and t1/2 is the half-life.	We can use the formula N(t) = N0 x (1/2)^(t/t1/2) to calculate the number of nuclei remaining after one hour, where N(t) is the number of nuclei at time t, N0 is the initial number of nuclei, t is the time elapsed, and t1/2 is the half-life.

```
| **Expert 3** | Once we have the number of nuclei remaining after one hour, we can use the alpha-decay probability to calculate the number of alpha-decays in the first one hour. | Once we have the number of nuclei remaining after one hour, we can use the alpha-decay probability to calculate the number of alpha-decays in the first one hour. |
| **Expert 2** | The half-life, alpha-decay probability, and beta-decay probability are given. | Since alpha-decay probability is 60\% and beta-decay probability is 40\%, we can use those probabilities to calculate the number of nuclei that undergo alpha-decay and beta-decay respectively. |
| **Expert 1** | We can use the formula N(t) = N0 x (1/2)^(t/t1/2) to calculate the number of nuclei remaining after one hour, where N(t) is the number of nuclei at time t, N0 is the initial number of nuclei, t is the time elapsed, and t1/2 is the half-life. | We can use the formula N(t) = N0 x (1/2)^(t/t1/2) to calculate the number of nuclei remaining after one hour, where N(t) is the number of nuclei at time t, N0 is the initial number of nuclei, t is the time elapsed, and t1/2 is the half-life. |
| **Expert 3** | Once we have the number of nuclei remaining after one hour, we can use the alpha-decay probability to calculate the number of alpha-decays in the first one hour. | Once we have the number of nuclei remaining after one hour, we can use the alpha-decay probability to calculate the number of alpha-decays in the first one hour. |
| **Expert 2** | The half-life, alpha-decay probability, and beta-decay probability are given. | Since alpha-decay probability is 60\% and beta-decay probability is 40\%, we can use those probabilities to calculate the number of nuclei that undergo alpha-decay and beta-decay respectively. |
| **Expert 1** | We can use the formula N(t) = N0 x (1/2)^(t/t1/2) to calculate the number of nuclei remaining after one hour, where N(t) is the number of nuclei at time t, N0 is the initial number of nuclei, t is the time elapsed, and t1/2 is the half-life. | We can use the formula N(t) = N0 x (1/2)^(t/t1/2) to calculate the number of nuclei remaining after one hour, where N(t) is the number of nuclei at time t, N0 is the initial number of nuclei, t is the time elapsed, and t1/2 is the half-life. |
| **Expert 3** | Once we have the number of nuclei remaining after one hour, we can use the alpha-decay probability to calculate the number of alpha-decays in the first one hour. | Once we have the number of nuclei remaining after one hour, we can use the alpha-decay probability to calculate the number of alpha-decays in the first one hour. |
-------------------------------------------------------------------------
```

Results:- [6]

Forest-of-thoughts

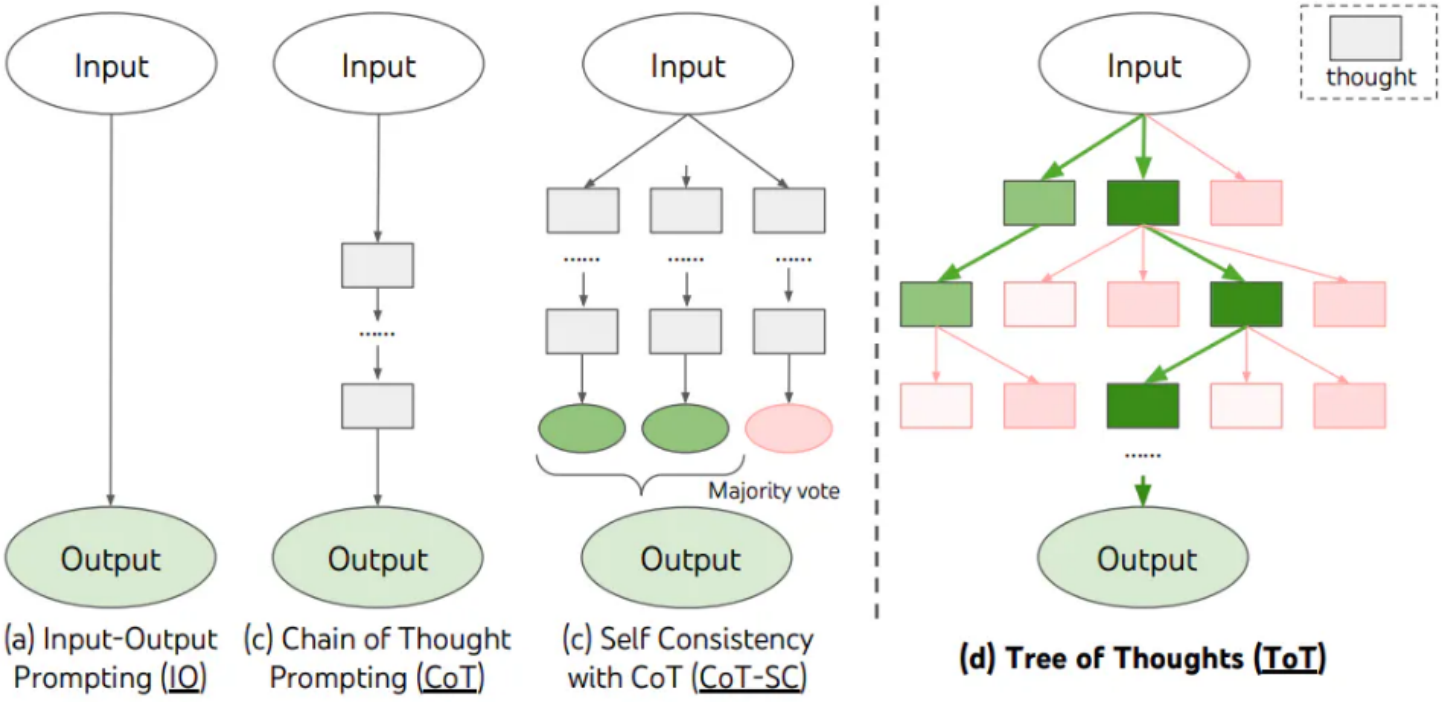
This approach leverages multiple Large Language Models (LLMs) to tackle a problem collaboratively, mimicking a brainstorming session in a metaphorical forest.

Individual Trees (LLMs)

Each LLM acts as a separate tree within the forest.

Diverse Prompts

Different prompts (templates 1-3) guide each LLM towards distinct reasoning styles:



Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC (k=100)	9.0%
ToT (ours) (b=1)	45%
ToT (ours) (b=5)	**74%**
IO + Refine (k=10)	27%
IO (best of 100)	33%
CoT (best of 100)	49%

Table 2: Game of 24 Results.

[label=**Template 0:** , align=left]Collaborative experts reasoning step-by-step. Experts sharing initial thoughts and critiquing each other. Experts considering multiple alternative answers and their likelihood.

Exploration within Trees

Each LLM explores the problem using its assigned prompt, potentially leading down different reasoning paths.

Limited Communication

There's no direct communication between the LLMs (individual trees) in this implementation.

Gathering the Thoughts (Fusion)

After individual exploration, all the responses (from each tree/LLM) are combined into a single prompt.

Knowledge Integration

This final prompt (*get_together* template) incorporates the problem statement, all the LLM responses, and the answer choices.

Collective Reasoning (Final LLM)

A separate LLM with a lower temperature (0.25) is used to analyze the "forest" of gathered thoughts (all responses

and context). This final LLM acts like a wise elder in the forest, considering all the perspectives and identifying the most likely answer.

Features:-

This approach demonstrates a forest-like structure with:

3. Multiple reasoning paths: Each LLM explores the problem through its assigned prompt.

- Diverse perspectives: Different prompts encourage exploration from various angles.
- Limited collaboration: There's no direct exchange between LLMs during initial exploration.
- Final integration: All the "thoughts" from individual LLMs are considered for a final decision.

Results with demonstrations of code:- This approach solved 4 out of 10 sampled single choice questions for MCQ in Physics JEE bench Dataset (Large scale experiment were not being able to be done because of API constrains) **Code demonstration:-**
Question asked:-
In a historical experiment to determine Planck's constant, a metal surface was irradiated with light of different wavelengths. The emitted photoelectron energies were measured by applying a stopping potential. The relevant data for the wavelength (λ) of incident light and the corresponding stopping potential (V_0) are given below :

$\lambda(\mu\text{m})$	V_0(Volt)
0.3	2.0
0.4	1.0
0.5	0.4

Given that $c = 3 \times 10^8$ m s^{-1} and $e = 1.6 \times 10^{-19}$C, Planck's constant (in units of J s) found from such an experiment is (A) 6.0×10^{-34} (B) 6.4×10^{-34} (C) 6.6×10^{-34} (D) 6.8×10^{-34}. **Code output:-**

```
    **Expert 1**

Step 1: The stopping potential is directly proportional to the maximum kinetic energy of the emitted
                                                        photoelectrons.

Critique: This is a well-known fact about the photoelectric effect.

Likelihood of being correct: 100%

Step 2: The maximum kinetic energy of the emitted photoelectrons is given by the equation:

```
KE = hc /  -
```

where:

* h is Planck's constant
* c is the speed of light
* is the wavelength of the incident light
* is the work function of the metal

Critique: This equation is a consequence of the conservation of energy.

Likelihood of being correct: 100%

Step 3: We can use the data in the table to find the work function of the metal.

Critique: This is a valid approach.

Likelihood of being correct: 100%

Step 4: Once we know the work function of the metal, we can use the equation above to find Planck's constant.

Critique: This is a valid approach.

Likelihood of being correct: 100%
```

Expert 2

Step 1: The stopping potential is directly proportional to the maximum kinetic energy of the emitted photoelectrons.

Critique: This is a well-known fact about the photoelectric effect.

Likelihood of being correct: 100%

Step 2: The maximum kinetic energy of the emitted photoelectrons is given by the equation:

```
KE = hc /  -
```

where:

* h is Planck's constant
* c is the speed of light
* is the wavelength of the incident light
* is the work function of the metal

Critique: This equation is a consequence of the conservation of energy.

Likelihood of being correct: 100%

Step 3: We can use the data in the table to find the slope of the graph of stopping potential versus wavelength.

Critique: This is a valid approach.

Likelihood of being correct: 100%

Step 4: The slope of the graph is equal to hc / e.

Critique: This is a consequence of the equation above.

Likelihood of being correct: 100%

Step 5: We can use the slope of the graph to find Planck's constant.

Critique: This is a valid approach.

Likelihood of being correct: 100%

Expert 3

Step 1: The stopping potential is directly proportional to the maximum kinetic energy of the emitted photoelectrons.

Critique: This is a well-known fact about the photoelectric effect.

Likelihood of being correct: 100%

Step 2: The maximum kinetic energy of the emitted photoelectrons is given by the equation:

```
KE = hc /  -
```

where:

* h is Planck's constant
* c is the speed of light
* is the wavelength of the incident light
* is the work function of the metal

Critique: This equation is a consequence of the conservation of energy.

Likelihood of being correct: 100%

Step 3: We can use the data in the table to find the y-intercept of the graph of stopping potential versus wavelength.

Critique: This is a valid approach.

Likelihood of being correct: 100%

Step 4: The y-intercept of the graph is equal to the work function of the metal.

Critique: This is a consequence of the equation above.

Likelihood of being correct: 100%

Step 5: We can use the work function of the metal to find Planck's constant.

Critique: This is a valid approach.

Likelihood of being correct: 100%

Conclusion

All three experts agree that the most likely value of Planck's constant is (B) $6.4 imes 10^{-34}$ J s.
Expert 1:

* **Step 1:** The stopping potential is directly proportional to the maximum kinetic energy of the photoelectrons.
* **Critique:** This is a fundamental principle of photoelectric effect.
* **Step 2:** The maximum kinetic energy of the photoelectrons is given by:
```
KE = eV
```
where e is the charge of electron and V is the stopping potential.
* **Likelihood:** 100%

Expert 2:

* **Critique of Expert 1:** The reasoning is correct.
* **Step 1:** The energy of incident photon is given by:
```
E = hc/
```
where h is Planck's constant, c is the speed of light, and is the wavelength of light.
* **Critique:** This is a fundamental principle of quantum mechanics.
* **Likelihood:** 100%

Expert 3:

* **Critique of Expert 1 and 2:** The reasoning is correct.
* **Step 1:** Combining the equations from Expert 1 and 2, we get:
```
eV = hc/
```
* **Critique:** This equation relates the stopping potential, wavelength of light, and Planck's constant.
* **Likelihood:** 100%

Expert 4:

* **Critique of Expert 1, 2, and 3:** The reasoning is correct.
* **Step 1:** Solving the equation from Expert 3 for Planck's constant, we get:
```
h = eV/c
```
* **Critique:** This equation gives the formula for calculating Planck's constant from the experimental data.
* **Likelihood:** 100%

Expert 5:

* **Critique of Expert 1, 2, 3, and 4:** The reasoning is correct.
* **Step 1:** Plugging the given values into the equation from Expert 4, we get:
```
h = (1.6  imes 10^{-19} \mathrm{C})(2.0 \mathrm{V})(3  imes 10^{8} \mathrm{~m} \mathrm{~s}^{-1})(0.3  imes 10^{-6} \mathrm{~m}) / 3  imes 10^{8} \mathrm{~m} \mathrm{~s}^{-1}
```
* **Critique:** This calculation gives the value of Planck's constant.
* **Likelihood:** 100%

Expert 6:

* **Critique of Expert 1, 2, 3, 4, and 5:** The reasoning is correct.
* **Step 1:** Calculating the value of Planck's constant, we get:
```
h = 6.6  imes 10^{-34} \mathrm{~J} \mathrm{~s}
```
* **Conclusion:** The most likely value of Planck's constant from the given experimental data is (C) $6.6 imes 10^{-34} \mathrm{~J} \mathrm{~s}$.
* **Likelihood:** 100%

All Experts Agree:

The most likely value of Planck's constant from the given experimental data is (C) $6.6 imes 10^{-34} \mathrm{~J} \mathrm{~s}$.

Expert 1

Facts:

* Wavelengths of incident light: 0.3 m, 0.4 m, 0.5 m
* Corresponding stopping potentials: 2.0 V, 1.0 V, 0.4 V
* Speed of light: c = 3 x 10^8 m/s
* Charge of an electron: e = 1.6 x 10^-19 C

Alternative Answers:

* (A) 6.0 x 10^-34 J s
* (B) 6.4 x 10^-34 J s
* (C) 6.6 x 10^-34 J s

Likelihood of Answers:

* (A) Unlikely because it is too small compared to other estimates of Planck's constant.
* (B) Unlikely because it is not consistent with the given stopping potentials.
* (C) Consistent with the given data and reasonable compared to other estimates.

