Enhancing Large Language Models through Transforming Reasoning Problems into Classification Tasks

Tarun Raheja[*◇] , **Raunak Sinha**[†◇], **Advit Deepak**[△†◇], **Will Healy**[‡◇],
Jayanth Srinivasa[△], **Myungjin Lee**[△], **Ramana Kompella**[△]
[*]University of Pennsylvania, [†]University of California, Los Angeles, [△]Cisco Research, [‡]Stanford University
traheja@upenn.edu, whealy@stanford.edu, raunaksinha@g.ucla.edu,
{addeepak, jasriniv, rkompella, myungjle}@cisco.com

Abstract

In this paper, we introduce a novel approach for enhancing the reasoning capabilities of large language models (LLMs) for constraint satisfaction problems (CSPs) by converting reasoning problems into classification tasks. Our method leverages the LLM's ability to decide when to call a function from a set of logical-linguistic primitives, each of which can interact with a local "scratchpad" memory and logical inference engine. Invocation of these primitives in the correct order writes the constraints to the scratchpad memory and enables the logical engine to verifiably solve the problem. We additionally propose a formal framework for exploring the "linguistic" hardness of CSP reasoning-problems for LLMs. Our experimental results demonstrate that under our proposed method, tasks with significant computational hardness can be converted to a form that is easier for LLMs to solve and yields a 40% improvement over baselines. This opens up new avenues for future research into hybrid cognitive models that integrate symbolic and neural approaches.

Keywords: logical-reasoning, textual-inference, constraint-satisfaction-problems, large-language-models

1. Introduction

The remarkable abilities of large language models (LLMs) have opened up new opportunities and challenges in the realm of artificial intelligence (AI) (Brown et al., 2020; Radford et al., 2018; Bubeck et al., 2023; OpenAI, 2023b; Wei et al., 2020). Their performance in tasks such as natural language understanding, translation, and content generation often surpasses expectations (Zhao et al., 2023; Rajasekharan et al., 2023; Zhu et al., 2023; Franceschelli and Musolesi, 2023).

However, LLMs continue to struggle with tasks that require explicit reasoning (Huang and Chang, 2023; Arkoudas, 2023; Schaeffer et al., 2023). We also empirically observe that in the context of Constraint Satisfaction Problems (CSPs), LLMs fail at both reasoning and ranking (Section 3.2 and Section 3.3). These intrinsic limitations in LLMs' reasoning capabilities pose a significant barrier to their application in more complex tasks.

Active research into improving reasoning in LLMs can be categorized into three main approaches: prompting based methods (Section 2.1), tool based methods (Section 2.2), and world model aided methods (Section 2.3). While each of these classes of methods has proven effective in various contexts, they face challenges in handling specific reasoning tasks, as evidenced by BIG-bench (Srivastava et al., 2022), a diverse benchmark for LLMs.

Our method aims to overcome these reasoning limitations by harnessing the classification abilities of LLMs. Specifically, we first break down the process of logical reasoning in a particular context (e.g. spatial reasoning) into a set of mutually independent primitives (Section 3.4.1). Each of these primitives is stored in local memory and in a logical inference engine. We then use an LLM which has been fine-tuned to select amongst APIs (OpenAI, 2023a). This enables the LLM to select amongst these primitives and write these primitives to memory in a structured manner (Section 3.4). The primitives create a representation of the problem's constraints in memory. The logical inference engine then solves the problem given these constraints.

Figure 1 shows a working example of this methodology on a CSP. The input problem is first parsed into component sentences. Next, the LLM determines the appropriate primitive for each sentence. Each primitive changes the state of the memory. The LLM also converts each answer choice into a primitive, but does not act on these primitives. The logical inference engine validates these choices based on the final memory state. The generalized algorithm can be found in Section 3.6.

Our findings are summarized in Section 4.3, where we observe that on the logical deduction datasets in BIG-Bench Hard (Suzgun et al. (2022)), our method achieves an absolute improvement of nearly 40 % on the baselines, while also being more explainable. In Section 4.2, we find that simpler primitives are much easier for reasoning with LLMs than complex primitives.

This paper is structured as follows. In Section 2,

[◇]Work done during summer internship at Cisco Research.

20-25 May, 2024. © 2024 ELRA Language Resource Association: CC BY-NC 4.0

we review the current understanding of reasoning capabilities and limitations in LLMs. Next, in Section 3, we explain our proposed method in detail, followed by an introduction to the concepts of computational hardness and linguistic hardness in the context of reasoning in LLMs. We then present a series of experiments validating our approach and subsequently discuss the results in Section 4.[1]

2. Background and Related Work

We present a survey of approaches for improving reasoning in LLMs and the broader context of the family of methods they fall under in Section 2.1, Section 2.2, and Section 2.3.

2.1. Prompting Based Methods

Wei et al. (2022) introduce chain-of-thought prompting, improving LLMs' reasoning abilities and achieving state-of-the-art accuracy on the GSM8K benchmark (Cobbe et al., 2021). Wang et al. (2022) propose iterative-consistency decoding, which samples diverse reasoning paths and selects the most consistent answer, enhancing performance on complex reasoning tasks. "Tree of Thoughts" by Yao et al. (2023) allows for deliberate decision-making by exploring multiple reasoning paths, with the capability to look ahead or backtrack. Clark et al. (2020) demonstrate transformers as "soft theorem provers" over explicit language theories, opening possibilities for explainability and counterfactual reasoning in Question Answering (QA).

