Faithful Chain-of-Thought Reasoning

Qing Lyu [*1] Shreya Havaldar [*1] Adam Stein [*1] Li Zhang [1] Delip Rao [1]
Eric Wong [1] Marianna Apidianaki [1] Chris Callison-Burch [1]

Abstract

While Chain-of-Thought (CoT) prompting boosts Language Models' (LM) performance on a gamut of complex reasoning tasks, the generated reasoning chain does not necessarily reflect how the model arrives at the answer (aka. *faithfulness*). We propose **Faithful CoT**, a faithful-by-construction framework that decomposes a reasoning task into two stages: **Translation** (Natural Language query → symbolic reasoning chain) and **Problem Solving** (reasoning chain → answer), using an LM and a deterministic solver respectively. We demonstrate the efficacy of our approach on 10 reasoning datasets from 4 diverse domains. It outperforms traditional CoT prompting on 9 out of the 10 datasets, with an average accuracy gain of 4.4 on Math Word Problems, 1.9 on Planning, 4.0 on Multi-hop Question Answering (QA), and 18.1 on Logical Inference, under greedy decoding. Together with self-consistency decoding, we achieve new state-of-the-art few-shot performance on 7 out of the 10 datasets, showing a strong synergy between faithfulness and accuracy.[1]

1. Introduction

Complex reasoning tasks, such as commonsense reasoning and math reasoning, have long been the Achilles heel of LMs (Bengio, 2019), until a recent line of work on Chain-of-Thought (CoT) reasoning (Wei et al., 2022; Wang et al., 2022, i.a.) brought striking performance gains, by prompting an LM to generate a reasoning chain along with the answer, given only a few in-context exemplars.

In addition to performance improvement, CoT is also claimed to "provide an interpretable window into the behavior of the model" (Wei et al., 2022). However, **it lacks one fundamental property of explanation, *faithfulness***, meaning "an explanation (i.e., the reasoning chain) should accurately represent the reasoning process behind the model's prediction" (i.e., how the model arrives at the final answer) (Jacovi & Goldberg, 2020, i.a.). In existing CoT methods, the final answer does not necessarily follow from the previously generated reasoning chain, so there is no guarantee on faithfulness. Figure 1 exemplifies such an unfaithful generation from our inspection of the model output from Wei et al. (2022) on GSM8K: the answer "0" is not even mentioned in the reasoning chain. This, along with more examples from Appendix B.1, illustrates that existing CoT methods do not provide true interpretability of how the model predicts the answer.

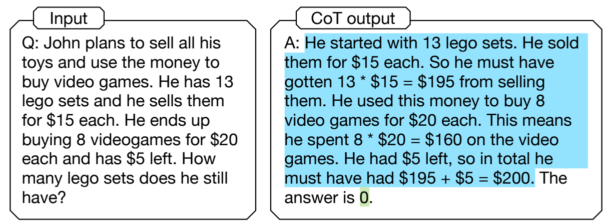

Figure 1. An example of *unfaithful* output from CoT prompting (Wei et al., 2022) on GSM8K. The answer (green) does not follow from the reasoning chain (blue).

The lack of faithfulness in CoT can be dangerous in high-stake applications because it may mislead people into believing that the model is inherently interpretable, while there is indeed no causal relationship between the reasoning chain and the answer. Even worse, when an *unfaithful* explanation looks *plausible* (i.e., convincing to humans) (Jacovi & Goldberg, 2020), this makes it easier for people (e.g., legal practitioners) to over-trust the model (e.g., a recidivism predictor) even if it has implicit biases (e.g., against racial minorities) (Pruthi et al., 2020; Slack et al., 2020).

To address this concern, we propose **Faithful CoT**, a faithful-by-construction framework where the answer is the result of deterministically executing the reasoning chain. Specifically, we break down a complex reasoning task into two sequential stages: **Translation** and **Problem Solving** (Figure 2). During Translation, an LM translates a Natural Language (NL) query into a reasoning chain, which interleaves NL and a task-dependent Symbolic Language (SL), such as Python, Datalog, or Planning Domain Definition Language (PDDL), as illustrated in Figure 3. Then, in the

[*]Equal contribution [1]Department of Computer and Information Science, University of Pennsylvania, Philadelphia, USA.

[1]Code and data to be released at https://github.com/veronica320/Faithful-COT.

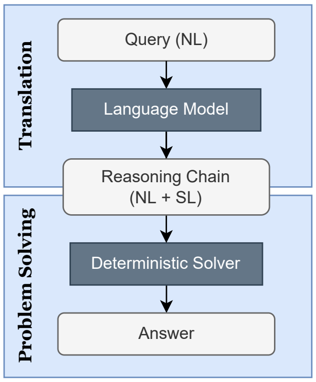

Figure 2. An overview of our 2-stage pipeline, consisting of **Translation**, where an LM translates a query (in NL/Natural Language) into a reasoning chain (which interleaves NL and SL/Symbolic Language), and **Problem Solving**, where an external solver executes the reasoning chain to derive the answer, thus ensuring faithfulness.

Problem Solving stage, the reasoning chain is executed by a deterministic solver, e.g., a Python/Datalog interpreter, or a PDDL planner, to derive the answer.

We evaluate our approach on 10 reasoning datasets from 4 diverse domains: Math Word Problems (MWP), Planning, Multi-hop QA, and Logical Inference. We compare it with standard prompting (Brown et al., 2020) and CoT prompting, using the same underlying LM (Codex (Chen et al., 2021)) and the same decoding strategies (greedy and self-consistent decoding (Wang et al., 2022)). Results show that on a majority of the datasets, our approach outperforms both baselines, with an average accuracy gain over CoT of 4.4 on MWP, 1.9 on Planning, 4.0 on Multi-hop QA, and 18.1 on Logical Inference, using greedy decoding. Together with self-consistent decoding, we achieve new state-of-the-art few-shot performance on 7 out of the 10 datasets.

Our key contributions are as follows:
(a) We propose Faithful CoT, a faithful-by-construction prompting framework, which decomposes reasoning into Translation and Problem Solving. The reasoning chain interleaves user-understandable natural language comments and executable symbolic language programs, thus providing interpretability of how the model arrives at the answer.
(b) Our approach is generalizable to multiple domains beyond arithmetic reasoning and simple symbolic reasoning, thanks to its flexible integration with any choice of SL and external solver. We set the new SOTA performance on 7 out of the 10 reasoning datasets, showing a strong synergy between faithfulness and accuracy.
(c) We provide an extensive analysis of the strengths and weaknesses of our method, showing its robustness to the choice of exemplars as well as the critical role of the solver.

2. Related Work

Faithfulness. In interpretability, *faithfulness* (also called *fidelity* or *reliability*) means that an explanation should "accurately represent the reasoning process behind the model's prediction", which is a fundamental requirement of an explanation (Harrington et al., 1985; Ribeiro et al., 2016; Gilpin et al., 2018; Jacovi & Goldberg, 2020).[2] It should be contrasted with *plausibility* (a.k.a. *persuasiveness* or *understandability*), which refers to "how convincing an explanation is to humans" (Herman, 2019; Jacovi & Goldberg, 2020). In the context of CoT prompting, a faithful reasoning chain needs to accurately reflect how the model arrives at the final answer, whereas a plausible reasoning chain is one that looks reasonable and coherent to humans. The vanilla CoT (Wei et al., 2022) generates the reasoning chain in pure NL, which may often look plausible; nevertheless, the final answer does not need to causally follow from the reasoning chain, thus not guaranteeing faithfulness.

Chain-of-Thought-style prompting. In CoT-style prompting, given a complex question Q, an LM is prompted to generate a reasoning chain C along with the final answer A. Specifically, the prompt consists of a few examples of (Q, C, A) triples, called in-context exemplars. This allows pre-trained LMs (e.g., GPT-3 (Brown et al., 2020)) to solve unseen questions with much higher accuracy than standard prompting, where the exemplars do not contain the reasoning chain C.