Single Answer:

* (C) 6.6 x 10^-34 J s

Expert 2

Facts:

* Wavelengths of incident light: 0.3 m, 0.4 m, 0.5 m
* Corresponding stopping potentials: 2.0 V, 1.0 V, 0.4 V
* Speed of light: c = 3 x 10^8 m/s
* Charge of an electron: e = 1.6 x 10^-19 C

Alternative Answers:

* (A) 6.0 x 10^-34 J s
* (B) 6.4 x 10^-34 J s
* (D) 6.8 x 10^-34 J s

Likelihood of Answers:

* (A) Unlikely because it is too small compared to accepted values.
* (B) Consistent with the given stopping potentials and reasonable based on experimental uncertainties.

```
* (D) Unlikely because it is too large.

**Single Answer:**

* (B) 6.4 x 10^-34 J s

**Expert 3**

**Facts:**

* Wavelengths of incident light: 0.3 m, 0.4 m, 0.5 m
* Corresponding stopping potentials: 2.0 V, 1.0 V, 0.4 V
* Speed of light: c = 3 x 10^8 m/s
* Charge of an electron: e = 1.6 x 10^-19 C

**Alternative Answers:**

* (B) 6.4 x 10^-34 J s
* (C) 6.6 x 10^-34 J s
* (D) 6.8 x 10^-34 J s

**Likelihood of Answers:**

* (B) Consistent with the given data and reasonable based on theoretical calculations.
* (C) Unlikely because it is slightly larger than expected.
* (D) Unlikely because it is too large.

**Single Answer:**

* (B) 6.4 x 10^-34 J s

**Expert 4**

**Facts:**

* Wavelengths of incident light: 0.3 m, 0.4 m, 0.5 m
* Corresponding stopping potentials: 2.0 V, 1.0 V, 0.4 V
* Speed of light: c = 3 x 10^8 m/s
* Charge of an electron: e = 1.6 x 10^-19 C

**Alternative Answers:**

* (B) 6.4 x 10^-34 J s
* (C) 6.6 x 10^-34 J s
* (D) 6.8 x 10^-34 J s

**Likelihood of Answers:**

* (B) Consistent with the given data and within the range of accepted values.
* (C) Unlikely because it is slightly larger than expected.
* (D) Unlikely because it is too large.

**Single Answer:**

* (B) 6.4 x 10^-34 J s
```

Reasoning via Planning (RAP) Technique (Implementing Monte Carlo Tree Search in LLM)

The RAP technique is developed to address the limitations of Large Language Models (LLMs) in performing tasks that require long-term planning, such as generating action plans or performing complex logical reasoning. The deficiency lies in the absence of an internal world model within LLMs, which prevents them from simulating future states and outcomes of actions.

Framework Overview: RAP repurposes LLMs both as a world model and a reasoning agent. This means that the

LLM is used to simulate the environment and predict future states (world model), while also performing reasoning and decision-making based on these predictions (reasoning agent).

Planning Algorithm: RAP incorporates a principled planning algorithm based on Monte Carlo Tree Search (MCTS). MCTS is a heuristic search algorithm that simulates a tree of possible future states, iteratively selecting actions to explore the most promising branches of the tree.

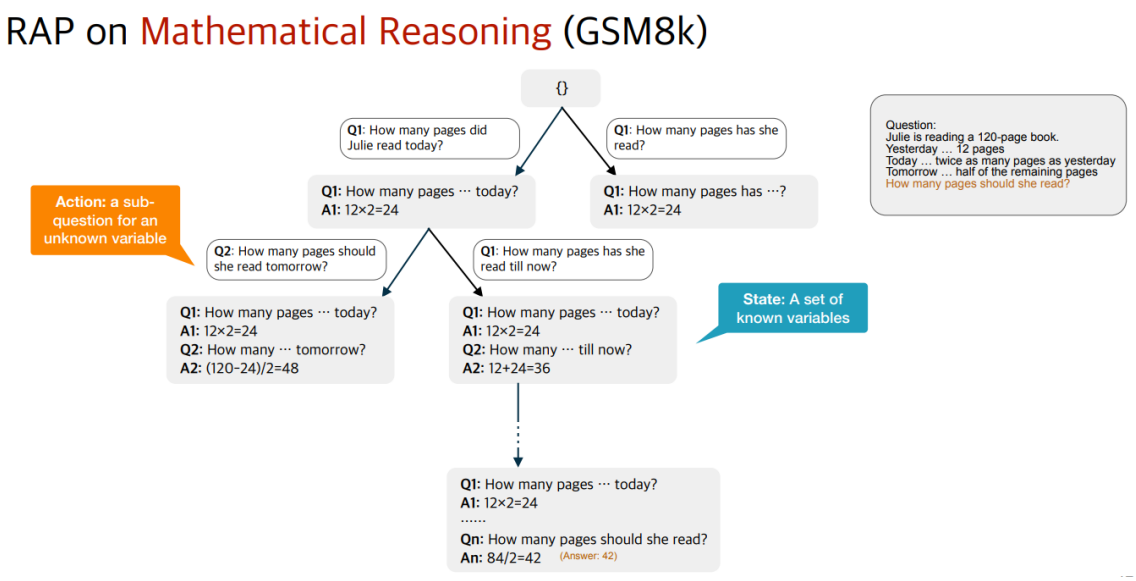
Incremental Reasoning Tree Construction: During reasoning, the LLM acts as the reasoning agent and incrementally builds a reasoning tree. This tree represents different possible reasoning paths and their associated outcomes. The construction of the reasoning tree is guided by the predictions of the LLM acting as the world model.

Balancing Exploration and Exploitation: RAP ensures a proper balance between exploration and exploitation during reasoning. Exploration involves searching for new reasoning paths and considering alternative actions, while exploitation involves selecting actions that are expected to lead to high rewards. This balance is crucial for efficient reasoning and decision-making.

Integration with Reward Signals: The LLM, acting as the reasoning agent, uses rewards obtained from the environment (or task-specific criteria) to guide the construction of the reasoning tree. High-reward reasoning paths are favored, and the algorithm aims to find the most rewarding path within the reasoning space.

Application to Challenging Reasoning Problems: RAP is applied to various challenging reasoning problems, including plan generation, math reasoning, and logical inference. By repurposing LLMs as both world models and reasoning agents and integrating MCTS-based planning, RAP demonstrates superiority over strong baselines in these tasks.

Experimental Validation: The effectiveness of RAP is validated through experiments comparing its performance with existing approaches and strong baselines. For example, in a plan generation setting, RAP with a specific LLM model (LLaMA-33B) achieves a 33 % relative improvement over the Chain-of-Thought (CoT) approach with GPT-4, highlighting its superiority in strategic exploration and reasoning efficiency.



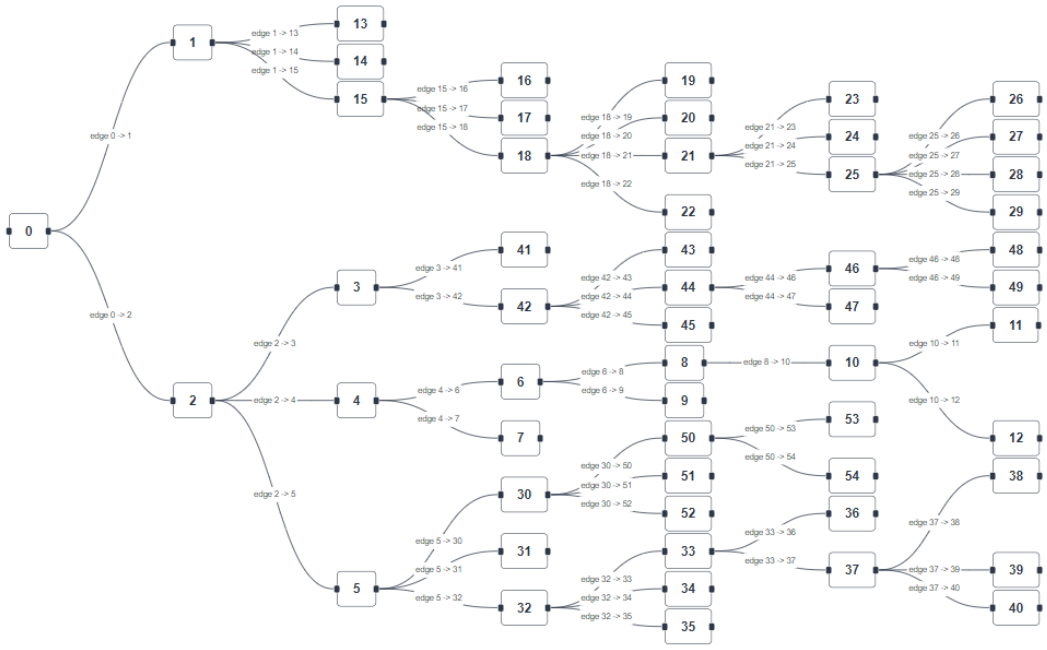
Limitations of LLMs on reasoning and solving quantitative problems

1. **Tokenization Issues:**

- Decimals: LLMs operate on discrete units of text called tokens. Decimals, when encountered, are treated like any other word or symbol. This disrupts the understanding of numerical value and order. For instance, the LLM might see "3.14" as three separate tokens - "3", ".", "14" - making it difficult to perform calculations or comparisons.

2. **Lack of Symbolic Reasoning:**

- LLMs struggle with the core concepts of mathematical operations like addition, subtraction, etc. They can identify these symbols in text but lack the inherent understanding of their function.



How the algorithm traverses the complete knowledge space

3. Limited Generalization:

- LLMs are trained on massive datasets of text and code. While they can perform calculations they've seen before, they struggle with novel problems or variations. For example, an LLM that can add single-digit numbers might fail when presented with a problem like "123 + 456."

4. Reversal Curse:

- LLMs trained on factual relationships (A is B) might not understand the converse (B is A). This is called the reversal curse. Imagine training an LLM that "Paris is the capital of France." It might struggle to answer "What country is Paris the capital of?"

5. No Real-World Understanding:

- LLMs lack the ability to connect symbols and concepts to real-world phenomena. This makes it difficult for them to grasp the meaning and context of quantitative problems.

Beyond these limitations:

Confabulations:

- LLMs are trained on massive amounts of text data, and this data may not always be accurate or truthful. When reasoning about quantitative problems, LLMs can pull from this data to create seemingly logical answers that are actually incorrect. This is akin to making up information to fill in gaps in their knowledge.

Here's how confabulations can manifest in the limitations mentioned earlier:

- **Tokenization Issues:** An LLM might encounter a nonsensical number during tokenization, like "123apple456" from poorly curated data. It could then confabulate a seemingly valid answer based on its understanding of the separate tokens ("123" and "456").
- **Limited Generalization:** When faced with a novel problem, the LLM might extrapolate from its training data and confabulate an answer that appears to fit the pattern, even if it's mathematically incorrect.
- **Reversal Curse:** Due to confabulations in the training data, the LLM might invent a relationship between concepts that doesn't actually exist.

Additional Points:

- LLMs can't indicate confidence in their answers. This makes it risky to rely on them for critical quantitative tasks.
- They struggle with problems requiring multi-step reasoning or logical deduction.

- Researchers are actively working on improving LLMs' reasoning and quantitative abilities. However, for now, it's important to be aware of these limitations and use LLMs cautiously in these domains.

Conclusions

Throughout this mini project journey, an extensive array of literature, comprising articles, research papers, and journals, was meticulously explored. Diverse techniques, such as CoT and ToT, were meticulously scrutinized and experimented with to bolster the reasoning process. Concurrently, efforts were devoted to refining the calculation aspect through the adept integration of coding methodologies like PAL, aimed at enhancing computational efficiency.

Moreover, the exploration extended to the realm of idea and thought generation, where innovative approaches, including the utilization of a Monte Carlo tree search-based algorithm, were implemented. This algorithm facilitated the sampling of prompts, thereby fostering a robust ideation process while concurrently discerning and discarding less fruitful notions. Nevertheless, the persistent challenge of formulating and operationalizing an optimal reward function loomed large, underscoring the complexity inherent in delineating parameters for assessing the efficacy of generated outputs.

Furthermore, amidst the project's exploration, an intriguing query emerged concerning the optimal utilization of large language models (LLMs) for conducting advanced calculations, such as integration and differentiation, with maximal efficiency. This inquiry delved into the intricate dynamics of leveraging LLMs to perform complex mathematical operations, presenting a multifaceted challenge that beckoned further investigation and refinement.

In culmination, this project journey encapsulated a multifaceted exploration of various techniques and methodologies aimed at enhancing reasoning, calculation, and ideation processes. While significant strides were made, the quest for optimizing reward functions and leveraging LLMs for advanced mathematical computations persists, beckoning forth further inquiry and innovation.

Discussion

Conducting experiments proved challenging as we had to rely solely on API calls, and loading large language models onto personal systems was nearly impossible, although training and fine-tuning smaller language models, such as Phi-2, was somewhat feasible. The JEE question data was in LaTeX format, necessitating handling accordingly. Due to limitations on API calls, only 10-15 questions could be experimented with at a time. The Wolfram Alpha API was somewhat underpowered and did not perform as expected. The code for RAP is merged with Tree of thoughts as complete implementation for Physics problems still require a review of what type of system physics problem form Dimensionality check was successful most of the times. Limiting cases criteria helped in checking and verifying the governing equations but for back substitution it always says correct even it may not be correct. It looks that it doesn't contradict itself on its own unless explicitly poked.

- Conducting experiments proved challenging as we had to rely solely on API calls, and loading large language models onto personal systems was nearly impossible, although training and fine-tuning smaller language models, such as Phi-2, was somewhat feasible.
- The JEE question data was in LaTeX format, necessitating handling accordingly.
- Due to limitations on API calls, only 10-15 questions could be experimented with at a time.
- The Wolfram Alpha API was somewhat underpowered and did not perform as expected.

Appendix A

JEEBench Dataset

[7] **Data Source**

- Curated from the highly competitive Indian Institute of Technology (IIT) JEE-Advanced exam, known for its difficulty.
- Contains 515 challenging problems across three STEM subjects: Physics, Chemistry, and Mathematics.
- Covers problems from the years 2016 to 2023 of the JEE Advanced exam.

Problem Types

- Focuses on problems requiring multi-step reasoning and deep subject knowledge.
- Goes beyond simple calculations and tests the Large Language Model's (LLM) ability to apply concepts and solve complex problems.

Brief Overview of JEE Advanced Exam

The Joint Entrance Examination (JEE) Advanced is an annual engineering entrance exam in India. It is conducted by one of the seven Indian Institutes of Technology (IITs) on behalf of the Joint Admission Board (JAB). Here are some key points about the exam:

- JEE Advanced is considered one of the toughest engineering entrance exams globally.
- Only students who qualify the JEE Main exam are eligible to appear for JEE Advanced.
- The exam is known for its rigorous and challenging questions that test candidates' problem-solving abilities and deep understanding of fundamental concepts in physics, chemistry, and mathematics.
- JEE Advanced serves as the gateway for admission to the prestigious Indian Institutes of Technology (IITs) and other top engineering colleges in India.