2.2. Tool Based Methods

"Program of Thoughts" by Chen et al. (2022) uses language models and a program interpreter for numerical reasoning, outperforming chain-of-thoughts prompting by around 12% on average. He-Yueya et al. (2023) combined an LLM with a symbolic solver for math word problems, showing comparable or superior performance to prior methods and emphasizing incremental representations. Pan et al. (2023)'s LOGIC-LM framework combines LLMs with symbolic reasoning and an iterative-refinement stage, improving logical problem-solving accuracy. Gao et al. (2023)'s Program-Aided Language models (PAL) blend LLMs with a symbolic interpreter, outperforming larger models in natural language reasoning tasks.

2.3. World Model Aided Methods

Hao et al. (2023)'s Reasoning via Planning (RAP) framework repurposes the LLM for strategic exploration in reasoning, enhancing LLMs' reasoning capabilities. Voyager by Wang et al. (2023), an LLM-powered agent in Minecraft, combines exploration, skill library expansion, and iterative prompting, demonstrating proficiency and generalization in the game environment.

In contrast to these approaches, our method introduces a novel paradigm, in which we convert reasoning tasks into classification tasks, a strategy that diverges significantly from the existing methods. Our approach aims to leverage the strengths of LLMs in a new and unique way, substantially improving their problem-solving capabilities.

3. Methodology

In this section, we identify LLM reasoning failures, formalize reasoning as a classification task, and discuss computational versus linguistic hardness. We introduce effective primitives for spatial reasoning and offer guidelines for other tasks. The overall algorithms are also discussed in this section.

3.1. Dataset

The logical deduction datasets in Beyond the Imitation Game Benchmark Hard (BBH) (Suzgun et al. (2022)) are designed to test logical reasoning skills. It requires the deduction of the order of a sequence of objects based on minimal conditions. Each instance in this dataset involves 3 to 7 similar objects (e.g., colored books on a shelf) and non-redundant clues about their placement (e.g., "the red book is to the right of the green book").

The goal is to correctly identify the position of each object. This task measures the model's ability to parse multiple objects' relationships, understand ordering rules, and perform multi-step logical reasoning. It targets the model's capacity for complex deduction rather than superficial pattern recognition, providing a meaningful challenge for current language models.

We focus on three datasets in the benchmark: `logical_deduction_with_seven_objects`, `logical_deduction_with_five_objects`, `logical_deduction_with_three_objects`. For our dataset, we randomly sample 100 problems from each of these datasets, which yields a total of 300 problems.

Table 1 provides an insight into the distribution of problem types across different numbers of objects. It is evident that the "leftright" problem type dominates when dealing with 3 and 7 objects, comprising 48% and 46% of the cases, respectively. On the other hand, with 5 objects, there is a noticeable increase in "competition" problems (24%) and a corresponding decrease in "leftright" problems (36%). The distribution of "age" and "price" prob-

[1]We provide a link to our anonymized github repository https://anonymous.4open.science/r/LLM_logical_reasoning/