We create a taxonomy of existing CoT-style prompting methods into three types: all-at-once, ensemble-based, and modularized. **All-at-once** prompting means that the LM produces C and A as one continuous string, without any dependencies or constraints in between. Scratchpad (Nye et al., 2021), the vanilla CoT (Wei et al., 2022), and "Let's think step by step" (Kojima et al., 2022), are all examples of this kind. **Ensemble-based** prompting is designed to overcome the local optimality issue of the one-shot generation in previous methods by sampling multiple (C, A) pairs and choosing the best answer via strategies like majority voting. Examples include Self-Consistent CoT (Wang et al., 2022), Minerva (Lewkowycz et al., 2022), and DIVERSE (Li et al., 2022), which differ mainly in the voting granularity and the underlying LM. **Modularized** methods break down Q into subproblems and then conquer them individually (Jung et al., 2022; Qian et al., 2022, i.a.). In particular, Least-

[2]Note that this differs from the notion of faithfulness in the Natural Language Generation (NLG) literature, primarily in what constitutes the **ground truth**. In interpretability, we talk about the faithfulness of an explanation w.r.t. **the model's underlying reasoning mechanism** – the ground truth is usually opaque. In NLG, we talk about the faithfulness of the generated text (e.g., a translated sentence, or a summary) w.r.t. some **explicit source** (e.g., the source sentence, or the full document) – the ground truth is transparent.

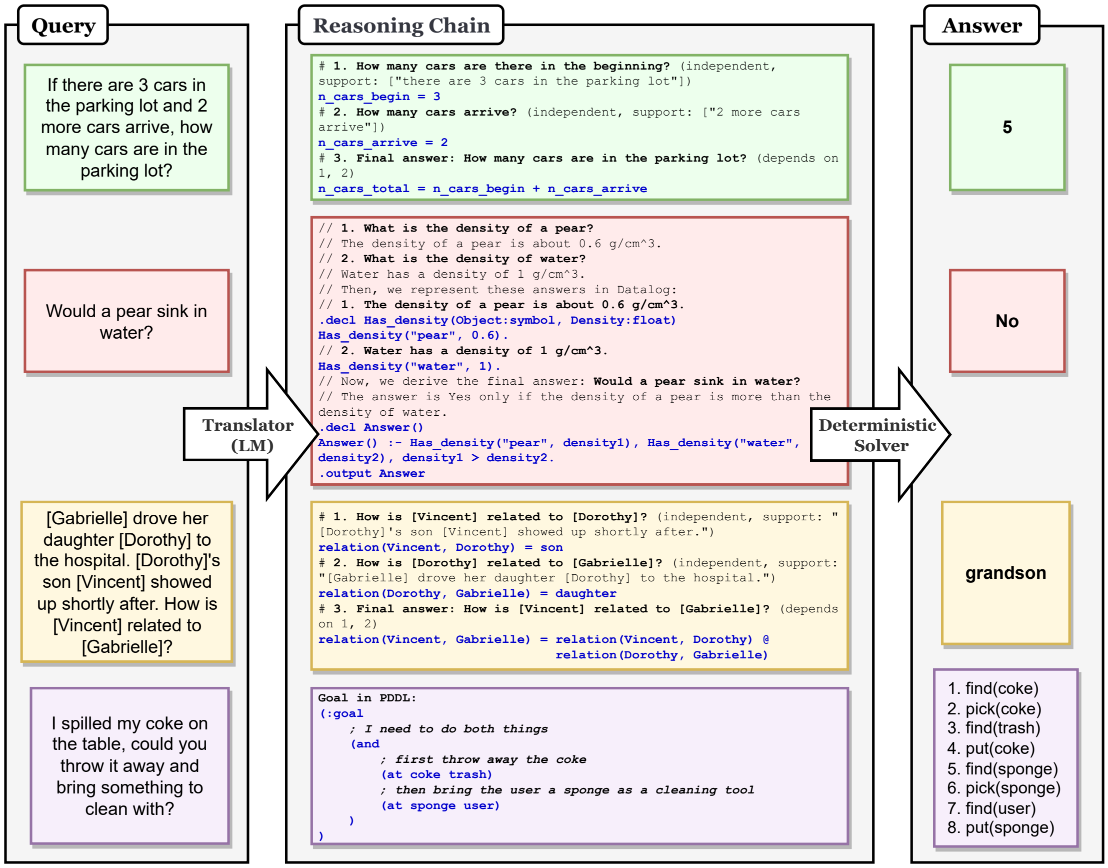

Figure 3. Examples from each task (Math Word Problems, Multi-hop QA, Planning, Logical Inference) showing our 2-stage Translation and Problem Solving pipeline.

to-Most prompting (Zhou et al., 2022) uses an LM to first reduce the question to subquestions and then sequentially answer them. However, the reasoning chain is entirely in NL, so there is still no faithfulness guarantee that the answer follows from it.

We note that our work is concurrent with Chen et al. (2022) and Gao et al. (2022), both generating the reasoning chain in Python code and calling a Python interpreter to derive the answer. While we do not compare with them empirically for this reason, we do want to highlight the following differences: (a) We demonstrate the generalizability of our approach to multiple symbolic languages beyond Python and multiple domains beyond arithmetic reasoning and simple symbolic reasoning. (b) In particular, we innovatively recast a diverse set of realistic tasks (Planning, Multi-hop QA, and Logical Inference) into a symbolic representation, which allows us to tackle them with a single framework. (c) Our reasoning chain interleaves NL and SL in a structured fashion, which allows the user to better understand and potentially interact with the model.

3. Method

Our method, **Faithful CoT**, is a 2-stage pipeline, as seen in Figure 2. Like previous CoT-style work, our prompt consists of (Q, C, A) triples. Notable differences lie in our unique interleaving of NL (natural language) and SL (symbolic language) in C, as well as the way we derive the final answer A.

In the **Translation** stage, given a complex query Q in NL, we prompt an LM to translate it into a reasoning chain C, which interleaves NL and a task-specific SL (e.g., Python, Datalog, or PDDL).[3] In the **Problem Solving** stage, we call a deterministic external solver, e.g., a Python interpreter, a Datalog executor, or PDDL planner, depending on the task, to obtain the answer A from the reasoning chain C. As shown in Figure 3, we define C_{NL} to be the NL component (black) and C_{SL} to be the SL component (blue) in C. Though we separate the two components notationally, they are interleaved in the generation. Using this approach, C is guaranteed to be a faithful model explanation, since our final A is the result of deterministically executing C_{SL}. Moreover, C_{NL} allows the user to better understand the

[3]Our prompts can be found in the Supplementary Materials.

reasoning process.[4]

We apply this method to 4 types of complex reasoning tasks: MWP, Multi-hop QA, Planning, and Logical Inference. Next, we will illustrate how our method works for each of them, with examples from Figure 3.

3.1. Math Word Problems (MWP)

Given a grade-school math question Q written in NL ("If there are 3 cars in the parking lot and 2 more cars arrive, how many cars are in the parking lot?", shown in green in Figure 3), we want to obtain A as a real-valued number (5). In the Translation stage, we prompt the LM to take in Q and generate a reasoning chain C, which interleaves C_{NL} and C_{SL}. Specifically, the C_{NL} component consists of three types of information:

(a) **Subquestions**: Q is broken down into multiple smaller-scale subquestions, e.g., "1. how many cars are there in the beginning?", "2. how many cars arrive?", and "3. how many cars are in the parking lot?".

(b) **Dependency Graph**: Each subquestion can either be answered directly via context (subquestions 1 and 2 are "independent") or rely on answers to previous subquestions (subquestion 3 "depends on 1 and 2").

(c) **Rationales**: Each subquestion is accompanied with rationale(s) to support the answer (the "support" field). The rationales can be either a subset of the original context ("2 more cars arrive") or any external knowledge ("there are 7 days in a week") relevant to the subquestion.