GSM8K Dataset

[3] **GSM8K Dataset Overview** The GSM8K dataset offers a rich resource for researchers working on machine learning models that tackle multi-step mathematical reasoning. Here's a more detailed breakdown of its characteristics: **Data Composition:**

- **Text Format:** Each problem in GSM8K is presented as a textual description in English This format mimics real-world word problems encountered by students.
- **Content Structure:** The problems focus on various mathematical concepts applicable to grade school math, likely ranging from elementary arithmetic to basic algebra.
- **Linguistic Diversity:** The creators aimed to incorporate a variety of sentence structures and phrasing within the problems to test the model's ability to handle different linguistic styles related to math problems .

Problem Characteristics:

- **Step-by-Step Approach:** The problems are designed to require a sequence of steps (between 2 and 8) to reach the solution . This challenges the model to follow a logical path and perform calculations iteratively.
- **Focus on Basic Operations:** The primary mathematical operations involved are addition, subtraction, multiplication, and division . While the concepts might be fundamental, solving them in a multi-step fashion requires reasoning and planning abilities in the model.
- **Answer Type:** It's likely the answer to each problem is a numerical value considering the focus on basic arithmetic operations .

Dataset Split:

- **Training Set (7.5K problems):** This larger portion of the dataset allows the machine learning model to learn the patterns and approaches required to solve these multi-step word problems .
- **Testing Set (1K problems):** This unseen set of problems evaluates the model's generalizability and its ability to apply the learned concepts to new problems .

Applications and Benefits:

- **Benchmarking Tool:** GSM8K serves as a benchmark for evaluating the capability of machine learning models in multi-step mathematical reasoning, a domain where many models struggle despite excelling in basic calculations [**?**].
- **AI Research:** Researchers can use this dataset to train and test models specifically designed to tackle word problems, potentially leading to advancements in educational AI or automated tutoring systems.

- **NLP and Math Integration:** The dataset bridges the gap between Natural Language Processing (NLP) and mathematical problem-solving, encouraging research in models that can understand and solve problems presented in natural language.

Overall, the GSM8K dataset provides a valuable resource for researchers pushing the boundaries of machine learning models in handling multi-step mathematical reasoning presented in a textual format. It allows for the evaluation and development of models that can not only perform basic calculations but also follow logical steps and solve problems mimicking real-world scenarios.

Appendix B

Combined code for Few shot, Chain of thought and Self consistency

```python
import os
from tqdm import tqdm
import json
import os
import openai
from tqdm import tqdm
import argparse
import multiprocessing
from copy import deepcopy
from functools import partial

prompt_library = {
  "MCQ": "In this problem, only one option will be correct. Give a detailed solution and end the solution with the final answer.",
  "MCQ(multiple)": "In this problem, multiple options can be correct. Give a detailed solution and end the solution with the final answer.",
  "Integer": "In this problem, the final answer will be a non-negative integer. Give a detailed solution and end the solution with the final answer.",
  "Numeric": "In this problem, the final will be a numeric value. Give the numerical answer correct upto the 2nd decimal digit. Give a detailed solution and end the solution with the final answer.",
}

few_shot_examples = json.load(open('data/few_shot_examples.json'))


def write_in_file(response_file, response_dict, question, mode, model_nickname):
  if os.path.exists(response_file):
    with open(response_file, 'r') as infile:
      responses = json.load(infile)
  else:
    responses = []

  found = False
  for i, old_resp in enumerate(responses):
    if old_resp['description'] == question['description'] and old_resp['index'] == question['index']:
      responses[i][f"{model_nickname}_{mode}_response" ] = response_dict[f"{model_nickname}_{mode}_response"]
      found = True
      break

  if not found:
    responses.append(response_dict)

  json.dump(sorted(responses, key=lambda elem: (elem['description'], elem['index'])), open(response_file, 'w'), indent=4)
  print(f"####UPDATED {response_file}, Current size : {len(responses)}####")


def get_response(question,model, model_nickname, mode, response_file, lock):

  response_dict = deepcopy(question)
  prefix_prompt = prompt_library[question['type']]
  suffix_prompt = ""

```

```
if mode in ['CoT', 'CoT+SC', 'CoT+Exam'] :
  suffix_prompt = "Let's think step by step.\n"

ques = question["question"]
stripped_ques = ques.replace("\n\n", "\n").strip()
if mode in ['CoT+OneShot', 'CoT', 'CoT+SC', 'CoT+Exam']:
  if mode == 'CoT+Exam':
    if response_dict['type'] in ['MCQ', 'MCQ(multiple)']:
      if response_dict['type'] == 'MCQ':
        exam_prompt = "If the answer is wrong, you'll be given -1 marks. If the answer is correct, you'll be
                                                    given +3 marks. If you're unsure of the answer, you can
                                                     skip the question, and you'll be given 0 marks."
      else:
        exam_prompt = "If any of the options in the final answer is wrong, you'll be given -2 marks. If all
                                                    the options are correct, you'll be given +4 marks. If
                                                    some of the options are correct, you'll be given +1 for
                                                     each correct option. If you're unsure of the answer,
                                                    you can skip the question, and you'll be given 0 marks.
                                                    "
      prompt = prefix_prompt + " " + exam_prompt + "\n\n" + "Problem: " + stripped_ques + "\nSolution: " +
                                                    suffix_prompt
    else:
      print("No point doing this for Numeric/Integer questions since there is no negative marking...")
      breakpoint()
  else:
    if mode == 'CoT+OneShot':
      ex = few_shot_examples[question['subject']][question['type']]
      prompt = prefix_prompt + "\n\n" + "Problem: " + ex['problem'] + "\nSolution: " + ex['solution'] + "\n\n
                                                    " + "Problem: " + stripped_ques + "\nSolution: "
    else:
      prompt = prefix_prompt + "\n\n" + "Problem: " + stripped_ques + "\nSolution: " + suffix_prompt
else:
  prompt = prefix_prompt + "\n\n" + "Problem: " + stripped_ques + suffix_prompt
prompt = prompt.strip()
response_dict[f"prompt"] = prompt
num_retries = 0
print(f'Question: {question["description"]}, Index: {question["index"]}, Model: {model_nickname}, Mode: {mode
                                                    }, query begins')

while True:
  try:
    if model in ["text-davinci-003", "text-davinci-002", 'davinci-002']:
      response = openai.Completion.create(
        model=model,
        prompt=prompt,
        max_tokens=2048,
        temperature=0 if mode in ['CoT', 'normal', 'CoT+Exam'] else 0.5,
        n=1 if mode in ['CoT', 'normal', 'CoT+Exam'] else 3
      )
    else:
      response = openai.ChatCompletion.create(
        model=model,
        messages=[
          {"role": "system", "content": ""},
          {"role": "user", "content": prompt}
        ],
        max_tokens=2048,
        temperature=0 if mode in ['CoT+OneShot', 'CoT', 'normal', 'CoT+Exam'] else 0.5,
        n=1 if mode in ['CoT+OneShot', 'CoT', 'normal', 'CoT+Exam'] else 8
      )

    lock.acquire()
    response_dict[f"{model_nickname}_{mode}_response"] = response
    write_in_file(response_file, response_dict, question, mode, model_nickname)
    lock.release()
    break

  except Exception as e:
    num_retries += 1
```

```
    print("Failure!", e)
  return

def main():
  '''
  The code can restart from the already done questions in case there is a failure midpoint.
  '''
  args = argparse.ArgumentParser()
  args.add_argument('--model', default='gpt-3.5-turbo')
  args.add_argument('--data', default='data/dataset.json')
  args.add_argument('--mode', default='normal')
  args.add_argument('--num_procs', default=1, type=int)
  args.add_argument('--max_questions', default=1, type=int)
  args = args.parse_args()

  openai.organization = os.getenv("OPENAI_ORG")
  openai.api_key = os.getenv("OPENAI_API_KEY")

  model_nickname = {
    "davinci-002": "davinci-002",
    "text-davinci-003": "GPT3",
    "gpt-3.5-turbo": "GPT3.5",
    "gpt-4-0613": "GPT4_0613",
    "gpt-4-0314": "GPT4"
  }
  assert args.model in model_nickname.keys()
  assert args.mode in ['normal', 'CoT', 'CoT+OneShot', 'CoT+Exam', 'CoT+SC']

  out_file_dir = f'responses/{model_nickname[args.model]}_{args.mode}_responses'
  out_file = os.path.join(out_file_dir, 'responses.json')
  questions = json.load(open(args.data))

  rem_ques = []

  if os.path.exists(out_file):

    for question in tqdm(questions[:args.max_questions]):
      if os.path.exists(out_file):
        with open(out_file, 'r') as infile:
          responses = json.load(infile)
          found = False

          for i, old_resp in enumerate(responses):
            if question['type'] in ['Numeric', 'Integer'] and args.mode == 'CoT+Exam':
              found = True
            if old_resp['description'] == question['description'] and old_resp['index'] == question['index']:

              found = all([old_resp.get(
                f"{model_nickname[args.model]}_{args.mode}_response", False) for model in [args.model]])
          if found:
            print("This question has already been done")
          else:
            rem_ques.append(question)
  else:
    os.makedirs(out_file_dir, exist_ok=True)
    if args.mode == 'CoT+Exam':
      rem_ques = []
      for q in questions:
        if q['type'] in ['MCQ', 'MCQ(multiple)']:
          rem_ques.append(q)
    else:
      rem_ques = questions[:args.max_questions]
  print(f"There are {len(rem_ques)} problems remaining")

  manager = multiprocessing.Manager()
  lock = manager.Lock()
  pool = multiprocessing.Pool(args.num_procs)
  f = partial(get_response, model=args.model, model_nickname=model_nickname[args.model], mode=args.mode,
                                                          response_file=out_file, lock=lock)
```

```
  pool.map(f, rem_ques)

if __name__ == '__main__':
  main()
```

Prompt Example for few shot:- Physics

Multiple Choice Questions (MCQ)

Problem

The diameter of a cylinder is measured using a vernier callipers with no zero error. It is found that the zero of the vernier scale lies between 5.10 cm and 5.15 cm of the main scale. The vernier scale has 50 division equivalent to 2.45 cm. The 24^{th} division of the vernier scale exactly coincides with one of the main scale divisions. The diameter of the cylinder is:

1. (A) 5.112 cm
2. (B) 5.124 cm
3. (C) 5.136 cm
4. (D) 5.148 cm

Solution

$$
\begin{aligned}
&50\,\text{VSD} = 2.45\,\text{cm}\\
&1\,\text{VSD} = \frac{2.45}{50}\,\text{cm} = 0.049\,\text{cm}\\
&\text{Least count of vernier} = 1\,\text{MSD} - 1\,\text{VSD} = 0.05\,\text{cm} - 0.049\,\text{cm} = 0.001\,\text{cm}\\
&\text{Thickness of the object} = \text{Main scale reading} + \text{vernier scale reading} \times \text{least count}\\
&= 5.10 + (24)(0.001) = 5.124\,\text{cm}.
\end{aligned}
$$

Therefore, the answer is $\boxed{\text{B}}$.

Multiple Choice Questions (Multiple)

Problem

A horizontal stretched string, fixed at two ends, is vibrating in its fifth harmonic according to the equation,

$$y(x,t) = (0.01\,\text{m}) \sin\left[\left(62.8\,\text{m}^{-1}\right)x\right] \cos\left[\left(628\,\text{s}^{-1}\right)t\right].$$

Assuming $\pi = 3.14$, the correct statement(s) is (are):

1. (A) The number of nodes is 5.
2. (B) The length of the string is 0.25 m.
3. (C) The maximum displacement of the midpoint of the string its equilibrium position is 0.01 m.
4. (D) The fundamental frequency is 100 Hz.

Solution

1. (A) There are 5 complete loops. Therefore, the total number of nodes is 6.
2. (B) $\omega = 628\,\text{sec}^{-1}$. Therefore,

$$\text{k} = 62.8\,\text{m}^{-1} = \frac{2\pi}{\lambda} \Rightarrow \lambda = \frac{1}{10}$$

$$\text{v}_\text{w} = \frac{\omega}{\text{k}} = \frac{628}{62.8} = 10\,\text{ms}^{-1}$$

$$L = \frac{5\lambda}{2} = 0.25\,\text{m}.$$

3. (C) Maximum amplitude of antinode $= 2\,\text{A} = 0.01$.

4. (D) $f = \frac{v}{2\ell} = \frac{10}{2\times 0.25} = 20\,\text{Hz}$.

Therefore, the answer is $\boxed{\text{BC}}$.

Integer Type Questions

Problem

The work functions of Silver and Sodium are 4.6 and 2.3 eV, respectively. The ratio of the slope of the stopping potential versus frequency plot for Silver to that of Sodium is:

Solution

We know that $KE_{\max} = hv - \phi$. Therefore, $eV_{\text{st}} = hv - \phi$. That is, $V_{\text{st}} = \left(\frac{h}{e}\right) v - \frac{\phi}{e}$

So slope will be $\left(\frac{h}{e}\right)$, and it will be same for both the metals. So ratio of the slopes is 1.

Therefore, the answer is $\boxed{1}$.

Numeric Type Questions

Problem

The work functions of Silver and Sodium are 4.6 and 2.3 eV, respectively. Let the slope of the stopping potential versus frequency plot for Silver be m_1 and for that of Sodium be m_2. Then what is the value of $\frac{m_1}{4m_2}$?

Solution

We know that $KE_{\max} = hv - \phi$. Therefore, $eV_{\text{st}} = hv - \phi$. That is, $V_{\text{st}} = \left(\frac{h}{e}\right) v - \frac{\phi}{e}$

So slope will be $\left(\frac{h}{e}\right)$, and it will be same for both the metals. Therefore, $m_1 = m_2$. This implies that $\frac{m_1}{4m_2} = \frac{1}{4} \approx 0.25$

Therefore, the answer is $\boxed{0.25}$.

Chemistry

Multiple Choice Questions (MCQ)

Problem

Concentrated nitric acid, upon long standing, turns yellow-brown due to the formation of

1. (A) NO
2. (B) NO_2
3. (C) N_2O
4. (D) N_2O_4

Solution

HNO_3 decomposes by giving NO_2, O_2, H_2O

$$4\ HNO_3 \longrightarrow 2\ H_2O + 2\ NO_2 + 3\ O_2$$

Therefore, the answer is $\boxed{\text{B}}$.

Multiple Choice Questions (Multiple)

Problem

Benzene and naphthalene form an ideal solution at room temperature. For this process, the true statement(s) is (are)

1. (A) ΔG is positive
2. (B) ΔS_{system} is positive
3. (C) $\Delta S_{\text{surroundings}} = 0$

4. (D) $\Delta H = 0$

Solution

For a solution formation, we know that ΔG is always -ve and ΔS_{system} is always +ve. Since there is no heat exchange between the solution and the surrounding $\Delta S_{\text{surr.}} = 0$. Since it is an ideal solution, $\Delta H = 0$.

Therefore, the answer is $\boxed{\text{BCD}}$.

Integer Type Questions

Problem

The atomic masses of He and Ne are 4 and 20 a.m.u., respectively. The value of the de Broglie wavelength of He gas at $-73°$C is "M" times that of the de Broglie wavelength of Ne at 727°C. What is M?