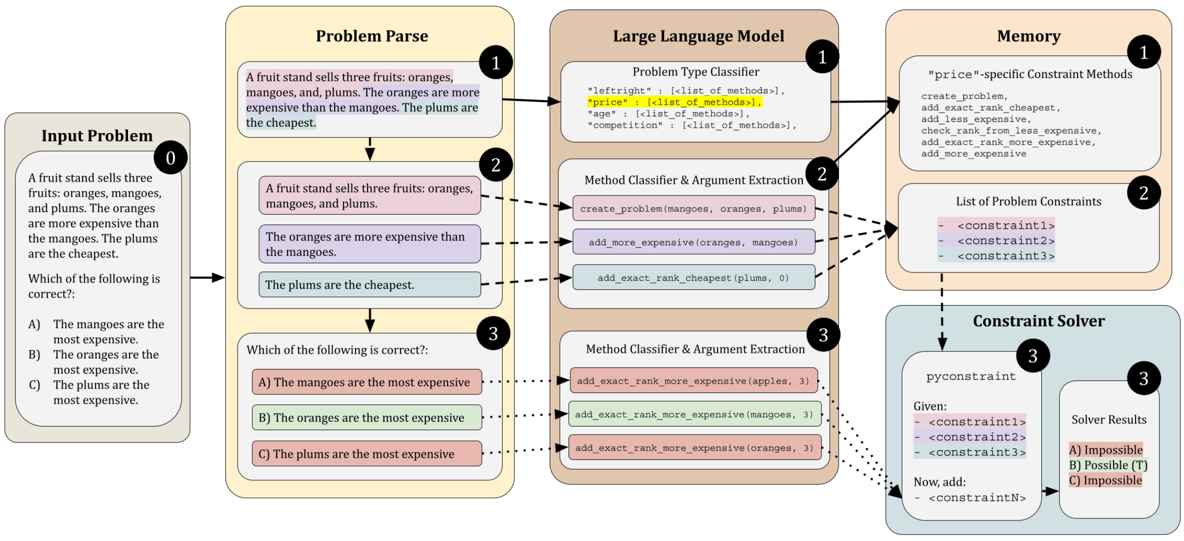

Input Problem
0
A fruit stand sells three fruits: oranges, mangoes, and plums. The oranges are more expensive than the mangoes. The plums are the cheapest.
Which of the following is correct?:
A) The mangoes are the most expensive.
B) The oranges are the most expensive.
C) The plums are the most expensive.
Problem Parse
1
A fruit stand sells three fruits: oranges, mangoes, and, plums. The oranges are more expensive than the mangoes. The plums are the cheapest.
2
A fruit stand sells three fruits: oranges, mangoes, and plums.
The oranges are more expensive than the mangoes.
The plums are the cheapest.
3
Which of the following is correct?:
A) The mangoes are the most expensive
B) The oranges are the most expensive
C) The plums are the most expensive
Large Language Model
1
Problem Type Classifier
"leftright" : [<list_of_methods>],
"price" : [<list_of_methods>],
"age" : [<list_of_methods>],
"competition" : [<list_of_methods>],
2
Method Classifier & Argument Extraction
create_problem(mangoes, oranges, plums)
add_more_expensive(oranges, mangoes)
add_exact_rank_cheapest(plums, 0)
3
Method Classifier & Argument Extraction
add_exact_rank_more_expensive(apples, 3)
add_exact_rank_more_expensive(mangoes, 3)
add_exact_rank_more_expensive(oranges, 3)
Memory
1
"price"-specific Constraint Methods
create_problem,
add_exact_rank_cheapest,
add_less_expensive,
check_rank_from_less_expensive,
add_exact_rank_more_expensive,
add_more_expensive
2
List of Problem Constraints
- <constraint1>
- <constraint2>
- <constraint3>
Constraint Solver
3
pyconstraint
Given:
- <constraint1>
- <constraint2>
- <constraint3>
Now, add:
- <constraintN>
3
Solver Results
A) Impossible
B) Possible (T)
C) Impossible

Figure 1: **Working example of proposed method on reasoning problem.** Stage 0: A problem from BIG-bench Hard is inputted. Stage 1: Classification of problem type as "price". Constraint methods for "price" are fetched from memory. Stage 2: Problem paragraph is decomposed into sentences. Each sentence is mapped to a constraint method and the relevant arguments are extracted by the LLM. Each sentence's equivalent constraint is added to a growing list in memory. Stage 3: The answer choice section is decomposed into individual choices. Each choice is mapped to a constraint method and the relevant arguments are extracted by the LLM. This method is passed to a constraint solver along with the preceding list, and the plausibilty of such a scenario is determined. Whichever scenario is deemed possible is marked as True. In this case, option B) is the only possible choice.

lem types appears relatively consistent across the three scenarios, with only minor variations.

Problem Type	3 Objects	5 Objects	7 Objects
leftright	48%	36%	46%
competition	18%	24%	11%
price	16%	19%	22%
age	18%	21%	21%

Table 1: Distribution of problem types based on number of objects.

3.2. Motivation: Observed Failure Modes with Weaker LLMs

We investigate reasoning problems as shown in Figure 1. Using Chain-of-Thought (CoT) or self consistency often results in confident yet incorrect answers from LLMs, as shown in Figure 2. Further quantitative results are provided in Section 4.3.

3.3. Motivation: Correlation Analysis of Ranking by CoT

We select 60 uniformly-distributed samples across problem sizes and types with gold standard rankings. Using CoT prompting, we compute rankings and assess them using spearman rank correlation. Table 2 shows that correlation is generally strongest for 3-object problems and declines with problem size. Notably, CoT fails to rank the "price" problem type correctly for 5 objects.

Type	3 Objects	5 Objects	7 Objects
leftright	**0.8000**	0.5500	0.5553
competition	**0.9000**	0.7800	0.5949
price	**0.8000**	-0.2500	0.4296
age	0.6250	**0.7200**	0.3333

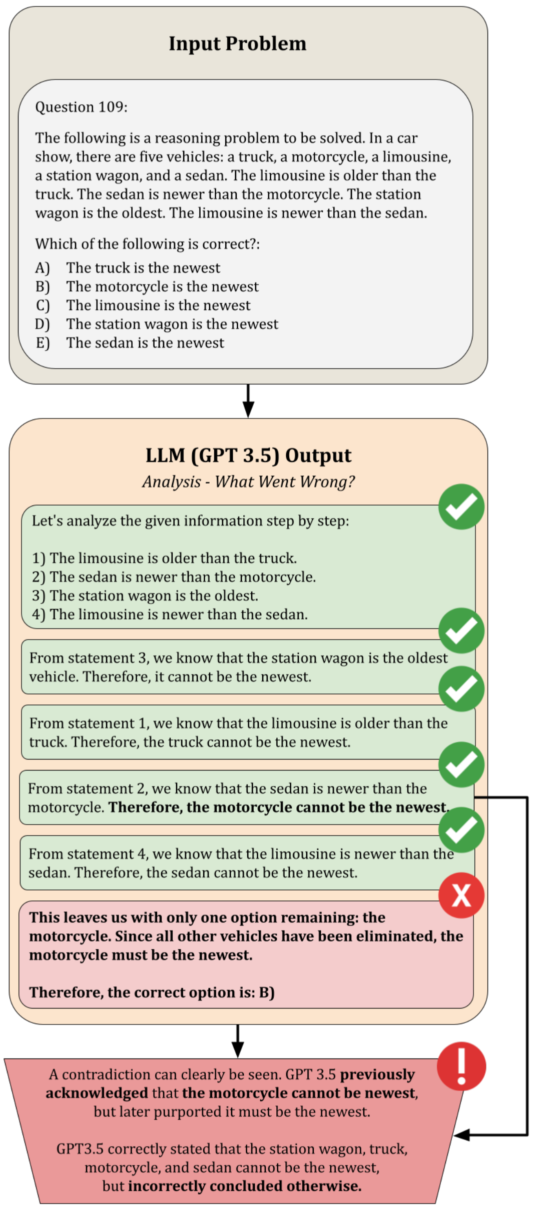
Table 2: Spearman rank correlations of CoT predicted ranks with gold standard ranks.