Each subquestion and its corresponding dependencies and rationales inform the subsequent generation of C_{SL}. In our example in Figure 3, C_{SL} consists of Python code generated to answer each subquestion in C_{NL}. During the Problem Solving stage, we execute C_{SL} using our solver, a Python interpreter, to derive A (5 cars in the end).

3.2. Multi-hop QA

Given a complex question Q that involves multiple steps of reasoning (e.g., "Would a pear sink in water?", shown in red in Figure 3), we want to obtain the answer A as a Boolean value or string value variable. Similar to our MWP task formulation, C interleaves C_{NL} (NL comments), and C_{SL} (symbolic program). Depending on the nature of the task, the format of the reasoning chain C is slightly different: for some datasets, the LM first generates all subquestions and their answers in NL, and then represents these answers as SL to derive A (see Figure 3); for others, the LM interleaves the NL subquestions and the SL program, similar to the case of MWP (see Table 11 and Table 12 for examples).

In terms of SL, we use both Python and Datalog, also depending on the dataset. As Multi-hop QA problems involve multi-step reasoning to solve, C_{SL} often utilizes Boolean algebra and string comparisons (in Python) along with relation definitions and logic programming (in Datalog). We use their corresponding interpreter as our deterministic solver to execute C_{SL} and obtain A.

In the example from Figure 3, the LM first generates the subquestions, "1. What is the density of a pear?" and "2. What is the density of water?", which are individually answered in NL. The answers ("Water has a density of $1g/cm^3$") are converted to Datalog statements (`Has_density("water", 1)`), which are then combined to formalize the truth condition of the final answer. Finally, we execute the Datalog program to determine that a pear would **not** sink in water.

3.3. Planning

In a user-robot interaction scenario, given a household task query Q from a user, we want to come up with a plan of actions A that the robot should take in order to accomplish the task. For example, in Figure 3, given user query "I spilled my coke on the table, could you throw it away and bring something to clean with?", a possible plan can be "find(coke), pick(coke), find(trash), put(coke) ...". In the Translation stage, an LM translates Q into C, consisting of C_{NL} (which breaks down Q into subtasks) and C_{SL} (which represents the subtasks as a symbolic `goal` in PDDL[5] — a language to define and solve classical planning problems). Figure 3 shows this translation, with C_{SL} in blue and C_{NL} in black. Finally, we call a PDDL Planner as the deterministic solver to obtain A, a plan to accomplish the goal C_{SL} under the predefined scenario.

3.4. Logical Inference

Given a logical inference problem Q written in NL, we want to obtain A as a string-valued variable. For example, the **CLUTRR** (Sinha et al., 2019) dataset involves inferring the family relationship (e.g., "grandson") between two people from a short story (e.g., "[Gabrielle] drove her daughter [Dorothy] to the hospital. [Dorothy]'s son [Vincent] showed up shortly after. How is [Vincent] related to [Gabrielle]?", shown in yellow in Figure 3). During the Translation stage, we prompt the LM to generate C, consisting of C_{NL} and C_{SL}. Similar to previous tasks, C_{NL} breaks down Q into subquestions ("How is [Vincent] related to [Dorothy]" and "How is [Dorothy] related to [Gabrielle]"), as well as provide input extracts as rationales to support the answer ("[Dorothy]'s son [Vincent] showed up shortly after", etc.). Each subquestion in C_{NL} is answered in C_{SL} via a logical expression representing the relation between

[4]While no constraints are enforced between C_{NL} and C_{SL} in our main experiments, we analyze this in Section C.3.

[5]https://en.wikipedia.org/wiki/Planning_Domain_Definition_Language. A `goal` is a special construct in PDDL.

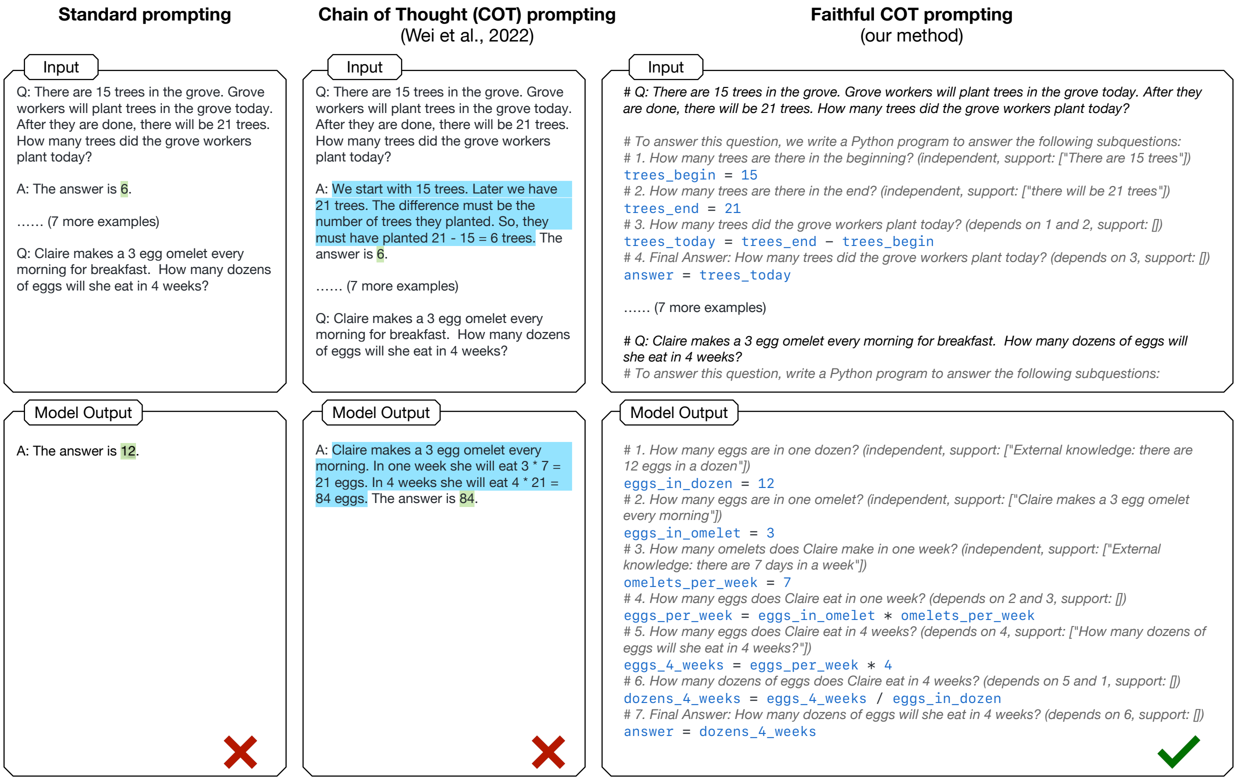

Figure 4. The prompt for GSM8K (a Math Word Problems dataset) from standard, CoT (Wei et al., 2022), and Faithful CoT prompting (ours). The ground-truth answer is 7, and only our method correctly computes the answer.

the mentioned entities, for example, `relation(Vincent, Dorothy)=son` denotes that Vincent is Dorothy's son. In the Problem Solving stage, our solver is a simple logical inference engine that relies on a set of transitivity rules provided by (Anonymous, 2022) among possible family relationships, e.g., `son@daughter=grandson` (the son of one's daughter is one's grandson). Our solver recursively applies these rules on C_{SL} to derive A, and determine that Vincent is Gabrielle's grandson.