Solution

$$\lambda = \frac{h}{\sqrt{2 \text{ m(KE)}}} \quad \text{KE} \propto \text{T}$$

$$\frac{\lambda_{He}}{\lambda_{Ne}} = \sqrt{\frac{m_{Ne} \text{ KE}_{Ne}}{m_{He} \text{ KE}E_{He}}} = \sqrt{\frac{20 \times 1000}{4 \times 200}} = 5$$

Therefore, the answer is $\boxed{5}$.

Numeric Type Questions

Problem

The atomic masses of He and Ne are 4 and 20 a.m.u., respectively. The value of the de Broglie wavelength of He gas at 2227°C is "M" times that of the de Broglie wavelength of Ne at 727°C. What is M?

Solution

$$2227°\text{C} = 2500K$$

$$727°\text{C} = 1000K$$

We know that $\lambda = \frac{h}{\sqrt{2 \text{ m(KE)}}} \quad \text{KE} \propto \text{T}$

$$\frac{\lambda_{He}}{\lambda_{Ne}} = \sqrt{\frac{m_{Ne} \text{ KE}_{Ne}}{m_{He} \text{ KE}E_{He}}} = \sqrt{\frac{20 \times 1000}{4 \times 2500}} = \sqrt{2} = 1.41$$

Therefore, the answer is $\boxed{1.41}$.

Mathematics

Multiple Choice Questions (MCQ)

Problem

Perpendiculars are drawn from points on the line $\frac{x+2}{2} = \frac{y+1}{-1} = \frac{z}{3}$ to the plane $x + y + z = 3$. The feet of perpendiculars lie on the line

1. (A) $\frac{x}{5} = \frac{y-1}{8} = \frac{z-2}{-13}$
2. (B) $\frac{x}{2} = \frac{y-1}{3} = \frac{z-2}{-5}$
3. (C) $\frac{x}{4} = \frac{y-1}{3} = \frac{z-2}{-7}$
4. (D) $\frac{x}{2} = \frac{y-1}{-7} = \frac{z-2}{5}$

Solution

Any point on line can be represented by λ s.t. $\frac{x+2}{2} = \frac{y+1}{-1} = \frac{z}{3} = \lambda$ Putting $\lambda = 0, 1$, we get two points A(-2,-1,0) and B(0,-2,3) . Let the foot of perpendicular from A on plane be (α, β, γ)

$$\Rightarrow \frac{\alpha+2}{1} = \frac{\beta+1}{1} = \frac{\beta-0}{1} = \mu \quad \text{(say)}$$

Also, $\alpha + \beta + \gamma = 3$. Therefore,

$$\Rightarrow \mu - 2 + \mu - 1 + \mu = 3 \Rightarrow \mu = 2$$
$$\Rightarrow M(0, 1, 2)$$

Similarly foot of perpendicular from $B(0, -2, 3)$ on plane is $N\left(\frac{2}{3}, \frac{-4}{3}, \frac{11}{3}\right)$ So, equation of MN is $\frac{x-0}{\frac{2}{3}} = \frac{y-1}{\frac{-7}{3}} = \frac{z-2}{\frac{5}{3}}$.

Therefore, the answer is $\boxed{D}$.

Multiple Choice Questions (Multiple)

Problem

For 3×3 matrices M and N, which of the following statement(s) is (are) NOT correct ?

1. (A) $N^T MN$ is symmetric or skew symmetric, according as M is symmetric or skew symmetric
2. (B) $MN - NM$ is skew symmetric for all symmetric matrices M and N
3. (C) MN is symetric for all symmetric matrices M and N
4. (D) $(\text{adj}M)(\text{adj}N) = \text{adj}(MN)$ for all invertible matrices M and N

Solution

1. (A) $(N^T MN)^T = N^T M^T N$. Therefore, it is symmetric if M is symmetric and skew-symmetric if M is skew-symmetric.
2. (B) $(MN - NM)^T = (MN)^T - (NM)^T = NM - MN = -(MN - NM)$. Therefore, it is skew symmetric.
3. (C) $(MN)^T = N^T M^T = NM \neq MN$ hence NOT correct.
4. (D) $\text{adj}(MN) = (\text{adj}N)(\text{adj}M) \neq (\text{adj}M)(\text{adj}N)$ hence NOT correct.

Therefore, the answer is $\boxed{CD}$.

Integer Type Questions

Problem

Consider the set of eight vectors $V = \{a\hat{i} + b\hat{j} + c\hat{k} : a, b, c \in \{-1, 1\}\}$. Three non-coplanar vectors can be chosen from V in 2^p ways. Then p is

Solution

Among set of eight vectors four vectors form body diagonals of a cube, remaining four will be parallel (unlike) vectors. Numbers of ways of selecting three vectors will be $\binom{4}{3} \times 2 \times 2 \times 2 = 2^5$ Hence $p = 5$.

Therefore, the answer is $\boxed{5}$.

Numeric Type Questions

Problem

Consider the set of eight vectors $V = \{a\hat{i} + b\hat{j} + c\hat{k} : a, b, c \in \{-1, 1\}\}$. Three non-coplanar vectors can be chosen from V in 2^p ways. Then what is $\frac{p}{2}$?

Solution

Among set of eight vectors four vectors form body diagonals of a cube, remaining four will be parallel (unlike) vectors. Numbers of ways of selecting three vectors will be $\binom{4}{3} \times 2 \times 2 \times 2 = 2^5$

Hence $p = 5$ Therefore, the answer is $\boxed{2.50}$.

Code for Step back prompting

```
# -*- coding: utf-8 -*-
"""Untitled23.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1wA_O2XRb_His1PJOMaBl05S9UyeLAbLB
"""

!pip install --upgrade --quiet  langchain-google-genai pillow

import getpass
import os

if "GOOGLE_API_KEY" not in os.environ:
    os.environ["GOOGLE_API_KEY"] = getpass.getpass("Provide your Google API Key")

from langchain_google_genai import ChatGoogleGenerativeAI

llm = ChatGoogleGenerativeAI(model="gemini-pro", convert_system_message_to_human=True)

from langchain_core.output_parsers import StrOutputParser
from langchain_core.prompts import ChatPromptTemplate, FewShotChatMessagePromptTemplate
from langchain_core.runnables import RunnableLambda

examples = [
    {
        "input": "Could the members of The Police perform lawful arrests?",
        "output": "what can the members of The Police do?",
    },
    {
        "input": "Jan Sindels was born in what country?",
        "output": "what is Jan Sindels personal history?",
    },
]
# We now transform these to example messages
example_prompt = ChatPromptTemplate.from_messages(
    [
        ("human", "{input}"),
        ("ai", "{output}"),
    ]
)
few_shot_prompt = FewShotChatMessagePromptTemplate(
    example_prompt=example_prompt,
    examples=examples,
)

prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """You are an expert at world knowledge. Your task is to step back and paraphrase a question to a
                                                    more generic step-back question, which is easier to
                                                    answer. Here are a few examples:""",
        ),
        # Few shot examples
        few_shot_prompt,
        # New question
        ("user", "{question}"),
    ]
)

question_gen = prompt | llm | StrOutputParser()

question = """A particle of mass m is constrained to move along a circle of radius R on a frictionless
                                                    horizontal plane. A string is wound on the particle,
                                                    and the other end of the string is held fixed at the
```

```
                                                    center of the circle. The particle is set into circular
                                                     motion with angular velocity . Subsequently, the
                                                    angular velocity is increased at a constant rate .

Task:
Calculate the tension in the string when the angular velocity becomes .

Options:
1. T = m(R + R)
2. T = m(R + R)
3. T = mR
4. T = m(R - R)
"""

q=question_gen.invoke({"question": question})
print(q)

!pip install langchain_community
!pip install -U duckduckgo-search
from langchain_community.utilities import DuckDuckGoSearchAPIWrapper

search = DuckDuckGoSearchAPIWrapper(max_results=4)


def retriever(query):
    return search.run(query)

retriever(q)

!pip install langchain
!pip install langchainhub
from langchain import hub

response_prompt = hub.pull("langchain-ai/stepback-answer")

chain = (
    {
        # Retrieve context using the normal question
        "normal_context": RunnableLambda(lambda x: x["question"]) ,
        # Retrieve context using the step-back question
        "step_back_context": question_gen | retriever,
        # Pass on the question
        "question": lambda x: x["question"],
    }
    | response_prompt
    | llm
    | StrOutputParser()
)
import pandas as pd
df=pd.read_csv("/content/data.csv")

df.head()

prompt_library = {
  "MCQ": "In this problem, only one option will be correct. Give a detailed solution and end the solution with
                                                    the final answer.",
  "MCQ(multiple)": "In this problem, multiple options can be correct. Give a detailed solution and end the
                                                    solution with the final answer.",
  "Integer": "In this problem, the final answer will be a non-negative integer. Give a detailed solution and
                                                    end the solution with the final answer.",
  "Numeric": "In this problem, the final will be a numeric value. Give the numerical answer correct upto the
                                                    2nd decimal digit. Give a detailed solution and end the
                                                     solution with the final answer.",
}

for i in range(5):
  print(df['gold'][i])

df['question'][i]
```

```
responses=[]
for i in range(5):

    prefix_prompt = prompt_library[df['type'][i]]
    suffix_prompt = ""
    stripped_ques = df['question'][i].replace("\n\n", "\n").strip()

    if df['type'][i] in ['MCQ', 'MCQ(multiple)']:
        if df['type'][i] == 'MCQ':
            exam_prompt = "If the answer is wrong, you'll be given -1 marks. If the answer is correct, you'll
                                                    be given +3 marks. If you're unsure of the answer, you
                                                    can skip the question, and you'll be given 0 marks."
        else:
            exam_prompt = "If any of the options in the final answer is wrong, you'll be given -2 marks. If all
                                                     the options are correct, you'll be given +4 marks. If
                                                    some of the options are correct, you'll be given +1 for
                                                     each correct option. If you're unsure of the answer,
                                                    you can skip the question, and you'll be given 0 marks.
                                                    "

        prompt = prefix_prompt + " " + exam_prompt + "\n\n" + "Problem: " + stripped_ques + "\nSolution: " +
                                                    suffix_prompt
    else:
        prompt= prefix_prompt+ " "+stripped_ques
    response=chain.invoke({"question": prompt})
    print(question_gen.invoke({"question": prompt}))
    print("xxxxxxxx")
    print(response)
    print("------")
    responses.append(response)
```

Code for LLM-SLM ensemble

```
from datasets import load_dataset
from transformers import LLaMATokenizer
import sys
import os
import re
os.environ['WANDB_DISABLED'] = 'true'
import os

os.environ["CUDA_VISIBLE_DEVICES"] = "0"
import torch
import torch.nn as nn
import bitsandbytes as bnb
from datasets import load_dataset
import transformers
from transformers import AutoTokenizer, AutoConfig, LLaMAForCausalLM, LLaMATokenizer
from peft import prepare_model_for_int8_training, LoraConfig, get_peft_model

# Setting for A100 - For 3090
MICRO_BATCH_SIZE = 4  # change to 4 for 3090
BATCH_SIZE = 128
GRADIENT_ACCUMULATION_STEPS = BATCH_SIZE // MICRO_BATCH_SIZE
EPOCHS = 8  # paper uses 3
LEARNING_RATE = 2e-5  # from the original paper
CUTOFF_LEN = 256  # 256 accounts for about 96% of the data
LORA_R = 4
LORA_ALPHA = 16
LORA_DROPOUT = 0.05

tokenizer = LLaMATokenizer.from_pretrained("13B_HF/tokenizer.model", add_eos_token=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.pad_token_id = tokenizer.eos_token_id

model = LLaMAForCausalLM.from_pretrained(
    "13B_HF/",
```

```
    load_in_8bit=True,
    device_map={"":0},
)

model = prepare_model_for_int8_training(model)
# sys.exit()

config = LoraConfig(
    r=LORA_R,
    lora_alpha=LORA_ALPHA,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=LORA_DROPOUT,
    bias="none",
    task_type="CAUSAL_LM",
)
model = get_peft_model(model, config)
tokenizer.pad_token_id = 0  # unk. we want this to be different from the eos token


def generate_prompt(data_point):
    if(data_point['question']):
        return f"""Below is an instruction that describes a task, paired with an input and a reasoning that
                                                provides further context. Write a response that
                                                appropriately completes the request.

    ### Instruction: Break the input question into multiple subquestions based on the reasoning provided.

    ### Input:
    {data_point["question"]}

    ### Reasoning:
    {data_point["Reasoning"]}

    ### Response:
    {data_point["sub-questions"]}"""


data = load_dataset("json", data_files="merged.json")

data = data.shuffle().map(
    lambda data_point: tokenizer(
        generate_prompt(data_point),
        truncation=True,
        max_length=CUTOFF_LEN,
        padding="max_length",
    )
)


# breakpoint()
trainer = transformers.Trainer(
    model=model,
    train_dataset=data["train"],
    args=transformers.TrainingArguments(
        per_device_train_batch_size=MICRO_BATCH_SIZE,
        gradient_accumulation_steps=GRADIENT_ACCUMULATION_STEPS,
        warmup_steps=100,
        num_train_epochs=EPOCHS,
        learning_rate=LEARNING_RATE,
        fp16=True,
        # logging_steps=1,
        output_dir="lora-alpaca-13B-context-qa",
        save_total_limit=3,
    ),
    data_collator=transformers.DataCollatorForLanguageModeling(tokenizer, mlm=False),
)
model.config.use_cache = False
trainer.train(resume_from_checkpoint=False)
```

```
model.save_pretrained("lora-alpaca-13B-context-qa")
```

Code to fine tune the SLM

Code for using wolfram alpha with LLM

```
# -*- coding: utf-8 -*-
"""LangChain <> Wolfram Alpha

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1AAyEdTz-Z6ShKvewbt1ZHUICqak0MiwR
"""

!pip install openai wolframalpha langchain==0.0.60

import os
os.environ["OPENAI_API_KEY"] = ""
os.environ["WOLFRAM_ALPHA_APPID"] = ""

from langchain.agents import load_tools, initialize_agent
from langchain.llms import OpenAI
from langchain.chains.conversation.memory import ConversationBufferMemory

llm = OpenAI(temperature=0.7)
tools = load_tools(['wolfram-alpha'])
memory = ConversationBufferMemory(memory_key="chat_history")
agent = initialize_agent(tools, llm, agent="conversational-react-description", memory=memory, verbose=True)




agent.run("In a historical experiment to determine Planck's constant, a metal surface was irradiated with light
                                                    of different wavelengths. The emitted photoelectron
                                                    energies were measured by applying a stopping potential
                                                    . The relevant data for the wavelength $(\lambda)$ of
                                                    incident light and the corresponding stopping potential
                                                     $\left(V_{0}\right)$ are given below : \begin{center}
                                                    \begin{tabular}{cc} \hline $\lambda(\mu \mathrm{m})$ &
                                                    $V_{0}($ Volt $)$ \\ \hline 0.3 & 2.0 \\ 0.4 & 1.0 \\ 0
                                                    .5 & 0.4 \\ \hline \end{tabular} \end{center} Given
                                                    that $c=3 \times 10^{8} \mathrm{~m} \mathrm{~s}^{-1}$
                                                    and $e=1.6 \times 10^{-19} \mathrm{C}$, Planck's
                                                    constant (in units of $J \mathrm{~s}$ ) found from such
                                                     an experiment is (A) $6.0 \times 10^{-34}$ (B) $6.4 \
                                                    times 10^{-34}$ (C) $6.6 \times 10^{-34}$ (D) $6.8 \
                                                    times 10^{-34}$")
```