3.4. Reasoning as Classification

We introduce linguistic-logical primitives with local memory and inference engines, enabling an LLM to reason step-by-step. The LLM parses a problem Q into sentences $S = s_1, s_2, \ldots, s_n$ and predicts the appropriate primitive $\hat{p}_i$ for each s_i, formalized as:

$$\hat{p}_i = \underset{r \in R}{\operatorname{argmax}}\, P(r|s_i, \text{context}) \tag{1}$$

Here, R is the set of primitives. Some primitives write to or read from memory, while others invoke


Input Problem

Question 109:

The following is a reasoning problem to be solved. In a car show, there are five vehicles: a truck, a motorcycle, a limousine, a station wagon, and a sedan. The limousine is older than the truck. The sedan is newer than the motorcycle. The station wagon is the oldest. The limousine is newer than the sedan.

Which of the following is correct?

A) The truck is the newest
B) The motorcycle is the newest
C) The limousine is the newest
D) The station wagon is the newest
E) The sedan is the newest

LLM (GPT 3.5) Output

Analysis - What Went Wrong?

Let's analyze the given information step by step:

1) The limousine is older than the truck.
2) The sedan is newer than the motorcycle.
3) The station wagon is the oldest.
4) The limousine is newer than the sedan.

From statement 3, we know that the station wagon is the oldest vehicle. Therefore, it cannot be the newest.

From statement 1, we know that the limousine is older than the truck. Therefore, the truck cannot be the newest.

From statement 2, we know that the sedan is newer than the motorcycle. **Therefore, the motorcycle cannot be the newest.**

From statement 4, we know that the limousine is newer than the sedan. Therefore, the sedan cannot be the newest.

This leaves us with only one option remaining: the motorcycle. Since all other vehicles have been eliminated, the motorcycle must be the newest.

Therefore, the correct option is: B)

A contradiction can clearly be seen. GPT 3.5 **previously acknowledged** that **the motorcycle cannot be newest**, but later purported it must be the newest.

GPT3.5 correctly stated that the station wagon, truck, motorcycle, and sedan cannot be the newest, but **incorrectly concluded otherwise.**

Figure 2: **Working Example of failure modes of Chain-of-Thought.** A problem from BIG-bench Hard is inputted to GPT-3.5, and the LLM is instructed to utilize CoT reasoning. Each section of GPT-3.5 output is analyzed. The first five sections of GPT-3.5's CoT reasoning are consistent with the input paragraph. However, the model contradicts itself by stating that the motorcycle must be the newest after previously stating it cannot be the newest. Hence, it solves the problem incorrectly.

logic engines for reasoning. This yields explainable solutions. See Figure 3 for a block diagram.

3.4.1. Linguistic-Logical Primitives

Linguistic-logical primitives (LLPs) are essential elements in the reasoning approach presented.

A primitive can be mathematically formulated as a function that takes a sentence s_i and context C as inputs and returns an action A and potentially a change in memory state M:

$$p_i(s_i, C) \rightarrow (A, M) \tag{2}$$

where:

- s_i: The sentence being processed.
- C: The context, including previous sentences and chosen primitives.
- A: The action to be taken, such as writing to memory or invoking a logical solving process.
- M: The change in memory state, if applicable.

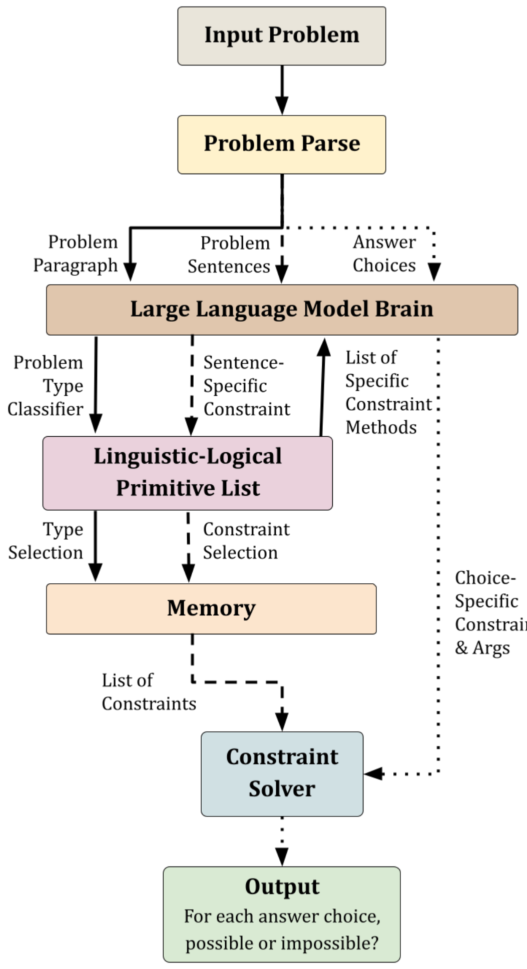

Input Problem
Problem Parse
Problem Paragraph
Problem Sentences
Answer Choices
Large Language Model Brain
Problem Type Classifier
Sentence-Specific Constraint
List of Specific Constraint Methods
Linguistic-Logical Primitive List
Type Selection
Constraint Selection
Choice-Specific Constraint & Args
Memory
List of Constraints
Constraint Solver
Output
For each answer choice, possible or impossible?