4. Experimental setup

4.1. Datasets

Here, we summarize the evaluation datasets used for each domain. We select the same number (6 to 10, depending on the task) of exemplars as in Wei et al. (2022) to form our few-shot prompt, which can be found in the Supplementary Materials. Unless otherwise stated, we use the official splits: training set for exemplar selection, validation set for prompt tuning, and test set for evaluation.[6]

Math Word Problems (MWP). We follow Wei et al. (2022) and consider the same five MWP benchmarks: **GSM8K** (Cobbe et al., 2021), **SVAMP** (Patel et al., 2021), **MultiArith** (Roy & Roth, 2015), **ASDiv** (Miao et al., 2020), and **AQuA** (Ling et al., 2017). For all datasets, the input question is phrased in NL. The answer is a string-valued mathematical expression for AQuA, and one or more integer(s) for all other datasets. We use the same 8-shot prompt for all datasets except AQuA.

Multi-hop QA. We consider the three datasets: **StrategyQA** (Geva et al., 2021), a dataset of open-domain questions that require an implicit multi-step strategy to answer, e.g., "Did Aristotle use a laptop?" involves answering "1. When did Aristotle live?", "2. When was the laptop invented?", and "3. Is #2 before #1?"; **Date Understanding** from BIG-bench (BIG-Bench collaboration, 2021), which asks the model to infer a date from a context, by performing computation on relative periods of time; and finally, **Sports Understanding** from BIG-bench, which asks the model to decide whether an artificially constructed statement related to sports is plausible or implausible. Since the latter two datasets do not have a training set, we follow Wei et al. (2022) and select 10 examples from the test set to form the prompt and use the rest for evaluation.

Planning. We use the **SayCan** dataset (Ahn et al., 2022), which assumes a scenario of a robot operating in a kitchen, helping the user with household tasks, e.g., "bring a coke

[6] See Appendix E for dataset statistics, examples, data cleaning method, splits, prompt construction strategy, etc.

Table 1. Accuracy of different prompting methods on 10 reasoning datasets from 4 domains. We compare our method, Faithful CoT, with standard promoting (Brown et al., 2020), CoT prompting (Wei et al., 2022), as well as previously published few-shot SOTA results. The best results within each decoding strategy are in **boldface**, and the new SOTA results across all strategies are <u>underlined</u>.

Method	Math Word Problems					Planning	Multi-hop QA			Logical Inference
	GSM8K	SVAMP	MultiArith	ASDiv	AQuA	SayCan	StrategyQA	Date	Sport	CLUTRR
Few-shot SOTA	78.0	86.8	<u>100.0</u>	<u>87.8</u>	52.0	88.3	<u>81.6</u>	65.3	98.5	-
Greedy Decoding										
Standard	19.7	69.9	44.0	74.0	29.5	85.8	67.1	49.0	71.7	41.1
CoT	63.1	76.4	96.2	78.6	45.3	87.4	**73.2**	59.9	97.9	40.8
Faithful CoT (ours)	**72.1**	**83.5**	**98.8**	**79.9**	**47.2**	**89.3**	63.0	**80.8**	**<u>99.1</u>**	**58.9**
Self-Consistency Decoding										
CoT	78.0	86.8	**100.0**	84.2	52.0	89.3	79.8	63.8	98.0	45.7
Faithful CoT (ours)	**<u>79.8</u>**	**<u>88.9</u>**	99.2	**84.4**	**<u>61.4</u>**	**<u>93.2</u>**	65.2	**<u>85.5</u>**	**99.0**	**<u>71.9</u>**

to the table". There are a number of locations and objects that the robot can interact with. The robot can only perform a fixed set of actions, including `find`, `pick`, and `put`. The task is to map a user query in NL to a plan of predefined actions. Following Wei et al. (2022), we manually write 7 exemplars, since no training set is provided.

Logical inference. We use the **CLUTRR** (Sinha et al., 2019) benchmark described in Section 3.4. The dataset has multiple splits based on the number of intermediate steps K required to reach the answer. We construct the prompt using 8 exemplars with $K \in \{2, 3\}$, and test the models on the remaining examples with K up to 10.

4.2. Evaluation Metrics

We evaluate the model performance with the accuracy of the final answer. Following previous work (Wei et al., 2022; Wang et al., 2022; Chen et al., 2022), for all MWP datasets (except AQuA) where the answer contains integer(s), a correct answer is defined as the exact match between the prediction and the ground truth both rounded up to the nearest integer (with the `math.ceil()` function in Python); for StrategyQA and Sports Understanding where the answer is a Boolean value, it is defined as the exact match between the prediction and the ground truth both evaluated as a Boolean variable; for SayCan, the generated plan is considered correct if it is among the ground truth plans; for all other datasets, we rely on the exact match between the prediction string and the ground truth string.

4.3. Language Model

In Translation, we always use OpenAI Codex (Chen et al., 2021) (`code-davinci-002`, with 175B parameters) as the underlying LM, since it is so far the only code-generation model with a public API.[7]

[7]See Appendix A for implementation details.

4.4. Baselines

We compare our method to two other baselines, shown in Figure 4: **standard** few-shot prompting, popularized by Brown et al. (2020), with demonstrations of only the question and the answer (green); and **CoT** prompting (Wei et al., 2022), which additionally provides a reasoning chain in NL (blue). We also show the published SOTA few-shot results.[8] All prompting methods are compared under two decoding strategies: **greedy** decoding, where the LM samples the most probable next token from the vocabulary (i.e., temperature = 0.0); and **self-consistency** decoding (Wang et al., 2022), where the LM generates multiple reasoning chains and chooses the final chain based on majority voting on the evaluated answer (we use a temperature of 0.4 and 40 generations for all datasets).[9] We reproduce the baseline results ourselves in cases when they are not reported or when we clean the test set.

5. Results

Our results on all datasets are shown in Table 1. We see that Faithful CoT outperforms CoT across most datasets and domains for both greedy and self-consistency decoding.

With greedy decoding, our method outperforms CoT on 9 of the 10 benchmark datasets spanning the 4 domains. On average, it improves over CoT in the MWP domain by 4.4, in Planning by 1.9, in Multi-hop QA by 4.0, and in Logical Inference by a surprising 18.1.

Our method also outperforms CoT under self-consistency decoding on 8 out of the 10 datasets. Compared to greedy decoding, the average accuracy gain becomes larger for Planning ($1.9 \rightarrow 3.9$) and Logical Inference ($18.1 \rightarrow 26.2$), but smaller for MWP ($4.4 \rightarrow 2.5$) and Multi-hop QA (4.0

[8]See Appendix B for sources of the SOTA results.

[9]Note that we do not report the performance of standard prompting with self-consistency decoding, since when the number of sampled outputs is large enough, this converges to standard prompting with greedy decoding (Wang et al., 2022).

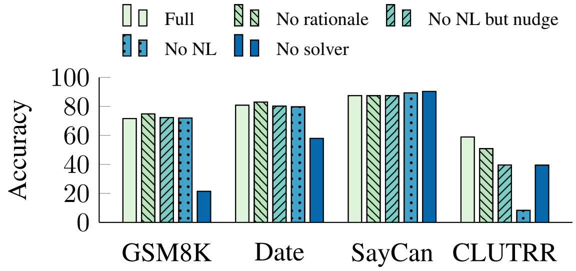

Figure 5. Ablation study results: accuracy when we remove different parts of the prompt. See Section 6.1 for details.

→ 2.7). Also, it is worth noting that our method achieves the new few-shot SOTA results on 7 datasets.

On StrategyQA (from the multi-hop QA domain), however, the performance of our method is still far from CoT and even standard prompting. To understand why, we specifically compare the examples where CoT makes a correct prediction but our method fails. As shown in Figure 9 in the Appendix, we find that the likely primary cause is the sparsity of Datalog in the pretraining data for Codex, as an overwhelming 29% of errors are syntax-related. Moreover, including Datalog in the prompt also interferes with NL generation, making it harder for Codex to produce relevant subquestions (17%), retrieve knowledge correctly (10%), and come up with valid reasoning from the knowledge to the answer (10%). Another potential cause is the nature of the task, as the difficulty for many StrategyQA questions does not lie in symbolic operations (value/string comparisons) but rather in retrieving correct facts and chaining them up, which makes the advantages of our deterministic solver less obvious. Still, with further pretraining on Datalog, we believe that there is room for improvement with our method.