Code for PAL

```
# -*- coding: utf-8 -*-
"""PAL.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1u4_RsdI0E79PCMDdcPiJUzYhdnjoXeXc

# Setup

## Install Dependencies
"""

!git clone https://github.com/luyug/pal
!pip install ./pal
```

```
"""## Configure Openai API"""

import openai
openai.api_key = '' #@param {type:"string"}
MODEL = 'code-davinci-002' #@param {type:"string"}

import pal
from pal.prompt import colored_object_prompt, math_prompts

"""# GSM-8K"""

interface = pal.interface.ProgramInterface(model=MODEL, get_answer_expr='solution()', verbose=True)

question = "Jan has three times the number of pets as Marcia. Marcia has two more pets than Cindy. If Cindy has four pets, how many total pets do the three have?" #@param {type:"string"}
prompt = math_prompts.MATH_PROMPT.format(question=question)

answer = interface.run(prompt, time_out=10)
print('\n==========================')
print(f'The answer is {answer}.')

"""# Reasoning about Colored Objects"""

interface = pal.interface.ProgramInterface(model=MODEL, get_answer_symbol='answer', stop='\n\n\n', verbose=True)

question = "On the desk, you see two blue booklets, two purple booklets, and two yellow pairs of sunglasses. If I remove all the pairs of sunglasses from the desk, how many purple items remain on it?"#@param {type:"string"}
prompt = colored_object_prompt.COLOR_OBJECT_PROMPT.format(question=question)

answer = interface.run(prompt, time_out=10)
print('\n==========================')
print(f'The answer is {answer}.')
```

Tree of thoughts

```
# -*- coding: utf-8 -*-
"""Tree of thoughts with PAL

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1FaANzwx9FSS6M_50iSQ_hPZp22FQC-8M
"""

!pip install swarms

!pip install weaviate-client

!pip install dotenv

!pip install tree_of_thoughts

!pip install weaviate-client

!pip install cohere

import logging
from swarms import Agent

# Logging
logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)
```

```python
logger = logging.getLogger(__name__)


class ToTAgent:
    """

    OpenAI Language Model API Wrapper

    Args:
        agent (Agent): Agent class from swarms
        strategy (str): Strategy to use for generating thoughts
        evaluation_strategy (str): Strategy to use for evaluating states
        enable_react (bool): Enable ReAct prompting
        k (int): Number of thoughts to generate

    Methods:
        run(task: str) -> list: Generate text from prompt using OpenAI API
        generate_thoughts(state, k, initial_prompt, rejected_solutions=None) -> list: Generate thoughts from
                                                        state using OpenAI API
        generate_solution(initial_prompt, state, rejected_solutions=None) -> str: Generate solution from state
                                                        using OpenAI API
        evaluate_states(states, initial_prompt) -> dict: Evaluate states of reasoning using OpenAI API

    Examples:
        >>> from tree_of_thoughts.tot_agent import ToTAgent
        >>> from swarms import Agent
        >>> agent = Agent()
        >>> model = ToTAgent(agent)
        >>> thoughts = model.run("Write down your observations in format 'Observation:xxxx', then write down
                                                        your thoughts in format 'Thoughts:xxxx'.")
        >>> print(thoughts)
        ['Observation:xxxx', 'Thoughts:xxxx']

    """

    def __init__(
        self,
        agent: Agent,
        strategy: str = "cot",
        evaluation_strategy: str = "value",
        enable_react: bool = True,
        k: int = 3,
        *args,
        **kwargs,
    ):
        self.agent = agent
        self.use_chat_api = True
        self.enable_react = enable_react
        self.strategy = strategy
        self.evaluation_strategy = evaluation_strategy
        self.k = k

        # reference : https://www.promptingguide.ai/techniques/react
        self.ReAct_prompt = ""
        if enable_react:
            self.react_prompt = (
                "Write down your observations in format 'Observation:xxxx',"
                " then write down your thoughts in format 'Thoughts:xxxx'."
            )

    def run(self, task: str):
        """Generate text from prompt using"""
        if self.use_chat_api:
            thoughts = []
            for _ in range(self.k):
                response = self.agent(task)
                thoughts += [response]
            return thoughts

```

```python
    def generate_thoughts(
        self, state, k, initial_prompt, rejected_solutions=None
    ):
        """
        Generate thoughts from state using OpenAI API

        Args:
            state (str or list): State of reasoning
            k (int): Number of thoughts to generate
            initial_prompt (str): Initial prompt
            rejected_solutions (list): List of rejected solutions

        Returns:
            list: List of thoughts



        """
        if type(state) == str:
            state_text = state
        else:
            state_text = "\n".join(state)
        print("New state generating thought:", state, "\n\n")
        prompt = f"""Y
        ou're an TreeofThoughts, an superintelligent AI model devoted to helping Humans by any means necessary.
                                                        You're purpose is to generate a series of solutions to
                                                        comply with the user's instructions, you must generate
                                                        solutions on the basis of determining the most
                                                       reliable solution in the shortest amount of time, while
                                                        taking rejected solutions into account and learning
                                                       from them.
        Considering the reasoning provided:\n\n
        ###'{state_text}'\n\n###
        Devise the best possible solution for the task: {initial_prompt}, Here are evaluated solutions that
                                                       were rejected:
        ###{rejected_solutions}###,
        complete the {initial_prompt} without making the same mistakes you did with the evaluated rejected
                                                       solutions. Be simple. Be direct. Provide intuitive
                                                       solutions as soon as you think of them."""

        prompt += self.react_prompt
        thoughts = self.run(prompt)
        return thoughts

    def generate_solution(self, initial_prompt, state, rejected_solutions=None):
        try:
            if isinstance(state, list):
                state_text = "\n".join(state)
            else:
                state_text = state

            prompt = f"""You're an TreeofThoughts, an superintelligent AI model devoted to helping Humans by
                                                       any means necessary. You're purpose is to generate a
                                                       series of solutions to comply with the user's
                                                       instructions, you must generate solutions on the basis
                                                       of determining the most reliable solution in the
                                                       shortest amount of time, while taking rejected
                                                       solutions into account and learning from them.
            Considering the reasoning provided:\n\n
            ###'{state_text}'\n\n###
            Devise the best possible solution for the task: {initial_prompt}, Here are evaluated solutions that
                                                        were rejected:
            ###{rejected_solutions}###,
            complete the {initial_prompt} without making the same mistakes you did with the evaluated rejected
                                                       solutions. Be simple. Be direct. Provide intuitive
                                                       solutions as soon as you think of them."""
            answer = self.run(prompt)
            print(f"Answerrrrrr {answer}")
            # print(thoughts)
```

```
            # print(f"General Solution : {answer}")
            return answer
        except Exception as e:
            logger.error(f"Error in generate_solutions: {e}")
            return None

    def evaluate_states(self, states, initial_prompt):
        if not states:
            return {}

        if self.evaluation_strategy == "value":
            state_values = {}
            for state in states:
                if type(state) == str:
                    state_text = state
                else:
                    state_text = "\n".join(state)
                print(
                    "We receive a state of type",
                    type(state),
                    "For state: ",
                    state,
                    "\n\n",
                )
                prompt = f""" To achieve the following goal: '{initial_prompt}', pessimistically value the
                                                    context of the past solutions and more importantly the
                                                    latest generated solution you had AS A FLOAT BETWEEN 0
                                                    AND 1\n
                    Past solutions:\n\n
                    {state_text}\n
                    If the solutions is not directly concretely making fast progress in achieving the goal,
                                                    give it a lower score.
                    Evaluate all solutions AS A FLOAT BETWEEN 0 and 1:\n,  DO NOT RETURN ANYTHING ELSE
                """

                response = self.agent(prompt)
                try:
                    value_text = self.openai_choice2text_handler(
                        response.choices[0]
                    )
                    # print(f'state: {value_text}')
                    value = float(value_text)
                    print(f"Evaluated Thought Value: {value}")
                except ValueError:
                    value = 0  # Assign a default value if the conversion fails
                state_values[state] = value
            return state_values

        elif self.evaluation_strategy == "vote":
            states_text = "\n".join([" ".join(state) for state in states])
            prompt = (
                "Given the following states of reasoning, vote for the best"
                " state utilizing an scalar value"
                f" 1-10:\n{states_text}\n\nVote, on the probability of this"
                f" state of reasoning achieveing {initial_prompt} and become"
                " very pessimistic very NOTHING ELSE"
            )
            response = self.agent(prompt)
            print(f"state response: {response}")
            best_state_text = self.agent(response.choices[0])
            print(f"Best state text: {best_state_text}")
            best_state = tuple(best_state_text.split())
            print(f"best_state: {best_state}")

            return {state: 1 if state == best_state else 0 for state in states}

        else:
            raise ValueError(
                "Invalid evaluation strategy. Choose 'value' or 'vote'."
```

```
            )

import concurrent.futures
import json
import logging
import os
import time
from queue import PriorityQueue
from typing import Any, Dict, Union

import numpy as np


DATA_PATH = "./data"


logging.basicConfig(
    level=logging.INFO, format="%(asctime)s - %(levelname)s - %(message)s"
)
logger = logging.getLogger(__name__)


class TreeofThoughts:
    """
    A class representing a tree of thoughts.

    Attributes:
        model: The model used for evaluation.
        tree: The tree structure containing the nodes and their evaluations.
        best_state: The best state found so far.
        best_value: The best value found so far.
        history: The history of evaluated states.

    Methods:
        save_tree_to_json: Saves the tree structure to a JSON file.
        log_new_state: Logs a new state and its evaluation.
        adjust_pruning_threshold_precentile: Adjusts the pruning threshold based on the evaluated thoughts
                                                        using percentile.
        adjust_pruning_threshold_moving_average: Adjusts the pruning threshold based on the evaluated thoughts
                                                        using moving average.
    """

    def __init__(self, model):
        self.model = model
        self.tree: Dict[str, Dict[str, Union[float, Dict[str, Any]]]] = {
            "nodes": {},
        }
        self.best_state = None
        self.best_value = float("-inf")
        self.history = []  # added line initalize history

    def save_tree_to_json(self, file_name):
        """
        Saves the tree structure to a JSON file.

        Args:
            file_name: The name of the JSON file to save the tree structure to.
        """
        os.makedirs(os.path.dirname(file_name), exist_ok=True)
        with open(file_name, "w") as json_file:
            json.dump(self.tree, json_file, indent=4)

    def log_new_state(self, state, evaluation):
        """
        Logs a new state and its evaluation.

        Args:
            state: The state to log.
            evaluation: The evaluation of the state.
```

```
        """
        if not (type(state) == str):
            state = " | ".join(state)
        if state in self.tree["nodes"]:
            self.tree["nodes"][state]["thoughts"].append(evaluation)
        else:
            self.tree["nodes"][state] = {"thoughts": [evaluation]}

    def adjust_pruning_threshold_precentile(
        self, evaluated_thoughts, percentile
    ):
        """
        Adjusts the pruning threshold based on the evaluated thoughts using percentile.

        Args:
            evaluated_thoughts: A dictionary of evaluated thoughts.
            percentile: The percentile value to use for adjusting the threshold.

        Returns:
            The adjusted pruning threshold.
        """
        values = np.array(list(evaluated_thoughts.values()))
        if values.size == 0:
            return 0
        return max(np.percentile(values, percentile), 0.1)

    def adjust_pruning_threshold_moving_average(
        self, evaluated_thoughts, window_size
    ):
        """
        Adjusts the pruning threshold based on the evaluated thoughts using moving average.

        Args:
            evaluated_thoughts: A dictionary of evaluated thoughts.
            window_size: The size of the moving average window.

        Returns:
            The adjusted pruning threshold.
        """
        values = list(evaluated_thoughts.values())
        if len(values) < window_size:
            return np.mean(values) if values else 0
        else:
            return max(np.mean(values[-window_size:]), 0.1)


######################


class BFS(TreeofThoughts):
    """Class representing the Breadth-First Search algorithm for Tree of Thoughts."""

    def solve(
        self,
        initial_prompt,
        num_thoughts,
        max_steps,
        max_states,
        value_threshold,
        pruning_threshold=0.5,
    ):
        """
        Solve the Tree of Thoughts problem using the Breadth-First Search algorithm.

        Args:
            initial_prompt (str): The initial prompt for generating thoughts.
            num_thoughts (int): The number of thoughts to generate at each state.
            max_steps (int): The maximum number of steps to take in the search.
            max_states (int): The maximum number of states to keep track of.
```

```
        value_threshold (float): The threshold value for selecting states.
        pruning_threshold (float, optional): The threshold for dynamic pruning. Defaults to 0.5.

    Returns:
        str or None: The generated solution or the highest rated state.
    """
    current_states = [initial_prompt]
    state_values = {}
    dynamic_pruning_threshold = pruning_threshold

    try:
        with concurrent.futures.ThreadPoolExecutor() as executor:
            for step in range(1, max_steps + 1):
                selected_states = []
                for state in current_states:
                    thoughts = self.model.generate_thoughts(
                        state, num_thoughts, initial_prompt
                    )
                    futures = [
                        executor.submit(
                            self.model.evaluate_states,
                            {thought: 0},
                            initial_prompt,
                        )
                        for thought in thoughts
                    ]
                    concurrent.futures.wait(futures)
                    evaluated_thoughts = {
                        thought: fut.result()
                        for thought, fut in zip(thoughts, futures)
                        if isinstance(fut.result(), (int, float))
                    }  # check if result is a number

                    if (
                        evaluated_thoughts
                    ):  # only adjust if you have evaluated thoughts
                        dynamic_pruning_threshold = (
                            self.adjust_pruning_threshold_moving_average(
                                evaluated_thoughts, 5
                            )
                        )

                    for thought, value in evaluated_thoughts.items():
                        flattened_state = (
                            (state, thought)
                            if isinstance(state, str)
                            else (*state, thought)
                        )
                        selected_states.append((flattened_state, value))

                    selected_states.sort(key=lambda x: x[1], reverse=True)
                    selected_states = selected_states[
                        :max_states
                    ]  # Select only the top states

                    for state, value in selected_states:
                        if value >= dynamic_pruning_threshold:
                            state_values[state] = value
                            self.log_new_state(state, value)
                            logger.debug(f"State Values: {state_values}")

        if state_values:
            highest_rated_solution = max(
                state_values.items(), key=lambda x: x[1]
            )
            highest_rated_state = highest_rated_solution[0]
            solution = self.model.generate_solution(
                initial_prompt, highest_rated_state
            )
```

```
                print(
                    "Highest_rated solution:"
                    f" {highest_rated_solution} highest_rated_solution:"
                    f" {highest_rated_solution} Solution: {solution}"
                )

                return solution if solution else highest_rated_state

            else:
                return None

        except Exception as e:
            logger.error(f"Error in tot_bfs: {e}")
            return None


############


class DFS(TreeofThoughts):
    """
    Depth-first search implementation for the TreeofThoughts class.

    Args:
        TreeofThoughts (class): Base class for the TreeofThoughtsDFS class.

    Methods:
        solve: Solves the problem using depth-first search algorithm.

    Attributes:
        None.
    """

    def solve(
        self,
        initial_prompt: str = None,
        num_thoughts: int = None,
        max_steps: int = 4,
        value_threshold: float = 0.9,
        pruning_threshold: float = 0.5,
    ):
        output = []

        def dfs(state, step):
            nonlocal output
            if step > max_steps:
                thought = self.model.generate_thoughts(state, 1, initial_prompt)
                value = self.model.evaluate_states({state}, initial_prompt)[
                    state
                ]
                output.append((thought, value))
                return

            thoughts = self.model.generate_thoughts(
                state, self.num_thoughts, initial_prompt
            )
            evaluated_thoughts = self.model.evaluate_states(
                {thought: 0 for thought in thoughts}, initial_prompt
            )
            filtered_thoughts = [
                thought
                for thought in thoughts
                if evaluated_thoughts[thought] >= self.pruning_threshold
            ]

            for next_state in filtered_thoughts:
                state_value = self.model.evaluate_states(
                    {next_state: 0}, initial_prompt
                )[next_state]
```

```
                if state_value > self.value_threshold:
                    child = (
                        (state, next_state)
                        if isinstance(state, str)
                        else (*state, next_state)
                    )
                    dfs(child, step + 1)

        try:
            dfs(initial_prompt, 1)
            best_state, _ = max(output, key=lambda x: x[1])
            solution = self.model.generate_solution(initial_prompt, best_state)
            return solution if solution else best_state
        except Exception as e:
            logger.error(f"Error in tot_dfs: {e}")
            return None


# v2 => best first search => explores state space of the quality of the states
# priority que or greedy BFS
class BESTSearch(TreeofThoughts):
    """
    Represents a tree of thoughts.

    Attributes:
        model: The model used for generating and evaluating thoughts.
        tree: The tree structure to store the thoughts and evaluations.
    """

    def __init__(self, model):
        """
        Initializes a TreeofThoughtsBEST object.

        Args:
            model: The model used for generating and evaluating thoughts.
        """
        self.model = model
        self.tree = {"nodes": {}}

    def save_tree_to_json(self, file_name):
        """
        Saves the tree structure to a JSON file.

        Args:
            file_name: The name of the JSON file to save the tree structure to.
        """
        os.makedirs(os.path.dirname(file_name), exist_ok=True)
        with open(file_name, "w") as json_file:
            json.dump(self.tree, json_file, indent=4)

    def log_new_state(self, state, evaluation):
        """
        Logs a new state and its evaluation in the tree structure.

        Args:
            state: The state to log.
            evaluation: The evaluation of the state.
        """
        state_key = " | ".join(state) if isinstance(state, tuple) else state
        if state_key in self.tree["nodes"]:
            self.tree["nodes"][state_key]["thoughts"].append(evaluation)
        else:
            self.tree["nodes"]["state_key"] = {"thoughts": [evaluation]}

    def solve(self, initial_prompt, num_thoughts, max_steps, pruning_threshold):
        """
        Solves the tree of thoughts problem.