Figure 3: **High-level diagram of the reasoning-classification conversion.** Given an input problem, the *Problem Parse* stage splits it into: problem paragraph, sentences, and answer choices. The *LLM* classifies the problem type, enabling *Memory* to fetch relevant constraint methods. Constraints are then extracted and stored in *Memory*. Finally, *Constraint Solver* validates answer choices based on these constraints.

The reasoning process iterates over the sequence of sentences, and for each sentence, the LLM predicts the appropriate primitive to be invoked. This primitive then performs its corresponding action, affecting the memory state and potentially moving the solution process forward.

3.4.2. Computational versus Linguistic Hardness

We distinguish two facets of "hardness" in problem-solving, namely computational complexity and linguistic complexity.

- **Computational Complexity:** We denote the computational complexity of a problem of size n as $C_c(n)$, and it is usually classified within known complexity classes such as P, NP, and EXPTIME.
- **Linguistic Complexity:** We model linguistic hardness as the entropy of the distribution of interpretations I over a problem sentence S of length n. The linguistic complexity $C_l(n)$ is given by:

  $$C_l(n) = -\sum_I P(I|S)\log P(I|S) \tag{3}$$

  This quantifies the uncertainty or ambiguity in predicting the correct interpretation of a sentence.

Note that $C_c(n)$ and $C_l(n)$ are mutually independent and can lead to contrasting levels of hardness in computational and linguistic terms. In practice, $P(I|S)$ may be empirically determined using human annotations, aligned with approaches in areas like word sense disambiguation.

However, we clarify that for our dataset, one can glean computational complexity from the number of objects in the task, and the linguistic complexity from the length (in characters) of the problem.

3.4.3. Example of Computational Complexity versus Linguistic Complexity

- **Computational complexity**: The inherent computational or asymptotic difficulty of solving a problem, e.g., $5 + 5$.
- **Linguistic complexity**: Complexity in language interpretation, even for a computationally simple problem. For example:

  "Add the number of fingers on a hand to the number of toes on a foot."

  This describes $5 + 5$, but the phrasing introduces ambiguity, making the problem linguistically complex, though computationally simple.

3.5. Spatial Reasoning Primitives

In the context of spatial reasoning, we define a set of primitives that allow for reasoning about the relative positions of objects within a constraint satisfaction problem (CSP). Let $\mathcal{O} = \{o_1, o_2, \ldots, o_k\}$ be the set of k objects. The primitives are as follows:

- `create_problem`(k): Defines a CSP with k objects. Formally, this primitive initializes the problem space:

  $$\text{create\_problem}(k) \Rightarrow \mathcal{O} = \{o_1, o_2, \ldots, o_k\} \tag{4}$$

- `add_left`(o_i, o_j): Adds a constraint that object o_i is to the left of object o_j:

  $$\text{add\_left}(o_i, o_j) \Rightarrow \text{position}(o_i) < \text{position}(o_j) \tag{5}$$

- `add_right`(o_i, o_j): Adds a constraint that object o_i is to the right of object o_j:

  $$\text{add\_right}(o_i, o_j) \Rightarrow \text{position}(o_i) > \text{position}(o_j) \tag{6}$$

- `add_position`(o_i, p): Adds a constraint that object o_i is at exact position p:

  $$\text{add\_position}(o_i, p) \Rightarrow \text{position}(o_i) = p \tag{7}$$

These primitives allow for building and solving spatial reasoning problems involving the relative positions of objects.

3.5.1. Properties of Primitives

- **Distinctness**: For a set of functions $\mathcal{F}$, distinctness is defined as:

  $$f_i \neq f_j \quad \forall\, i \neq j,\, i, j \in \{1, \ldots, |\mathcal{F}|\} \tag{8}$$

- **Expressive Power**: A function $f \in \mathcal{F}$ is expressive if there exists a mapping M such that:

  $$M(f, \mathcal{C}) = \mathcal{S} \tag{9}$$

  where $\mathcal{C}$ represents the constraints and $\mathcal{S}$ represents the solvable form.

- **Completeness**: The set of classes $\mathcal{C}$ is complete if:

  $$\bigcup_{c \in \mathcal{C}} c = \mathcal{A} \tag{10}$$

  where $\mathcal{A}$ is the set of all required actions.

- **Minimal**: A set of functions $\mathcal{F}$ is minimal if:

  $$\Pi(\mathcal{F}) > \Pi(\mathcal{F} \setminus \{f_i\}) \quad \forall\, f_i \in \mathcal{F} \tag{11}$$

  where $\Pi(\cdot)$ denotes the performance of the set of functions.