6. Analysis

In this section, we analyze the role of different components in our pipeline, to better understand where its capabilities come from and where it still struggles.[10] Unless otherwise stated, we choose one dataset from each domain for analysis – GSM8K, Date Understanding, SayCan, and CLUTRR – using greedy decoding.

6.1. Ablation Study

Given the strong performance of Faithful CoT, we now address a natural question: **how much does each part of the prompt contribute to the accuracy?** We perform an ablation study where we remove different parts of the prompt and see how the performance changes. In addition to the original prompt ("Full"), we test four variations, illustrated with the example from Figure 4:

[10]See Appendix C for full results where figures do not show accuracy scores, and additional analysis of constraints and LMs.

Table 2. Robustness to the choice of exemplars across 6 runs.

Exemplars	GSM8K	Date	SayCan	CLUTRR
Set 0 (Table 1)	71.6	80.8	89.3	58.9
Set 1	72.3	81.3	90.3	59.0
Set 2	70.8	85.0	85.4	57.2
Set 3	71.6	82.5	88.3	58.0
Set 4	70.6	77.4	88.3	55.5
Set 5	68.5	85.0	89.3	56.0
Mean	70.9	82.0	88.5	57.4
Std.	1.3	2.9	1.7	1.5

No rationale. We remove the rationales, i.e., everything in the brackets from the NL comments, e.g., "independent, support: ['There are 15 trees']".
No NL but nudge. We remove all NL comments except the "nudge" line: e.g., "# To answer this question, we write a Python program to answer the following subquestions".
No NL. We remove all NL comments.
No solver. Instead of calling the external solver, we add "Answer: {answer}" to the end of every exemplar and let the LM predict the answer itself.

Figure 5 shows the results of all prompt variations. On GSM8K, Date Understanding, and SayCan, NL comments contribute little to the performance, and sometimes even slightly hurt it. On CLUTRR, however, their role is crucial, since the exclusion of each component (rationale, nudge, subquestions) results in a clear accuracy drop. In particular, comparing **No NL but nudge** and **No NL**, the nudge line itself brings a striking improvement by 31.3 points.

The external solver relieves the burden of problem solving from the LM. Without it, the accuracy suffers a huge decline on GSM8K, Date Understanding, and CLUTRR (-50.8, -22.9, and -19.4 respectively), while on SayCan it improves by 2.9 nonetheless. One potential influencing factor is that SayCan might be too homogeneous, as it contains a set of only 3 predefined actions. This can make the task relatively easy, which allows all model variants to achieve around 90% accuracy and renders the solver unnecessary. Another potential reason is the level of correspondence between the final answer and the reasoning chain for different datasets: as shown in Figure 3, the answer in SayCan is a sequence of actions (e.g., `find(redbull)`), each directly corresponding to one step in the reasoning chain (e.g., `at redbull trash`). However, the answer in the other three datasets is only a single number or string, which can only be derived after executing *all* the steps in the reasoning chain. Therefore, the latter type of tasks further necessitates the presence of an external solver.

6.2. Robustness to Exemplars

We now answer the next question: **how much does the choice of exemplars matter**? To do this, we annotate 20 examples in total, randomly sample k (7-10, depending on the dataset) to construct the prompt, and repeat the process five times. Table 2 shows the performance of all six runs, including the original (from Table 1). The mean accuracy is close to the original (-1.5 to +1.2), still above the baselines by a large margin (7 to 17) on all datasets except the arguably easiest SayCan, considering the standard deviation (1.3 to 2.9). This strongly suggests that the benefits of Faithful CoT are minimally influenced by the choice of exemplars.

6.3. Error Analysis

To further investigate where our method still fails, we inspect 100 errors[11] from model predictions on each of the four datasets and manually annotate the error categories. We only present the results on GSM8K here, shown in Figure 6; see Appendix F for those on the other datasets.

We categorize the errors on GSM8K into 6 types, inversely sorted with frequency:
Wrong Subquestion (49%): The LM produces a wrong NL subquestion, which eventually leads to the incorrect answer. While this is the majority error type in our sample, it is worth noting that in a typical human-in-the-loop collaboration, these errors are easily fixable. Even if the user is unfamiliar with programming, they can inspect the NL subquestions and potentially correct the model error by simply deleting or editing a wrong subquestion.
Wrong Code (24%): The NL subquestion is correct, but the code fails to answer the subquestion correctly. For example, the code uses a variable that has not been previously defined.
Semantic Understanding Error (12%): The LM incorrectly interprets certain semantic subtleties in the query. This is the most complex and most interesting error category. For example, consider the following problem:

> *If Martin eats Cheerios every day for breakfast, he'll* ***lose*** *1.25 pounds/week. If he eats donuts every day for breakfast, he'll* ***gain*** *1.75 pounds/week. What will be the difference in his weight at the end of 5 weeks between the two breakfast options?*

The generated code, included in Appendix F.1, does not assign opposite polarities (signs) for "pounds lost" vs. "pounds gained". For other examples in this category, we notice errors like missing that a pair of something has 2 items in it, missing to subtract 2 for "two years ago" when it occurs as a subjunctive, and so on. Fixing these errors, in general, will require more than providing additional examples in the prompt.
Generation Cutoff (7%): The generation stops midway, mainly due to the LM producing the same steps over and over again. These errors could be easily detected in postprocessing and possibly fixed by re-prompting the LM.
Wrong Gold Label (5%): We find 5 (out of our 100) examples that are genuine annotation errors in the gold labels.
Missing Subquestion (3%): The LM misses a relevant subquestion needed for the rest of the reasoning chain to work. These errors are also potentially fixable via human-in-the-loop interaction, where the user can insert a subquestion into the reasoning chain.

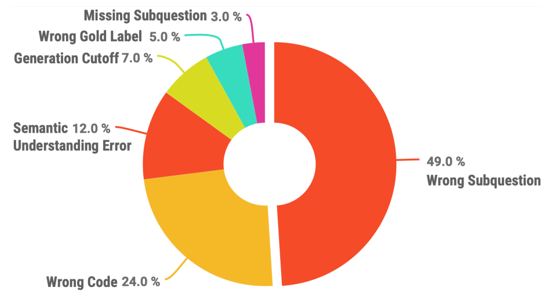

Figure 6. Error analysis for GSM8K. For a detailed description of the error categories, see Section 6.3.

7. Conclusion

We propose Faithful CoT, a framework that decomposes complex reasoning into Translation and Problem Solving. During Translation, an LM produces a reasoning chain in the form of interleaved natural and symbolic language. The Problem-Solving stage calls an external solver that executes the reasoning chain and derives the final answer. This process guarantees that the reasoning chain is a faithful explanation of how the model arrives at the answer. We demonstrate the efficacy of our approach on 4 types of complex reasoning problems: Math Word Problems, Multi-hop QA, Planning, and Logical Inference. Our method sets new SOTA performance on 7 of the 10 datasets, while additionally providing a faithful explanation for the final answer. These results give empirical evidence that improving model *interpretability*, by guaranteeing the faithfulness of an explanation, does not come at the expense of overall *performance*; in fact, we see a strong synergy in between. Through a comprehensive analysis on the strengths and weaknesses of our method, we show its robustness to the choice of exemplars, the pivotal role of the solver, as well as frequent error patterns where it still struggles.

One limitation of our work is that the Translation stage is still opaque, leaving an open question about whether it is possible to improve its faithfulness as well. Moreover, it will be helpful to perform a human evaluation on the correctness of the generated reasoning chains. Finally, the NL comments in the reasoning chain can serve as an interface for users without a programming background to interactively debug the model, which should be explored in future work.

[11] To encourage sample diversity, we embed all the errors using `text-davinci-002` and cluster the embeddings using spectral clustering. This produces around 70 clusters of different sizes, from which we gather 100 samples using importance sampling.