```

```
        Args:
            initial_prompt: The initial prompt for generating thoughts.
            num_thoughts: The number of thoughts to generate at each step.
            max_steps: The maximum number of steps to perform.
            pruning_threshold: The threshold for pruning thoughts.

        Returns:
            The solution to the tree of thoughts problem.
        """
        visited_states = set()
        state_queue = PriorityQueue()

        state_queue.put((0, initial_prompt))

        for _ in range(max_steps):
            if state_queue.empty():
                break

            _, state = state_queue.get()

            if state in visited_states:
                continue

            visited_states.add(state)

            thoughts = self.model.generate_thoughts(
                state, num_thoughts, initial_prompt
            )
            evaluated_thoughts = {
                thought: self.model.evaluate_states(
                    {thought: 0}, initial_prompt
                )[thought]
                for thought in thoughts
            }

            for thought, value in evaluated_thoughts.items():
                if value >= pruning_threshold:
                    new_state = (
                        (state, thought)
                        if isinstance(state, str)
                        else (*state, thought)
                    )
                    state_queue.put((value, new_state))
                    self.log_new_state(new_state, value)

        best_state = max(visited_states, key=self.model.evaluate_states)
        solution = self.model.generate_solution(initial_prompt, best_state)
        print(f"Highest_rated solution: {best_state}  Solution: {solution}")
        return solution if solution else best_state


# A* search algorithm
class ASearch(TreeofThoughts):
    def __init__(self, model):
        self.model = model

    def solve(
        self,
        initial_prompt,
        num_thoughts=5,
        max_steps=30,
        pruning_threshold=0.4,
    ):
        # the open set is implemented as a piorituve quue where the priority is -f_score
        open_set = PriorityQueue()
        open_set.put((0, 0, initial_prompt))

        # the set of visited_states
        visited_states = set()
```

```
        # the g_scores and f-scores are stored as dictionaries
        g_scores = {initial_prompt: 0}
        f_scores = {
            initial_prompt: self.model.evaluate_states(
                {initial_prompt: 0}, initial_prompt
            )[initial_prompt]
        }

        # the parent of each state is stored in a dictionary
        came_from = {}

        for _ in range(max_steps):
            if open_set.empty():
                break

            _, _, current_state = open_set.get()

            if self.is_goal(current_state, f_scores[current_state]):
                return self.reconstruct_path(
                    came_from, current_state, initial_prompt
                )

            thoughts = self.model.generate_thoughts(
                current_state, num_thoughts, initial_prompt
            )
            evaluated_thoughts = {
                thought: self.model.evaluate_states(
                    {thought: 0}, initial_prompt
                )[thought]
                for thought in thoughts
            }

            for thought, value in evaluated_thoughts.items():
                if value < pruning_threshold or thought in visited_states:
                    continue

                tentative_g_score = g_scores[current_state] + 1 / value
                if (
                    thought not in g_scores
                    or tentative_g_score < g_scores[thought]
                ):
                    came_from[thought] = current_state
                    g_scores[thought] = tentative_g_score
                    f_scores[thought] = tentative_g_score + value
                    open_set.put(
                        (-f_scores[thought], g_scores[thought], thought)
                    )

        return self.reconstruct_path(came_from, current_state, initial_prompt)

    def is_goal(self, state, score):
        # if eval state is above 0.9
        return score >= 0.9

    def reconstruct_path(self, came_from, current_state, initial_prompt):
        path = [current_state]
        while current_state in came_from:
            current_state = came_from[current_state]
            path.append(current_state)
        path.reverse()

        path = self.reconstruct_path(came_from, current_state, initial_prompt)
        solution = self.model.generate_solution(initial_prompt, path)
        print(f"Path: {path} solution: {solution}")
        return solution if solution else path


class MonteCarloSearch(TreeofThoughts):
```

```
    """
    A class representing a Monte Carlo Tree of Thoughts.

    Attributes:
        model (Model): The model used for generating thoughts and evaluating states.
        objective (str): The objective of the optimization process.
        solution_found (bool): Indicates whether a solution has been found.
        tree (Dict[str, Dict[str, Union[float, Dict[str, Any]]]]): The tree structure containing nodes,
                                                                   thoughts, and evaluations.

    Methods:
        __init__(self, model, objective="balance"): Initializes a MonteCarloSearch instance.
        optimize_params(self, num_thoughts, max_steps, max_states): Optimizes the parameters based on the
                                                                    objective.
        solve(self, initial_prompt, num_thoughts, max_steps, max_states, pruning_threshold): Solves the problem
                                                                     using Monte Carlo search.
        monte_carlo_search(self, initial_prompt, num_thoughts, max_steps, max_states, pruning_threshold):
                                                                    Performs the Monte Carlo search algorithm.
    """

    def __init__(self, model, objective="balance"):
        super().__init__(model)
        self.objective = objective
        self.solution_found = False
        self.tree: Dict[str, Dict[str, Union[float, Dict[str, Any]]]] = {
            "nodes": {},
            "metrics": {"thoughts": {}, "evaluations": {}},
        }

    def optimize_params(self, num_thoughts, max_steps, max_states):
        """
        Optimizes the parameters based on the objective.

        Args:
            num_thoughts (int): The number of thoughts to generate.
            max_steps (int): The maximum number of steps in the search.
            max_states (int): The maximum number of states to consider.

        Returns:
            Tuple[int, int, int]: The optimized values of num_thoughts, max_steps, and max_states.
        """
        if self.objective == "speed":
            num_thoughts = max(1, num_thoughts - 1)
            max_steps = max(1, max_steps - 1)
            max_states = max(1, max_states - 1)
        elif self.objective == "reliability":
            num_thoughts += 1
            max_steps += 1
            max_states += 1
        elif self.objective == "balance":
            if self.solution_found:
                num_thoughts = max(1, num_thoughts - 1)
                max_steps = max(1, max_steps - 1)
                max_states = max(1, max_states - 1)
            else:
                num_thoughts += 1
                max_steps += 1
                max_states += 1

        return num_thoughts, max_steps, max_states

    def solve(
        self,
        initial_prompt: str,
        num_thoughts: int,
        max_steps: int,
        max_states: int,
        pruning_threshold: float,
    ):
```

```
        """
        Solves the problem using Monte Carlo search.

        Args:
            initial_prompt (str): The initial prompt for the search.
            num_thoughts (int): The number of thoughts to generate.
            max_steps (int): The maximum number of steps in the search.
            max_states (int): The maximum number of states to consider.
            pruning_threshold (float): The threshold for pruning states.

        Returns:
            Union[str, Tuple]: The solution generated by the model or the best state found.
        """
        self.file_name = "logs/tree_of_thoughts_output_montecarlo.json"
        return self.monte_carlo_search(
            initial_prompt,
            num_thoughts,
            max_steps,
            max_states,
            pruning_threshold,
        )

    def monte_carlo_search(
        self,
        initial_prompt: str,
        num_thoughts: int,
        max_steps: int,
        max_states: int,
        pruning_threshold: float,
    ):
        """
        Performs the Monte Carlo search algorithm.

        Args:
            initial_prompt (str): The initial prompt for the search.
            num_thoughts (int): The number of thoughts to generate.
            max_steps (int): The maximum number of steps in the search.
            max_states (int): The maximum number of states to consider.
            pruning_threshold (float): The threshold for pruning states.

        Returns:
            Union[str, Tuple]: The solution generated by the model or the best state found.
        """
        current_states = [initial_prompt]
        state_values = {}
        visit_counts = {initial_prompt: 0}
        transposition_table = {}

        best_state = None
        best_value = float("-inf")

        for step in range(1, max_steps + 1):
            selected_states = []

            for state in current_states:
                if state in transposition_table:
                    transposition_table[state]
                else:
                    time.sleep(1)
                    thoughts = self.model.generate_thoughts(
                        state, num_thoughts, initial_prompt
                    )
                    time.sleep(1)
                    evaluated_thoughts = self.model.evaluate_states(
                        thoughts, initial_prompt
                    )

                    for thought, value in evaluated_thoughts.items():
                        flattened_state = (
```

```
                            (state, thought)
                            if isinstance(state, str)
                            else (*state, thought)
                        )
                        transposition_table[flattened_state] = value

                for thought, value in evaluated_thoughts.items():
                    flattened_state = (
                        (state, thought)
                        if isinstance(state, str)
                        else (*state, thought)
                    )

                    if flattened_state not in visit_counts:
                        visit_counts[flattened_state] = 0

                    if (
                        visit_counts[state] > visit_counts[flattened_state]
                        and visit_counts[flattened_state] > 0
                    ):
                        ucb1_value = value + np.sqrt(
                            2
                            * np.log(visit_counts[state])
                            / visit_counts[flattened_state]
                        )

                        if ucb1_value >= pruning_threshold:
                            selected_states.append(flattened_state)
                            state_values[flattened_state] = value

                            # Update the best state if the current state value is greater than the best value
                            if value > best_value:
                                best_state = flattened_state
                                best_value = value

                visit_counts[state] += 1

            if len(selected_states) > max_states:
                current_states = selected_states[:max_states]
            self.save_tree_to_json(self.file_name)

        # if best_state is not None:
        #     solution = self.model.generate_solution(initial_prompt, best_state)
        #     return solution
        # else:
        #     solution = None

        # return None
        solution = self.model.generate_solution(initial_prompt, best_state)
        return solution if solution else best_state

import os


from dotenv import load_dotenv
from swarms import Agent
from swarms.models import Gemini

load_dotenv()




from swarms import Agent, HuggingfaceLLM

# Get the API key from the environment


# Initialize an agent from swarms
```

```python
agent = Agent(
    agent_name="tree_of_thoughts",
    agent_description=(
        "This agent uses the tree_of_thoughts library to generate thoughts."
    ),
    system_prompt=None,
    llm=Gemini(
    gemini_api_key='AIzaSyDnOxrkz3JKzWu8sAX5S8w7lLrMmCUzO0Q',
    max_tokens=4028,
    temperature=0.7,
    model_name="gemini-1.0-pro"


),
)

# Initialize the ToTAgent class with the API key
model = ToTAgent(
    agent,
    strategy="cot",
    evaluation_strategy="value",
    enable_react=True,
    k=5,
)


# Initialize the MonteCarloSearch class with the model
tree_of_thoughts = MonteCarloSearch(model)

# Define the initial prompt
initial_prompt = """


Three experts with exceptional logical thinking skills are collaboratively answering a question using the tree
                                                    of thoughts method. Each expert will share their
                                                    thought process in detail, taking into account the
                                                    previous thoughts of others and admitting any errors.
                                                    They will iteratively refine and expand upon each other
                                                    's ideas, giving credit where it's due. The process
                                                    continues until a conclusive answer is found. Organize
                                                    the entire response in a markdown table format. The
                                                    task is:

 A heavy nucleus Q of half-life 20 minutes undergoes alpha-decay with a probability of 60% and beta-decay with
                                                    a probability of 40%. Initially, the number of Q nuclei
                                                     is 1000. The number of alpha-decays of Q in the first
                                                    one hour is;
"""

# Define the number of thoughts to generate
num_thoughts = 1
max_steps = 3
max_states = 4
pruning_threshold = 0.5


# Generate the thoughts
solution = tree_of_thoughts.solve(
    initial_prompt=initial_prompt,
    num_thoughts=num_thoughts,
    max_steps=max_steps,
    max_states=max_states,
    pruning_threshold=pruning_threshold,
    # sleep_time=sleep_time
)

print(f"Solution: {solution}")
```