3.5.2. Generating Primitives

Generating primitives for arbitrary logical problems involves analyzing the fundamental logical structures and operations within the domain. A systematic approach might include identifying basic logical components (such as conjunctions, disjunctions, negations), defining corresponding functions or relations, and encapsulating these elements as primitives. The resulting set of primitives must be expressive enough to represent any logical problem within the domain, ensuring completeness and non-redundancy.

3.6. Algorithms

We now discuss our main method in Algorithm 1, as well as its variants.

Algorithm 1 Solving Reasoning Problems via Classification

```
Require: Problem statement $S = \{s_1, s_2, \ldots, s_n\}$
Require: Set of linguistic-logical primitives $P$
 1: Initialize memory storage $M$
 2: for $i = 1$ to $n$ do
 3:    Parse sentence $s_i$
 4:    Predict primitive $\hat{p}_i = \underset{p \in P}{\mathrm{argmax}}\, P(p|s_i, \text{ctxt})$
 5:    if primitive writes to memory then
 6:       Write to memory $M$
 7:    else if primitive invokes logic engine then
 8:       Execute solution process using $M$
 9:    end if
10: end for
11: return Solution obtained from memory $M$
```

3.6.1. Complex Primitives Classification

Establishing conventions is important in reasoning problems. As an example, the statement "A is to the right of B" can be classified as either of the primitives right(A, B) or left(B, A). To fix conventions, we introduce complex primitives. Let $\mathcal{F}(A, B)$ represent a complex primitive, defined as add_to_left_or_cheapest_or_newest(A, B).

3.6.2. Simple Primitives Classification

Complex primitives may confuse models. To mitigate this, we introduce an additional classifier layer to determine the type of reasoning problem, such as age, competition, price, or left-right, and use this information to select a specific subset of simpler primitives.

Given a reasoning problem R, the first layer of classification determines the problem type T, where $T \in \{\text{age}, \text{competition}, \text{price}, \text{left-right}\}$. The second layer selects a subset of simpler primitives P_T corresponding to T.

Algorithm 2 Algorithm for selecting primitives based on problem type

```
 1: Determine problem type $T$ for reasoning problem $R$
 2: if $T =$ age then
 3:    $P_T = \{\text{add\_older}, \text{add\_newer}, \ldots\}$
 4: else if $T =$ competition then
 5:    $P_T = \{\text{add\_winner}, \text{add\_loser}, \ldots\}$
 6: else if $T =$ price then
 7:    $P_T = \{\text{add\_cheaper}, \text{add\_expensive}, \ldots\}$
 8: else if $T =$ left-right then
 9:    $P_T = \{\text{add\_left}, \text{add\_right}, \ldots\}$
10: end if
11: Use $P_T$ to reason about $R$
```

Refer to Algorithm 2 for details on selecting simpler primitives specific to the problem type.

3.6.3. Iterative Refinement

In the case of simple primitives, the model may fail to invoke the correct primitive or encounter other issues. To tackle this, we employ an iterative-refinement process exploiting the stochastic nature of LLMs. We iteratively fine-tune the model's choices using edited prompts to improve the success rate. Let R be a reasoning problem, and P_T be the set of primitives associated with the determined problem type T. The iterative-refinement process can be defined as follows:

Algorithm 3 Iterative-Refinement for Primitive Selection

```
 1: Initialize attempts counter $k = 0$
 2: repeat
 3:    Edit prompt for problem $R$, producing $R'$
 4:    Call the correct primitive from $P_T$ using $R'$
 5:    Evaluate success of call; if successful, exit loop
 6:    Increment $k$ by 1
 7: until success or $k \geq K$, where $K$ is the maximum number of attempts
 8: if success then
 9:    Continue to next step in reasoning
10: else
11:    Handle failure, e.g., log error or alert user
12: end if
```

The process involves iterative editing of prompts, attempting to call the correct primitive, and evaluating the success of each call. The loop continues until either success is achieved or the maximum number of attempts K is reached. This approach introduces an element of stochasticity, allowing for exploration of different prompt variations and increasing the likelihood of success. Refer to Algorithm 3 for a detailed procedure. We choose $k = 5$.

4. Evaluation

In this section we discuss the dataset and experimental protocol. Then we go over the various methods we benchmark (GPT, GPT+CoT, GPT+CoT+SC, Simple Primitives, Iterative Refinement) and perform analyses across problem types.

4.1. Experimental Protocol

We evaluate our proposed methods against three established baselines [2] : GPT-3.5 base, Chain-of-Thought (CoT) (Wei et al., 2022), and Chain-of-Thought with Self Consistency (SC) (Wang et al., 2022). Additionally, we introduce two novel models, namely, classification with simple primitives and iterative refinement. We operate on the randomized dataset we sample above.

In our experiments, we employ the function-calling capabilities provided by OpenAI's large language models (LLMs) (OpenAI, 2023a) in order to convert reasoning problems to classification tasks. Specifically, we use `gpt-3.5-turbo-0613`.

4.2. Complex versus Simple Primitives

As introduced in Section 3.6.1 and Section 3.6.2, we define simple and complex primitives. We investigate the impact of primitive complexity on model performance. Table 3 reveals that simple primitives consistently outperform complex ones, particularly as task complexity rises (from 3 to 7 objects).

Table 4 further underscores the advantage of simple primitives in all but the "age" problem type. In "competition," the use of simple primitives nearly doubles performance.

Finally, Table 5 indicates variable performance for complex primitives. While effective in "age" and "competition" with 3 objects, their performance deteriorates with an increased number of objects.