Acknowledgements

This research is based upon work supported in part by the DARPA KAIROS Program (contract FA8750-19-2-1004), the DARPA LwLL Program (contract FA8750-19-2-0201), the IARPA BETTER Program (contract 2019-19051600004), the IARPA HIATUS Program (contract 2022-22072200005), and the NSF (Award 1928631). Approved for Public Release, Distribution Unlimited. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, DARPA, IARPA, NSF, or the U.S. Government.

We appreciate the support from OpenAI on increasing the rate limit for the Codex API. We also thank Jiani Huang, Ziyang Li, Litao Yan, Andrew Head, and Mayur Naik for their valuable feedback.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N. J., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, aug 2022. URL http://arxiv.org/abs/2204.01691. arXiv:2204.01691 [cs].

Bengio, Y. FROM SYSTEM 1 DEEP LEARNING TO SYSTEM 2 DEEP LEARNING. 2019.

BIG-Bench collaboration. Beyond the Imitation Game: Measuring and extrapolating the capabilities of language models, 2021. URL https://github.com/google/BIG-bench/.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

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

Chen, W., Ma, X., Wang, X., and Cohen, W. W. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks, nov 2022. URL http://arxiv.org/abs/2211.12588. arXiv:2211.12588 [cs].

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training Verifiers to Solve Math Word Problems, nov 2021. URL http://arxiv.org/abs/2110.14168. arXiv:2110.14168 [cs].

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., and Neubig, G. PAL: Program-aided Language Models, nov 2022. URL http://arxiv.org/abs/2211.10435. arXiv:2211.10435 [cs].

Geva, M., Khashabi, D., Segal, E., Khot, T., Roth, D., and Berant, J. *Did Aristotle Use a Laptop?* A Question Answering Benchmark with Implicit Reasoning Strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361, apr 2021. ISSN 2307-387X. doi: 10.1162/tacl_a_00370. URL https://direct.mit.edu/tacl/article/doi/10.1162/tacl_a_00370/100680/Did-Aristotle-Use-a-Laptop-A-Question-Answering.

Gilpin, L. H., Bau, D., Yuan, B. Z., Bajwa, A., Specter, M., and Kagal, L. Explaining Explanations: An Overview of Interpretability of Machine Learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 80–89, oct 2018. doi: 10.1109/DSAA.2018.00018.

Harrington, L. A., Morley, M. D., Šcedrov, A., and Simpson, S. G. *Harvey Friedman's Research on the Foundations of Mathematics*. Elsevier, nov 1985. ISBN 978-0-08-096040-1. Google-Books-ID: 2plPRR4LDxIC.

Herman, B. The Promise and Peril of Human Evaluation for Model Interpretability. *arXiv:1711.07414 [cs, stat]*, oct 2019. URL http://arxiv.org/abs/1711.07414. arXiv: 1711.07414.

Jacovi, A. and Goldberg, Y. Towards faithfully interpretable NLP systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4198–4205. Association for Computational Linguistics, jul 2020. doi: 10.18653/v1/2020.acl-main.386. URL https://aclanthology.org/2020.acl-main.386.

Jung, J., Qin, L., Welleck, S., Brahman, F., Bhagavatula, C., Bras, R. L., and Choi, Y. Maieutic Prompting: Logically Consistent Reasoning with Recursive Explanations, may 2022. URL http://arxiv.org/abs/2205.11822. arXiv:2205.11822 [cs].

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large Language Models are Zero-Shot Reasoners. oct 2022. URL https://openreview.net/forum?id=e2TBb5y0yFf.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., Slone, A., Anil, C., Schlag, I., Gutman-Solo, T., Wu, Y., Neyshabur, B., Gur-Ari, G., and Misra, V. Solving Quantitative Reasoning Problems with Language Models, jun 2022. URL http://arxiv.org/abs/2206.14858. arXiv:2206.14858 [cs].

Li, Y., Lin, Z., Zhang, S., Fu, Q., Chen, B., Lou, J.-G., and Chen, W. On the Advance of Making Language Models Better Reasoners, jun 2022. URL http://arxiv.org/abs/2206.02336. arXiv:2206.02336 [cs].

Ling, W., Yogatama, D., Dyer, C., and Blunsom, P. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 158–167. Association for Computational Linguistics, July 2017. doi: 10.18653/v1/P17-1015. URL https://aclanthology.org/P17-1015. arXiv:1705.04146 [cs].

Miao, S.-y., Liang, C.-C., and Su, K.-Y. A diverse corpus for evaluating and developing English math word problem solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 975–984, Online, jul 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.92. URL https://aclanthology.org/2020.acl-main.92.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., Sutton, C., and Odena, A. Show Your Work: Scratchpads for Intermediate Computation with Language Models, nov 2021. URL http://arxiv.org/abs/2112.00114. arXiv:2112.00114 [cs].

Patel, A., Bhattamishra, S., and Goyal, N. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 2080–2094, Online, jun 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.168. URL https://aclanthology.org/2021.naacl-main.168.

Pruthi, D., Gupta, M., Dhingra, B., Neubig, G., and Lipton, Z. C. Learning to deceive with attention-based explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4782–4793, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.432. URL https://aclanthology.org/2020.acl-main.432.

Qian, J., Wang, H., Li, Z., Li, S., and Yan, X. Limitations of Language Models in Arithmetic and Symbolic Induction, aug 2022. URL http://arxiv.org/abs/2208.05051. arXiv:2208.05051 [cs].

Ribeiro, M., Singh, S., and Guestrin, C. "why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, KDD '16, pp. 97–101, New York, NY, USA, June 2016. Association for Computational Linguistics. ISBN 978-1-4503-4232-2. doi: 10.18653/v1/N16-3020. URL https://aclanthology.org/N16-3020.

Roy, S. and Roth, D. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1743–1752, Lisbon, Portugal, sep 2015. Association for Computational Linguistics. doi: 10.18653/v1/D15-1202. URL https://aclanthology.org/D15-1202.

Sinha, K., Sodhani, S., Dong, J., Pineau, J., and Hamilton, W. L. CLUTRR: A diagnostic benchmark for inductive reasoning from text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4506–4515, Hong Kong, China, nov 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1458. URL https://aclanthology.org/D19-1458.

Slack, D., Hilgard, S., Jia, E., Singh, S., and Lakkaraju, H. Fooling LIME and SHAP: Adversarial Attacks on Post hoc Explanation Methods. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, pp. 180–186. Association for Computing Machinery, New York, NY, USA, feb 2020. ISBN 978-1-4503-7110-0. URL https://doi.org/10.1145/3375627.3375830.

Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-Consistency Improves Chain of Thought Reasoning in Language Models, oct 2022. URL http://arxiv.org/abs/2203.11171. arXiv:2203.11171 [cs].

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. Chain of Thought Prompting Elicits Reasoning in Large Language Models. oct 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Bousquet, O., Le, Q., and Chi, E. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models, may 2022. URL http://arxiv.org/abs/2205.10625. arXiv:2205.10625 [cs].

A. Implementation Details

In all our experiments, we use OpenAI GPT-3 (`text-davinci-001` and `text-davinci-002`) and Codex (`code-davinci-001` and `code-davinci-002`) models through the Python API available at `beta.openai.com`, from Sept, 2022 to Jan, 2023. The inference cost per example is $0 for all Codex models since they are in limited beta period, and $0.01 - $0.03 for GPT-3 models depending on the dataset. It takes 2-15 seconds to run inference on one example with Codex models under a rate limit of 150,000 tokens/minute, and 1-8 seconds with GPT-3 models under 250,000 tokens/minute, also depending on the dataset. For example, on the GSM8K test set of 1,319 examples, it takes 3.5h to finish the inference with Codex and 2.3h with GPT-3.