Code for Forest of thought

```
import os
from langchain import PromptTemplate, LLMChain

from langchain_google_genai import ChatGoogleGenerativeAI
from google.colab import userdata

os.environ["GOOGLE_API_KEY"]=userdata.get('GOOGLE_API_KEY')
llm_1 = ChatGoogleGenerativeAI(model="gemini-pro", temperature =0.7)
llm_2 = ChatGoogleGenerativeAI(model="gemini-pro",temperature =0.8)
llm_3 = ChatGoogleGenerativeAI(model="gemini-pro",temperature =0.9)



template_1 = """Imagine three different experts are answering this question.
They will brainstorm the answer step by step reasoning carefully and taking all facts into consideration
All experts will write down 1 step of their thinking,
then share it with the group.
They will each critique their response, and the all the responses of others
They will check their answer based on science and the laws of physics
Then all experts will go on to the next step and write down this step of their thinking.
They will keep going through steps until they reach their conclusion taking into account the thoughts of the
                                                    other experts
If at any time they realise that there is a flaw in their logic they will backtrack to where that flaw occurred
If any expert realises they're wrong at any point then they acknowledges this and start another train of
                                                    thought
Each expert will assign a likelihood of their current assertion being correct
Continue until the experts agree on the single most likely location
They keep checking for dimensionality and keep backsubstituting and check for limiting cases
The question is {question}

The experts reasoning is...
"""

template_2 = """Imagine six different experts are answering this question.
They will brainstorm the answer step by step reasoning carefully and taking all facts into consideration
Each expert will share their first step with all of the other experts
They will each critique their response, and the all the responses of others
They will check their answer being careful to think through any consequences
Each expert will then write down the next step of their thought process
Each expert will assign a likelihood of their current assertion being correct
Continue until the experts agree on the single most likely location
They keep checking for dimensionality and keep backsubstituting and check for limiting cases
The question is {question}

The experts reasoning is...
"""

template_3 = """Imagine four different experts are answering this question.
They will first write down all the facts
They will then consider three different alternative answers and communicate these answers to the other experts
they will write down the likelihood of each answer
Based on this they will each come up with a single answer
They keep checking for dimensionality and keep backsubstituting and check for limiting cases
The question is {question}

The experts reasoning is...
"""


question = """In a historical experiment to determine Planck's constant, a metal surface was irradiated with
                                                    light of different wavelengths. The emitted
                                                    photoelectron energies were measured by applying a
                                                    stopping potential. The relevant data for the
                                                    wavelength $(\lambda)$ of incident light and the
                                                    corresponding stopping potential $\left(V_{0}\right)$
                                                    are given below : \begin{center} \begin{tabular}{cc} \
```

```
                                                    hline $\lambda(\mu \mathrm{m})$ & $V_{0}($ Volt $)$ \\
                                                    \hline 0.3 & 2.0 \\ 0.4 & 1.0 \\ 0.5 & 0.4 \\ \hline \
                                                    end{tabular} \end{center} Given that $c=3 \times 10^{8}
                                                     \mathrm{~m} \mathrm{~s}^{-1}$ and $e=1.6 \times 10^{-
                                                    19} \mathrm{C}$, Planck's constant (in units of $J \
                                                    mathrm{~s}$ ) found from such an experiment is (A) $6.0
                                                     \times 10^{-34}$ (B) $6.4 \times 10^{-34}$ (C) $6.6 \
                                                    times 10^{-34}$ (D) $6.8 \times 10^{-34}$. Check for
                                                    limiting cases, dimensionality and back substitute to
                                                    check whether they satisfy or not"""


prompt1 = PromptTemplate(template=template_1, input_variables=["question"])
prompt2 = PromptTemplate(template=template_2, input_variables=["question"])
prompt3 = PromptTemplate(template=template_3, input_variables=["question"])

llm_chain_1 = LLMChain(prompt=prompt1, llm=llm_1)
llm_chain_2 = LLMChain(prompt=prompt2, llm=llm_2)
llm_chain_3 = LLMChain(prompt=prompt3, llm=llm_3)

response_1 = llm_chain_1.run(question)
response_2 = llm_chain_2.run(question)
response_3 = llm_chain_3.run(question)

print(response_1)
print(response_2)
print(response_3)

get_together = """Several experts have been asked this question
In a historical experiment to determine Planck's constant, a metal surface was irradiated with light of
                                                    different wavelengths. The emitted photoelectron
                                                    energies were measured by applying a stopping potential
                                                    . The relevant data for the wavelength $(\lambda)$ of
                                                    incident light and the corresponding stopping potential
                                                     $\left(V_{0}\right)$ are given below : \begin{center}
                                                    \begin{tabular}{cc} \hline $\lambda(\mu \mathrm{m})$ &
                                                    $V_{0}($ Volt $)$ \\ \hline 0.3 & 2.0 \\ 0.4 & 1.0 \\ 0
                                                    .5 & 0.4 \\ \hline \end{tabular} \end{center} Given
                                                    that $c=3 \times 10^{8} \mathrm{~m} \mathrm{~s}^{-1}$
                                                    and $e=1.6 \times 10^{-19} \mathrm{C}$, Planck's
                                                    constant (in units of $J \mathrm{~s}$ ) found from such
                                                     an experiment is (A) $6.0 \times 10^{-34}$ (B) $6.4 \
                                                    times 10^{-34}$ (C) $6.6 \times 10^{-34}$ (D) $6.8 \
                                                    times 10^{-34}$ their resulting answers are {answer}
which answer is most likely?
The most likely answer is..."""

answer = response_1+response_2+response_3
fusion = PromptTemplate(template=get_together, input_variables=["answer"])

wood_for_trees = ChatGoogleGenerativeAI(model="gemini-pro",temperature =0.25)
final = LLMChain(prompt=fusion, llm=wood_for_trees)
conclusion = final.run(answer)

print(f"Conclusion is {conclusion}")
```

References

[1] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," 2023.

[2] G. Team, R. Anil, S. Borgeaud, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Glaese, J. Chen, E. Pitler, T. Lillicrap, A. Lazaridou, O. Firat, J. Molloy, M. Isard, P. R. Barham, T. Hennigan, B. Lee, F. Viola, M. Reynolds, Y. Xu, R. Doherty, E. Collins, C. Meyer, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, J. Krawczyk, C. Du, E. Chi, H.-T. Cheng,

E. Ni, P. Shah, P. Kane, B. Chan, M. Faruqui, A. Severyn, H. Lin, Y. Li, Y. Cheng, A. Ittycheriah, M. Mahdieh, M. Chen, P. Sun, D. Tran, S. Bagri, B. Lakshminarayanan, J. Liu, A. Orban, F. Güra, H. Zhou, X. Song, A. Boffy, H. Ganapathy, S. Zheng, H. Choe, Ágoston Weisz, T. Zhu, Y. Lu, S. Gopal, J. Kahn, M. Kula, J. Pitman, R. Shah, E. Taropa, M. A. Merey, M. Baeuml, Z. Chen, L. E. Shafey, Y. Zhang, O. Sercinoglu, G. Tucker, E. Piqueras, M. Krikun, I. Barr, N. Savinov, I. Danihelka, B. Roelofs, A. White, A. Andreassen, T. von Glehn, L. Yagati, M. Kazemi, L. Gonzalez, M. Khalman, J. Sygnowski, A. Frechette, C. Smith, L. Culp, L. Proleev, Y. Luan, X. Chen, J. Lottes, N. Schucher, F. Lebron, A. Rrustemi, N. Clay, P. Crone, T. Kocisky, J. Zhao, B. Perz, D. Yu, H. Howard, A. Bloniarz, J. W. Rae, H. Lu, L. Sifre, M. Maggioni, F. Alcober, D. Garrette, M. Barnes, S. Thakoor, J. Austin, G. Barth-Maron, W. Wong, R. Joshi, R. Chaabouni, D. Fatiha, A. Ahuja, G. S. Tomar, E. Senter, M. Chadwick, I. Kornakov, N. Attaluri, I. Iturrate, R. Liu, Y. Li, S. Cogan, J. Chen, C. Jia, C. Gu, Q. Zhang, J. Grimstad, A. J. Hartman, X. Garcia, T. S. Pillai, J. Devlin, M. Laskin, D. de Las Casas, D. Valter, C. Tao, L. Blanco, A. P. Badia, D. Reitter, M. Chen, J. Brennan, C. Rivera, S. Brin, S. Iqbal, G. Surita, J. Labanowski, A. Rao, S. Winkler, E. Parisotto, Y. Gu, K. Olszewska, R. Addanki, A. Miech, A. Louis, D. Teplyashin, G. Brown, E. Catt, J. Balaguer, J. Xiang, P. Wang, Z. Ashwood, A. Briukhov, A. Webson, S. Ganapathy, S. Sanghavi, A. Kannan, M.-W. Chang, A. Stjerngren, J. Djolonga, Y. Sun, A. Bapna, M. Aitchison, P. Pejman, H. Michalewski, T. Yu, C. Wang, J. Love, J. Ahn, D. Bloxwich, K. Han, P. Humphreys, T. Sellam, J. Bradbury, V. Godbole, S. Samangooei, B. Damoc, A. Kaskasoli, S. M. R. Arnold, V. Vasudevan, S. Agrawal, J. Riesa, D. Lepikhin, R. Tanburn, S. Srinivasan, H. Lim, S. Hodkinson, P. Shyam, J. Ferret, S. Hand, A. Garg, T. L. Paine, J. Li, Y. Li, M. Giang, A. Neitz, Z. Abbas, S. York, M. Reid, E. Cole, A. Chowdhery, D. Das, D. Rogozińska, V. Nikolaev, P. Sprechmann, Z. Nado, L. Zilka, F. Prost, L. He, M. Monteiro, G. Mishra, C. Welty, J. Newlan, D. Jia, M. Allamanis, C. H. Hu, R. de Liedekerke, J. Gilmer, C. Saroufim, S. Rijhwani, S. Hou, D. Shrivastava, A. Baddepudi, A. Goldin, A. Ozturel, A. Cassirer, Y. Xu, D. Sohn, D. Sachan, R. K. Amplayo, C. Swanson, D. Petrova, S. Narayan, A. Guez, S. Brahma, J. Landon, M. Patel, R. Zhao, K. Villela, L. Wang, W. Jia, M. Rahtz, M. Giménez, L. Yeung, J. Keeling, P. Georgiev, D. Mincu, B. Wu, S. Haykal, R. Saputro, K. Vodrahalli, J. Qin, Z. Cankara, A. Sharma, N. Fernando, W. Hawkins, B. Neyshabur, S. Kim, A. Hutter, P. Agrawal, A. Castro-Ros, G. van den Driessche, T. Wang, F. Yang, S. yiin Chang, P. Komarek, R. McIlroy, M. Lučić, G. Zhang, W. Farhan, M. Sharman, P. Natsev, P. Michel, Y. Bansal, S. Qiao, K. Cao, S. Shakeri, C. Butterfield, J. Chung, P. K. Rubenstein, S. Agrawal, A. Mensch, K. Soparkar, K. Lenc, T. Chung, A. Pope, L. Maggiore, J. Kay, P. Jhakra, S. Wang, J. Maynez, M. Phuong, T. Tobin, A. Tacchetti, M. Trebacz, K. Robinson, Y. Katariya, S. Riedel, P. Bailey, K. Xiao, N. Ghelani, L. Aroyo, A. Slone, N. Houlsby, X. Xiong, Z. Yang, E. Gribovskaya, J. Adler, M. Wirth, L. Lee, M. Li, T. Kagohara, J. Pavagadhi, S. Bridgers, A. Bortsova, S. Ghemawat, Z. Ahmed, T. Liu, R. Powell, V. Bolina, M. Iinuma, P. Zablotskaia, J. Besley, D.-W. Chung, T. Dozat, R. Comanescu, X. Si, J. Greer, G. Su, M. Polacek, R. L. Kaufman, S. Tokumine, H. Hu, E. Buchatskaya, Y. Miao, M. Elhawaty, A. Siddhant, N. Tomasev, J. Xing, C. Greer, H. Miller, S. Ashraf, A. Roy, Z. Zhang, A. Ma, A. Filos, M. Besta, R. Blevins, T. Klimenko, C.-K. Yeh, S. Changpinyo, J. Mu, O. Chang, M. Pajarskas, C. Muir, V. Cohen, C. L. Lan, K. Haridasan, A. Marathe, S. Hansen, S. Douglas, R. Samuel, M. Wang, S. Austin, C. Lan, J. Jiang, J. Chiu, J. A. Lorenzo, L. L. Sjösund, S. Cevey, Z. Gleicher, T. Avrahami, A. Boral, H. Srinivasan, V. Selo, R. May, K. Aisopos, L. Hussenot, L. B. Soares, K. Baumli, M. B. Chang, A. Recasens, B. Caine, A. Pritzel, F. Pavetic, F. Pardo, A. Gergely, J. Frye, V. Ramasesh, D. Horgan, K. Badola, N. Kassner, S. Roy, E. Dyer, V. C. Campos, A. Tomala, Y. Tang, D. E. Badawy, E. White, B. Mustafa, O. Lang, A. Jindal, S. Vikram, Z. Gong, S. Caelles, R. Hemsley, G. Thornton, F. Feng, W. Stokowiec, C. Zheng, P. Thacker, Çağlar Ünlü, Z. Zhang, M. Saleh, J. Svensson, M. Bileschi, P. Patil, A. Anand, R. Ring, K. Tsihlas, A. Vezer, M. Selvi, T. Shevlane, M. Rodriguez, T. Kwiatkowski, S. Daruki, K. Rong, A. Dafoe, N. FitzGerald, K. Gu-Lemberg, M. Khan, L. A. Hendricks, M. Pellat, V. Feinberg, J. Cobon-Kerr, T. Sainath, M. Rauh, S. H. Hashemi, R. Ives, Y. Hasson, E. Noland, Y. Cao, N. Byrd, L. Hou, Q. Wang, T. Sottiaux, M. Paganini, J.-B. Lespiau, A. Moufarek, S. Hassan, K. Shivakumar, J. van Amersfoort, A. Mandhane, P. Joshi, A. Goyal, M. Tung, A. Brock, H. Sheahan, V. Misra, C. Li, N. Rakićević, M. Dehghani, F. Liu, S. Mittal, J. Oh, S. Noury, E. Sezener, F. Huot, M. Lamm, N. D. Cao, C. Chen, S. Mudgal, R. Stella, K. Brooks, G. Vasudevan, C. Liu, M. Chain, N. Melinkeri, A. Cohen, V. Wang, K. Seymore, S. Zubkov, R. Goel, S. Yue, S. Krishnakumaran, B. Albert, N. Hurley, M. Sano, A. Mohananey, J. Joughin, E. Filonov, T. Kępa, Y. Eldawy, J. Lim, R. Rishi, S. Badiezadegan, T. Bos, J. Chang, S. Jain, S. G. S. Padmanabhan, S. Puttagunta, K. Krishna, L. Baker, N. Kalb, V. Bedapudi, A. Kurzrok, S. Lei, A. Yu, O. Litvin, X. Zhou, Z. Wu, S. Sobell, A. Siciliano, A. Papir, R. Neale, J. Bragagnolo, T. Toor, T. Chen, V. Anklin, F. Wang, R. Feng, M. Gholami, K. Ling, L. Liu, J. Walter, H. Moghaddam, A. Kishore, J. Adamek, T. Mercado, J. Mallinson, S. Wandekar, S. Cagle, E. Ofek, G. Garrido, C. Lombriser, M. Mukha, B. Sun, H. R. Mohammad, J. Matak, Y. Qian, V. Peswani, P. Janus, Q. Yuan, L. Schelin, O. David, A. Garg, Y. He, O. Duzhyi, A. Älgmyr, T. Lottaz, Q. Li, V. Yadav, L. Xu, A. Chinien, R. Shivanna, A. Chuklin, J. Li, C. Spadine, T. Wolfe, K. Mohamed,