From this comparison we see that simple primitives are better at defining actions that build the basic logic block for reasoning. So we use simple primitives in our final approach as shown in Section 4.3.

[2]We would like to note that we initially sought to benchmark our results with LogicLM (Pan et al., 2023), but unfortunately, their method returned errors for all data points in our specific test cases and resorted to their default failsafe, CoT. As a result, we resorted to using the CoT method, the results of which are included in the table. We express our gratitude to the LogicLM team for their work, and the discrepancy may be due to specificities in our dataset or experimental setup.

Model	BBH3	BBH5	BBH7
Complex Primitives	51%	45%	37%
Simple Primitives	**85**%	**69**%	**57**%

Table 3: Accuracy of complex primitives versus simple primitives across datasets.

Problem Type	Complex Primitives	Simple Primitives
leftright	39.23%	**76.92**%
competition	45.28%	**88.68**%
price	26.32%	**57.89**%
age	**71.67**%	51.67%

Table 4: Accuracy of complex versus simple primitives across different problem types.

Problem Type	3 Objects	5 Objects	7 Objects
leftright	33.33%	**52.78%**	34.78%
competition	**83.33%**	16.67%	45.45%
price	31.25%	**31.58%**	18.18%
age	**83.33%**	76.19%	57.14%

Table 5: Accuracy of complex primitives based on problem type and number of objects.

4.3. Results

We present our model's performance in Table 6. We then present aggregate accuracy scores of each method with different problem types in Table 7. We break this down further by problem type and size and present more granular performance in Table 8. We also depict correlations between each method in Table 9.

Table 6 shows that "Iterative Refinement" outperforms other methods across datasets (BBH3, BBH5, BBH7), leading by 74.66%. Other methods degrade with more objects, whereas "Simple Primitives" and "Iterative Refinement" remain robust.

In Table 7, "Iterative Refinement" excels in all problem types, notably reaching 94.34% in "competition" problems. "Simple Primitives" also surpass base models in all but one case.

Table 8 reveals that "Simple Primitives" excel in specific problems with fewer objects, while "Iterative Refinement" is consistently strong. Adding CoT and SC to GPT is shown to not always improve performance.

Lastly, Table 9 indicates moderate correlation between CoT methods, with stronger correlation between "Simple Primitives" and "Iterative Refinement." Base GPT shows low correlation with other methods, suggesting distinct prediction patterns.

Dataset	Base GPT	GPT+CoT	GPT+CoT+SC	Simple Primitives	Iterative Refinement
BBH3	57%	57%	52%	85%	**86%**
BBH5	42%	34%	34%	69%	**74%**
BBH7	32%	32%	36%	57%	**64%**
Overall	43.66%	41%	40.66%	70.33%	**74.66%**

Table 6: Accuracy of different methods on various datasets.

Problem Type	Base GPT	GPT+CoT	GPT+CoT+SC	Simple Primitives	Iterative Refinement
leftright	36.15%	33.85%	37.69%	76.92%	**80.77%**
competition	43.40%	43.40%	35.85%	88.68%	**94.34%**
price	59.65%	56.14%	50.88%	57.89%	**64.91%**
age	45.00%	40.00%	41.67%	51.67%	**53.33%**

Table 7: Accuracy across different problem types.

Method	Problem Type	3 Objects	5 Objects	7 Objects
Baseline 1 (GPT)	leftright	54.17%	25.00%	26.09%
Baseline 1 (GPT)	competition	50.00%	41.67%	36.36%
Baseline 1 (GPT)	price	75.00%	68.42%	40.91%
Baseline 1 (GPT)	age	55.56%	47.62%	33.33%
Baseline 2 (GPT + CoT)	leftright	54.17%	16.67%	26.09%
Baseline 2 (GPT + CoT)	competition	50.00%	45.83%	27.27%
Baseline 2 (GPT + CoT)	price	68.75%	47.37%	54.55%
Baseline 2 (GPT + CoT)	age	61.11%	38.10%	23.81%
Baseline 3 (GPT + CoT + SC)	leftright	50.00%	30.56%	30.43%
Baseline 3 (GPT + CoT + SC)	competition	50.00%	25.00%	36.36%
Baseline 3 (GPT + CoT + SC)	price	62.50%	52.63%	40.91%
Baseline 3 (GPT + CoT + SC)	age	50.00%	33.33%	42.86%
Simple Primitives	leftright	**93.75%**	**83.33%**	54.35%
Simple Primitives	competition	**100.00%**	79.17%	**90.91%**
Simple Primitives	price	62.50%	52.63%	59.09%
Simple Primitives	age	**66.67%**	47.62%	**42.86%**
Iterative Refinement	leftright	91.67%	**83.33%**	**67.39%**
Iterative Refinement	competition	**100.00%**	**91.67%**	**90.91%**
Iterative Refinement	price	**75.00%**	57.89%	63.64%
Iterative Refinement	age	**66.67%**	**52.38%**	**42.86%**

Table 8: Accuracy based on problem type and number of objects.

Method	Base GPT	GPT+CoT	GPT+CoT+SC	Simple Primitives	Iterative Refinement
Base GPT	1.00	0.46	0.46	0.32	0.24
GPT+CoT	0.46	1.00	0.48	0.21	0.22
GPT+CoT+SC	0.46	0.48	1.00	0.21	0.22
Simple Primitives	0.32	0.21	0.21	1.00	0.68
Iterative Refinement	0.24	0.22	0.22	0.68	1.00

Table 9: Correlation matrix between predictions of different methods.