We use the following hyper-parameters throughout all experiments:

temperature: 0.0 for greedy decoding, 0.4 for self-consistent decoding;
max_tokens: 1000;
n: 1 for greedy decoding, 40 for self-consistent decoding;
frequency_penalty: 0;
presence_penalty: 0.

Any unspecified hyper-parameters are set to the default value on `https://beta.openai.com/docs/api-reference/completions/create`.

B. Extended Results and Analysis

In this section, we present more results that do not fit into the main text.

B.1. Examples of Unfaithful CoT Output

Figure 7 shows examples of unfaithful output from the CoT method (Wei et al., 2022) on three datasets: GSM8K (MWP), StrategyQA (Multi-hop QA), and SayCan (Planning). Note that here we only show model outputs that are apparently unfaithful. In other cases, there is no *known* inconsistency between the reasoning chain and the answer, but this still does not imply that the reasoning chain is a faithful explanation, since the ground truth (how the model predicts the answer) is unknown.

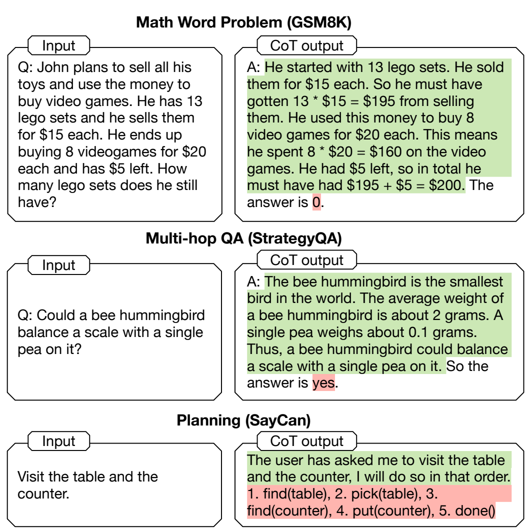

Figure 7. Examples of *unfaithful* output from CoT prompting (Wei et al., 2022) on three datasets. The answer (green) does not follow from the reasoning chain (blue).

The GSM8K example is explained in Section 1. In the StrategyQA example, though the reasoning chain correctly identifies that a hummingbird weighs much more than a pea, the answer is still "yes"; in the SayCan example, the reasoning chain only mentions "visit the table and the counter", but the plan contains unnecessary "pick" and "put" operations.

Wei et al. (2022) claim that CoT "provides an interpretable window into the behavior of the model, suggesting how it might have arrived at a particular answer and providing opportunities to debug where the reasoning path went wrong". As we have pointed out in Section 1, since CoT does not guarantee faithfulness, how the model arrives at the answer could differ drastically from what is shown in the reasoning chain. Furthermore, it is still hard for the user to debug the model: even if they manually correct the reasoning chain and let the model regenerate the answer, it might still be wrong, since there is no causality between the reasoning chain and the answer.

B.2. Few-shot SOTA Sources

The published few-shot SOTA results we compare to in Section 5 are from the following studies:

GSM8K, SVAMP, MultiArith, ASDiv, AQuA, StrategyQA: Wang et al. (2022);

SayCan, Date Understanding, Sports Understanding: (Wei et al., 2022);

CLUTRR: No existing work reports few-shot performance on CLUTRR with K up to 10.

C. Extended Analysis

C.1. Ablation Study

Table 3 shows the full results of the ablation study from Section 6.1.

Table 3. Ablation study results that accompany Figure 5. We report accuracy when we remove different parts of the prompt.

Exemplars	GSM8K	Date	SayCan	CLUTRR
Full	71.6	80.8	87.4	**58.9**
No rationale	**74.8**	**83.0**	87.4	50.9
No NL but nudge	72.3	80.2	87.4	39.6
No NL	72.0	79.7	89.3	8.3
No solver	21.4	57.9	**90.3**	39.5

C.2. Effect of LM

In this analysis, we want to answer the question: **how much does the choice of LM matter?** All experiments above are done using `code-davinci-002`. Here we examine the effect of using different LMs as the translator, as shown in Figure 8 and Table 4. Clearly, `code-davinci-002` is far superior to all other models. While the exact differences between these closed-source models are not yet clear, we speculate the following causes. Given that our prompt is a mixture of code and NL comments:

`code-davinci-002` is pretrained on NL and then code;

`code-davinci-001` is pretrained on code only, which might explain its inability to work with NL comments;

`text-davinci-001` is pretrained on NL only, which might explain its inability to work with code;

`text-davinci-002` is pretrained on both NL and code and receives further instruction tuning in NL, which might have drifted it from code again.

Table 4. Accuracy of Faithful COT with different LMs as the Translator, accompanying Figure 8. Due to the prompt length limit, `text-davinci-001` only allows us to run experiments on CLUTRR.

Exemplars	GSM8K	Date	SayCan	CLUTRR
code-davinci-002	71.6	80.8	89.3	58.9
text-davinci-002	62.1	76.6	79.6	43.9
code-davinci-001	26.5	43.2	66.0	23.4
text-davinci-001	N/A	N/A	N/A	17.3

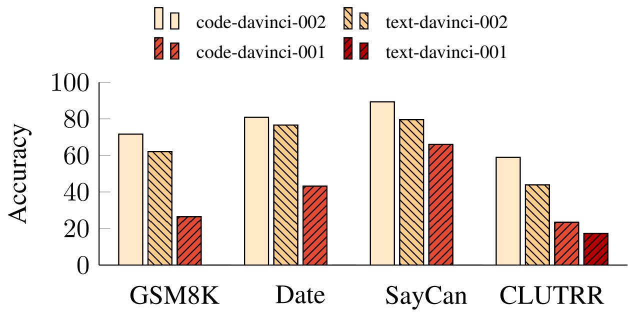

Figure 8. Accuracy of Faithful COT with different LMs as the Translator. Due to the prompt length limit, `text-davinci-001` only allows us to experiment on CLUTRR.

Table 5. Accuracy change after enforcing different constraints on the generation. The "None" row shows the original performance without any constraint (from Table 1). Each row below adds a different set of constraints: G stands for "graph validity", O for "no over-dependency", and U for "no under-dependency". Results are on all MWP datasets under self-consistent decoding.

Constraint	GSM8K	SVAMP	MultiArith	ASDiv	AQuA
None	79.5	88.9	98.8	81.9	61.4
+ G	0.0	0.0	0.0	-0.1	-0.8
+ O	-0.9	-0.1	-0.1	+0.4	-3.9
+ U	-1.0	-3.6	0.0	-1.2	+1.2
+ GO	-1.7	-0.4	-0.1	+0.2	-3.9
+ GU	-1.0	-3.7	0.0	-1.2	+0.8
+ OU	-4.0	-5.4	-0.1	-2.6	-4.3
+ GOU	-5.0	-5.9	-0.1	-3.2	-5.5

C.3. Enforcing Constraints

Since our generated reasoning chain contains structured components (e.g., dependency graphs), another natural question to ask is: **will it be helpful to enforce certain constraints on the generation?** Using MWP datasets as a case study, we examine the effect of three such constraints:

Graph validity. The dependency graph must be a Directed Cyclic Graph (DAG), e.g., it is not allowed for a subquestion to depend on itself.

No over-dependency. The code cannot depend on *any* variable that its corresponding subquestion has not mentioned, e.g. in Figure 4, since Q5 says "depend on 4", then the corresponding code should *not* use the variable `eggs_in_dozen`, since it is not the output of Q4.

No under-dependency. The code must depend on *all* variables that its corresponding subquestion has mentioned, e.g. in the same example, since Q5 says "depend on 4", then the corresponding code *must* use the variable `eggs_in_dozen`.