S. Das, Z. Dai, K. He, D. von Dincklage, S. Upadhyay, A. Maurya, L. Chi, S. Krause, K. Salama, P. G. Rabinovitch, P. K. R. M, A. Selvan, M. Dektiarev, G. Ghiasi, E. Guven, H. Gupta, B. Liu, D. Sharma, I. H. Shtacher, S. Paul, O. Akerlund, F.-X. Aubet, T. Huang, C. Zhu, E. Zhu, E. Teixeira, M. Fritze, F. Bertolini, L.-E. Marinescu, M. Bölle, D. Paulus, K. Gupta, T. Latkar, M. Chang, J. Sanders, R. Wilson, X. Wu, Y.-X. Tan, L. N. Thiet, T. Doshi, S. Lall, S. Mishra, W. Chen, T. Luong, S. Benjamin, J. Lee, E. Andrejczuk, D. Rabiej, V. Ranjan, K. Styrc, P. Yin, J. Simon, M. R. Harriott, M. Bansal, A. Robsky, G. Bacon, D. Greene, D. Mirylenka, C. Zhou, O. Sarvana, A. Goyal, S. Andermatt, P. Siegler, B. Horn, A. Israel, F. Pongetti, C.-W. L. Chen, M. Selvatici, P. Silva, K. Wang, J. Tolins, K. Guu, R. Yogev, X. Cai, A. Agostini, M. Shah, H. Nguyen, N. Donnaile, S. Pereira, L. Friso, A. Stambler, A. Kurzrok, C. Kuang, Y. Romanikhin, M. Geller, Z. Yan, K. Jang, C.-C. Lee, W. Fica, E. Malmi, Q. Tan, D. Banica, D. Balle, R. Pham, Y. Huang, D. Avram, H. Shi, J. Singh, C. Hidey, N. Ahuja, P. Saxena, D. Dooley, S. P. Potharaju, E. O'Neill, A. Gokulchandran, R. Foley, K. Zhao, M. Dusenberry, Y. Liu, P. Mehta, R. Kotikalapudi, C. Safranek-Shrader, A. Goodman, J. Kessinger, E. Globen, P. Kolhar, C. Gorgolewski, A. Ibrahim, Y. Song, A. Eichenbaum, T. Brovelli, S. Potluri, P. Lahoti, C. Baetu, A. Ghorbani, C. Chen, A. Crawford, S. Pal, M. Sridhar, P. Gurita, A. Mujika, I. Petrovski, P.-L. Cedoz, C. Li, S. Chen, N. D. Santo, S. Goyal, J. Punjabi, K. Kappaganthu, C. Kwak, P. LV, S. Velury, H. Choudhury, J. Hall, P. Shah, R. Figueira, M. Thomas, M. Lu, T. Zhou, C. Kumar, T. Jurdi, S. Chikkerur, Y. Ma, A. Yu, S. Kwak, V. Ähdel, S. Rajayogam, T. Choma, F. Liu, A. Barua, C. Ji, J. H. Park, V. Hellendoorn, A. Bailey, T. Bilal, H. Zhou, M. Khatir, C. Sutton, W. Rzadkowski, F. Macintosh, K. Shagin, P. Medina, C. Liang, J. Zhou, P. Shah, Y. Bi, A. Dankovics, S. Banga, S. Lehmann, M. Bredesen, Z. Lin, J. E. Hoffmann, J. Lai, R. Chung, K. Yang, N. Balani, A. Bražinskas, A. Sozanschi, M. Hayes, H. F. Alcalde, P. Makarov, W. Chen, A. Stella, L. Snijders, M. Mandl, A. Kärrman, P. Nowak, X. Wu, A. Dyck, K. Vaidyanathan, R. R, J. Mallet, M. Rudominer, E. Johnston, S. Mittal, A. Udathu, J. Christensen, V. Verma, Z. Irving, A. Santucci, G. Elsayed, E. Davoodi, M. Georgiev, I. Tenney, N. Hua, G. Cideron, E. Leurent, M. Alnahlawi, I. Georgescu, N. Wei, I. Zheng, D. Scandinaro, H. Jiang, J. Snoek, M. Sundararajan, X. Wang, Z. Ontiveros, I. Karo, J. Cole, V. Rajashekhar, L. Tumeh, E. Ben-David, R. Jain, J. Uesato, R. Datta, O. Bunyan, S. Wu, J. Zhang, P. Stanczyk, Y. Zhang, D. Steiner, S. Naskar, M. Azzam, M. Johnson, A. Paszke, C.-C. Chiu, J. S. Elias, A. Mohiuddin, F. Muhammad, J. Miao, A. Lee, N. Vieillard, J. Park, J. Zhang, J. Stanway, D. Garmon, A. Karmarkar, Z. Dong, J. Lee, A. Kumar, L. Zhou, J. Evens, W. Isaac, G. Irving, E. Loper, M. Fink, I. Arkatkar, N. Chen, I. Shafran, I. Petrychenko, Z. Chen, J. Jia, A. Levskaya, Z. Zhu, P. Grabowski, Y. Mao, A. Magni, K. Yao, J. Snaider, N. Casagrande, E. Palmer, P. Suganthan, A. Castaño, I. Giannoumis, W. Kim, M. Rybiński, A. Sreevatsa, J. Prendki, D. Soergel, A. Goedeckemeyer, W. Gierke, M. Jafari, M. Gaba, J. Wiesner, D. G. Wright, Y. Wei, H. Vashisht, Y. Kulizhskaya, J. Hoover, M. Le, L. Li, C. Iwuanyanwu, L. Liu, K. Ramirez, A. Khorlin, A. Cui, T. LIN, M. Wu, R. Aguilar, K. Pallo, A. Chakladar, G. Perng, E. A. Abellan, M. Zhang, I. Dasgupta, N. Kushman, I. Penchev, A. Repina, X. Wu, T. van der Weide, P. Ponnapalli, C. Kaplan, J. Simsa, S. Li, O. Dousse, F. Yang, J. Piper, N. Ie, R. Pasumarthi, N. Lintz, A. Vijayakumar, D. Andor, P. Valenzuela, M. Lui, C. Paduraru, D. Peng, K. Lee, S. Zhang, S. Greene, D. D. Nguyen, P. Kurylowicz, C. Hardin, L. Dixon, L. Janzer, K. Choo, Z. Feng, B. Zhang, A. Singhal, D. Du, D. McKinnon, N. Antropova, T. Bolukbasi, O. Keller, D. Reid, D. Finchelstein, M. A. Raad, R. Crocker, P. Hawkins, R. Dadashi, C. Gaffney, K. Franko, A. Bulanova, R. Leblond, S. Chung, H. Askham, L. C. Cobo, K. Xu, F. Fischer, J. Xu, C. Sorokin, C. Alberti, C.-C. Lin, C. Evans, A. Dimitriev, H. Forbes, D. Banarse, Z. Tung, M. Omernick, C. Bishop, R. Sterneck, R. Jain, J. Xia, E. Amid, F. Piccinno, X. Wang, P. Banzal, D. J. Mankowitz, A. Polozov, V. Krakovna, S. Brown, M. Bateni, D. Duan, V. Firoiu, M. Thotakuri, T. Natan, M. Geist, S. tan Girgin, H. Li, J. Ye, O. Roval, R. Tojo, M. Kwong, J. Lee-Thorp, C. Yew, D. Sinopalnikov, S. Ramos, J. Mellor, A. Sharma, K. Wu, D. Miller, N. Sonnerat, D. Vnukov, R. Greig, J. Beattie, E. Caveness, L. Bai, J. Eisenschlos, A. Korchemniy, T. Tsai, M. Jasarevic, W. Kong, P. Dao, Z. Zheng, F. Liu, F. Yang, R. Zhu, T. H. Teh, J. Sanmiya, E. Gladchenko, N. Trdin, D. Toyama, E. Rosen, S. Tavakkol, L. Xue, C. Elkind, O. Woodman, J. Carpenter, G. Papamakarios, R. Kemp, S. Kafle, T. Grunina, R. Sinha, A. Talbert, D. Wu, D. Owusu-Afriyie, C. Du, C. Thornton, J. Pont-Tuset, P. Narayana, J. Li, S. Fatehi, J. Wieting, O. Ajmeri, B. Uria, Y. Ko, L. Knight, A. Héliou, N. Niu, S. Gu, C. Pang, Y. Li, N. Levine, A. Stolovich, R. Santamaria-Fernandez, S. Goenka, W. Yustalim, R. Strudel, A. Elqursh, C. Deck, H. Lee, Z. Li, K. Levin, R. Hoffmann, D. Holtmann-Rice, O. Bachem, S. Arora, C. Koh, S. H. Yeganeh, S. Põder, M. Tariq, Y. Sun, L. Ionita, M. Seyedhosseini, P. Tafti, Z. Liu, A. Gulati, J. Liu, X. Ye, B. Chrzaszcz, L. Wang, N. Sethi, T. Li, B. Brown, S. Singh, W. Fan, A. Parisi, J. Stanton, V. Koverkathu, C. A. Choquette-Choo, Y. Li, T. Lu, A. Ittycheriah, P. Shroff, M. Varadarajan, S. Bahargam, R. Willoughby, D. Gaddy, G. Desjardins, M. Cornero, B. Robenek, B. Mittal, B. Albrecht, A. Shenoy, F. Moiseev, H. Jacobsson, A. Ghaffarkhah, M. Rivière, A. Walton, C. Crepy, A. Parrish, Z. Zhou, C. Farabet, C. Radebaugh, P. Srinivasan, C. van der Salm, A. Fidjeland, S. Scellato, E. Latorre-Chimoto, H. Klimczak-Plucińska, D. Bridson, D. de Cesare, T. Hudson, P. Mendolicchio, L. Walker, A. Morris, M. Mauger, A. Guseynov, A. Reid, S. Odoom, L. Loher, V. Cotruta,

M. Yenugula, D. Grewe, A. Petrushkina, T. Duerig, A. Sanchez, S. Yadlowsky, A. Shen, A. Globerson, L. Webb, S. Dua, D. Li, S. Bhupatiraju, D. Hurt, H. Qureshi, A. Agarwal, T. Shani, M. Eyal, A. Khare, S. R. Belle, L. Wang, C. Tekur, M. S. Kale, J. Wei, R. Sang, B. Saeta, T. Liechty, Y. Sun, Y. Zhao, S. Lee, P. Nayak, D. Fritz, M. R. Vuyyuru, J. Aslanides, N. Vyas, M. Wicke, X. Ma, E. Eltyshev, N. Martin, H. Cate, J. Manyika, K. Amiri, Y. Kim, X. Xiong, K. Kang, F. Luisier, N. Tripuraneni, D. Madras, M. Guo, A. Waters, O. Wang, J. Ainslie, J. Baldridge, H. Zhang, G. Pruthi, J. Bauer, F. Yang, R. Mansour, J. Gelman, Y. Xu, G. Polovets, J. Liu, H. Cai, W. Chen, X. Sheng, E. Xue, S. Ozair, C. Angermueller, X. Li, A. Sinha, W. Wang, J. Wiesinger, E. Koukoumidis, Y. Tian, A. Iyer, M. Gurumurthy, M. Goldenson, P. Shah, M. Blake, H. Yu, A. Urbanowicz, J. Palomaki, C. Fernando, K. Durden, H. Mehta, N. Momchev, E. Rahimtoroghi, M. Georgaki, A. Raul, S. Ruder, M. Redshaw, J. Lee, D. Zhou, K. Jalan, D. Li, B. Hechtman, P. Schuh, M. Nasr, K. Milan, V. Mikulik, J. Franco, T. Green, N. Nguyen, J. Kelley, A. Mahendru, A. Hu, J. Howland, B. Vargas, J. Hui, K. Bansal, V. Rao, R. Ghiya, E. Wang, K. Ye, J. M. Sarr, M. M. Preston, M. Elish, S. Li, A. Kaku, J. Gupta, I. Pasupat, D.-C. Juan, M. Someswar, T. M., X. Chen, A. Amini, A. Fabrikant, E. Chu, X. Dong, A. Muthal, S. Buthpitiya, S. Jauhari, N. Hua, U. Khandelwal, A. Hitron, J. Ren, L. Rinaldi, S. Drath, A. Dabush, N.-J. Jiang, H. Godhia, U. Sachs, A. Chen, Y. Fan, H. Taitelbaum, H. Noga, Z. Dai, J. Wang, C. Liang, J. Hamer, C.-S. Ferng, C. Elkind, A. Atias, P. Lee, V. Listík, M. Carlen, J. van de Kerkhof, M. Pikus, K. Zaher, P. Müller, S. Zykova, R. Stefanec, V. Gatsko, C. Hirnschall, A. Sethi, X. F. Xu, C. Ahuja, B. Tsai, A. Stefanoiu, B. Feng, K. Dhandhania, M. Katyal, A. Gupta, A. Parulekar, D. Pitta, J. Zhao, V. Bhatia, Y. Bhavnani, O. Alhadlaq, X. Li, P. Danenberg, D. Tu, A. Pine, V. Filippova, A. Ghosh, B. Limonchik, B. Urala, C. K. Lanka, D. Clive, Y. Sun, E. Li, H. Wu, K. Hongtongsak, I. Li, K. Thakkar, K. Omarov, K. Majmundar, M. Alverson, M. Kucharski, M. Patel, M. Jain, M. Zabelin, P. Pelagatti, R. Kohli, S. Kumar, J. Kim, S. Sankar, V. Shah, L. Ramachandruni, X. Zeng, B. Bariach, L. Weidinger, A. Subramanya, S. Hsiao, D. Hassabis, K. Kavukcuoglu, A. Sadovsky, Q. Le, T. Strohman, Y. Wu, S. Petrov, J. Dean, and O. Vinyals, "Gemini: A family of highly capable multimodal models," 2024.

[3] 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," *arXiv preprint arXiv:2110.14168*, 2021.

[4] G. Juneja, S. Dutta, S. Chakrabarti, S. Manchanda, and T. Chakraborty, "Small language models fine-tuned to coordinate larger language models improve complex reasoning," 2024.

[5] T. Trinh, Y. Wu, Q. Le, H. He, and T. Luong, "Solving olympiad geometry without human demonstrations," *Nature*, 2024.

[6] J. Long, "Large language model guided tree-of-thought," 2023.

[7] D. Arora, H. G. Singh, and Mausam, "Have llms advanced enough? a challenging problem solving benchmark for large language models," 2023.