5. Conclusions

In this paper, we examined various methods to solve constraint satisfaction reasoning problems with LLMs across varying problem types and sizes. We proposed a novel method for reasoning via classification into simple and complex primitives. Our findings indicate that simple primitives often outperform complex ones, with the iterative refinement method consistently showing superior performance across the board by almost 40%.

Our study reveals a decoupling between computational and linguistic complexities in problem-solving tasks with large language models (LLMs). We find that as LLMs are optimized, linguistic operation complexity decreases, indicating an inversely proportional relationship and supporting the development of more efficient LLM architectures.

6. Bibliographical References

K. Arkoudas. 2023. Gpt-4 can't reason. *ArXiV*.

T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. 2020. Language models are few-shot learners. *ArXiV*.

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. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *Microsoft Blog*.

W. Chen, X. Ma, X. Wang, and W. W. Cohen. 2022. Program of Thoughts: Reasoning about Numerical Problems with Language Models and a Program Interpreter. *arXiv preprint arXiv:2211.12588*.

P. Clark, O. Tafjord, and K. Richardson. 2020. Transformers as Soft Reasoners over Language. In *IJCAI Proceedings 2020*.

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. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

G. Franceschelli and M. Musolesi. 2023. On the creativity of large language models. *ArXiV*.

L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig. 2023. Program-Aided Language Models. *arXiv preprint arXiv:23-Luyu-Gao$_b$*.

S. Han, H. Schoelkopf, Y. Zhao, Z. Qi, M. Riddell, L. Benson, L. Sun, E. Zubova, Y. Qiao, M. Burtell, D. Peng, J. Fan, Y. Liu, B. Wong, M. Sailor, A. Ni, L. Nan, J. Kasai, T. Yu, R. Zhang, S. Joty, A. R. Fabbri, W. Kryscinski, X. V. Lin, C. Xiong, and D. Radev. 2022. FOLIO: A Dataset for Natural Language Reasoning with First-Order Logic Annotations. *arXiv preprint arXiv:2209.00840*.

S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu. 2023. Reasoning via Planning: A New Approach to Improve the Reasoning Capabilities of Large Language Models. *arXiv preprint arXiv:2305.14992*.

J. He-Yueya, G. Poesia, R. E. Wang, and N. D. Goodman. 2023. Combining Language Models and Symbolic Solvers for Math Word Problems. *arXiv preprint arXiv:2304.09102*.

J. Huang and K. C. C. Chang. 2023. Towards reasoning in large language models: A survey. *ACL Anthology*.

J. Liu, L. Cui, H. Liu, D. Huang, Y. Wang, and Y. Zhang. 2020. LogiQA: A Challenge Dataset for Machine Reading Comprehension with Logical Reasoning. *arXiv preprint arXiv:2007.08124*.

OpenAI. 2023a. Function calling and other api updates. Accessed: 2023-07-28.

OpenAI. 2023b. Gpt-4 technical report. *ArXiV*.

Liangming Pan, Alon Albalak, Xinyi Wang, and William Yang Wang. 2023. LOGIC-LM: Leveraging Large Language Models for Logical Reasoning. *arXiv preprint arXiv:2305.12295*.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. 2018. Improving language understanding by generative pre-training. *ArXiV*.

A. Rajasekharan, Y. Zeng, P. Padalkar, and G. Gupta. 2023. Reliable natural language understanding with large language models and answer set programming. *ArXiV*.

R. Schaeffer, B. Miranda, and S. Koyejo. 2023. Are emergent abilities of large language models a mirage? *ArXiV*.

A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, A. Ray, A. Warstadt, A. W. Kocurek, and et al. 2022. Beyond the Imitation Game benchmark (BIG-bench): Quantifying and Extrapolating the Capabilities of Language Models. In *Stanford University CICL Papers*.

G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *ArXiV*.

X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. 2022. Self-Consistency: A General-Purpose Decoding Strategy for Improving the Reasoning Capabilities of Large Language Models. *arXiv preprint arXiv:2203.11171*.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. 2020. Emergent abilities of large language models. *ArXiV*.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. 2022. Improving the Reasoning Capabilities of Large Language Models with Chain-of-Thought Prompting. *arXiv preprint arXiv:2201.11903*.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. 2023. Tree of Thoughts: A New Inference Framework for Large Language Models. *arXiv preprint arXiv:2305.10601*.

W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, Y. Du, C. Yang, Y. Chen, Z. Chen, J. Jiang, R. Ren, Y. Li, X. Tang, Z. Liu, P. Liu, J. Y. Nie, and J. R. Wen. 2023. A survey of large language models. *ArXiv*.

W. Zhu, H. Liu, Q. Dong, J. Xu, S. Huang, L. Kong, J. Chen, and L. Li. 2023. Multilingual machine translation with large language models: Empirical results and analysis. *ArXiV*.

7. Language Resource References

Mirac Suzgun and Nathan Scales and Nathanael Schärli and Sebastian Gehrmann and Yi Tay and Hyung Won Chung and Aakanksha Chowdhery and Quoc V. Le and Ed H. Chi and Denny Zhou and Jason Wei. 2022. *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. PID https://arxiv.org/abs/2210.09261.