We investigate the effect of adding constraints on the generations under self-consistent decoding. Starting with our original results (without any constraint), we add a different set of constraints at each time and report the accuracy change in Table 5. Individually, the graph validity constraint results in little to no change in the performance, but the other two constraints lead to a more unstable change–mostly a decrease–across datasets. Adding two or more constraints further lowers the performance in almost all cases except on MultiArith (the easiest dataset), revealing the tradeoff between accuracy and satisfying the constraints. It also indicates that a proportion of generations (1.0% to 8.9%) in our existing results do not satisfy all constraints. However, it may still be worth enforcing some of these constraints (e.g., graph validity) at the cost of performance, in order for users to better control and interact with the model.

D. Broader Impacts

With the recent success of generative large LMs, they are now being used to solve complex reasoning problems. When using the output of an LM for reasoning, there is a danger that if the reasoning *appears* realistic, then the final answer or conclusion will also be considered reliable. As we highlighted in Figure 7, this is often not true, since an LM may produce a reasoning chain that looks plausible, but the final answer is still wrong. This work is a step in the direction of making the use of LMs more trustworthy by using the LM for just expressing its reasoning in a symbolic program and executing the program independently. In this work, we have ensured the faithfulness of the reasoning chain w.r.t how the final answer is produced in a variety of domains, but admittedly the Translation phase is still opaque. Therefore, our pipeline is still not entirely interpretable and can sometimes produce erroneous answers, which may pose a risk for users that rely on our method for decision making. Another potential impact of our work is that since the reasoning chain interleaves NL comments and symbolic programs, it may allow users without background knowledge in programming to easily understand the model output and debug the model when it makes errors.

E. Dataset Details

E.1. Statistics

We show the dataset details in Table 6, including the statistics, the number of few-shot exemplars used in the prompt, and example inputs and outputs.

In particular, we notice that in one of our baselines Wei et al. (2022), the reported number of exemplars used in the prompt is inconsistent between the main text (10) and the appendix (6). To ensure fair comparison, we rerun the baseline with 10 exemplars for our results in Table 1, which is what we use for our method.

Table 6. Datasets used for evaluation. "# Shot" stands for the number of few-shot examples in the prompt (following Wei et al. (2022)) and "# Test" stands for the number of test examples.

Domain	Dataset	# Shot	# Test	Example
Math Word Problems	GSM8K	8	1,319	Q: Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May. How many clips did Natalia sell altogether in April and May? A: `72`
	SVAMP	8	1,000	Q: Each pack of dvds costs 76 dollars. If there is a discount of 25 dollars on each pack. How much do you have to pay to buy each pack? A: `51`
	MultiArith	8	600	Q: For Halloween Debby and her sister combined the candy they received. Debby had 32 pieces of candy while her sister had 42. If they ate 35 pieces the first night, how many pieces do they have left? A: `39`
	ASDiv	8	2,096	Q: Seven red apples and two green apples are in the basket. How many apples are in the basket? A: `9`
	AQuA	8	254	Q: A car finishes a journey in 20 hours at the speed of 60 km/hr. If the same distance is to be covered in 10 hours, how much speed does the car gain? A: `"120 kmph"`
Multi-hop QA	StrategyQA	6	2,290	Q: Did Aristotle use a laptop? A: `False`
	Date Understanding	10	359	Q: Yesterday was April 30, 2021. What is the date tomorrow in MM/DD/YYYY? A: `"05/02/2021"`
	Sports Understanding	10	977	Q: Is the following sentence plausible: "Lebron James hit the turnaround jumper"? A: `True`
Planning	SayCan	7	103	Q: Could you get me a drink with caffeine? A: `"1.find(redbull) 2.pick(redbull) 3.find(user) 4.put(redbull) 5.done()."`
Logical Inference	CLUTRR	8	1,042	Q: [Carlos] is [Clarence]'s brother. [Carlos] and his sister, [Annie], went shopping. [Annie] asked her mom [Valerie] if she wanted anything, but [Valerie] said no. How is [Valerie] related to [Clarence]? A: `"mother"`

E.2. URLs and Licenses

We use the same distribution of datasets following Wei et al. (2022):

Math Word Problems

- GSM8K (Cobbe et al., 2021): https://github.com/openai/grade-school-math, MIT license: https://github.com/openai/grade-school-math/blob/master/LICENSE.
- SVAMP (Patel et al., 2021): https://github.com/arkilpatel/SVAMP, MIT license: https://github.com/arkilpatel/SVAMP/blob/main/LICENSE.
- MultiArith (Roy & Roth, 2015), license: CC BY 4.0.
- ASDiv (Miao et al., 2020): https://github.com/chaochun/nlu-asdiv-dataset.
- AQuA (Ling et al., 2017): https://github.com/deepmind/AQuA, license: https://github.com/deepmind/AQuA/blob/master/LICENSE.

Multi-hop QA

- StrategyQA (Geva et al., 2021): we use the open-domain setting (question-only set) from (BIG-Bench collaboration, 2021): https://github.com/google/BIG-bench/tree/main/bigbench/benchmark_tasks/strategyqa.
- Date Understanding and Sports Understanding from BIG-Bench (BIG-Bench collaboration, 2021): Apache License v.2: https://github.com/google/BIG-bench/blob/main/LICENSE.

Planning

- SayCan (Ahn et al., 2022): SayCan dataset can be accessed at https://say-can.github.io/ under CC BY 4.0 license.

Logical Reasoning

- CLUTRR (Sinha et al., 2019): https://github.com/facebookresearch/clutrr, license: https://github.com/facebookresearch/clutrr/blob/main/LICENSE.

E.3. Data Cleaning

We perform manual cleaning on ASDiv, Date Understanding, Sports Understanding, and SayCan as we discover a number of annotation issues. In our experiment, we rerun all baselines on our cleaned version of the test sets. They are provided in the Supplementary Materials to assist future research.

Specifically, we clean each of the datasets as follows:

ASDiv: We start with the test set used by Wei et al. (2022), which removes all questions with float-valued and string-valued answers. However, in their released version, we notice an error in the answer extraction step for questions with more than one value in the answer (e.g., “what is the width and length of X?”, where the answer consists of two values). In their implementation, only the first value is extracted as the ground truth answer, which is then compared against model outputs. This might artificially inflate the final accuracy. To fix this, we extract all values in the answer as a set and compare model outputs against it.

Date Understanding: We find a number of wrong answers in the test set. For example, for the question “Jane and John married on Jan 2, 1958. It is their 5-year anniversary today. What is the date today in MM/DD/YYYY?”, the provided answer is “01/02/1961”, whereas the correct answer should be “01/02/1963”. We manually correct these answers, and the resulting test set has the same number of examples as the original one.

Sports Understanding: We notice a few ambiguities with the Sports Understanding dataset. For instance, running out of bounds is illegal in many sports. The phrase "Domantas Sabonis ran out of bounds" is labeled as implausible, however, Domantas Sabonis is a basketball player, and basketball players can indeed run out of bounds on the court. We remove 8 questions with such action-based ambiguities. Additionally, since the release of this dataset, a few new athletes have risen to fame with identical names to those mentioned in the dataset. For example, the question "Chris Paul struck out the side" is implausible, as the referenced “Chris Paul” is a famous basketball player. However, “Chris Paul” is also the name of a new MLB baseball player, in which case this statement is plausible. We remove 5 questions with such name-based ambiguities.

SayCan: We discover a few issues in the test set: (1) the environment setup (e.g., the list of objects, the list of locations, and the initial location of each object) is not the same for all examples; (2) the annotation of the ground truth answer is often incomplete (i.e., for a given task like "visit all locations", there exist many possible plans in terms of the order of locations visited, but not all of them are included in the annotation); (3) there are ambiguous descriptions in certain queries, for example, in "Could you get me something refreshing?", it is unclear what drinks are considered "refreshing". For these questions, we complete the annotation whenever possible, and filter out the rest. The resulting test set contains 103 examples out of the original 120.

E.4. Dataset Splits

As stated in Section 4.1, we use the official splits whenever possible: training set for exemplar selection, validation set for prompt tuning, and test set for evaluation. In cases where they are available, we adopt the following strategies for each dataset:

GSM8K: it only has training and test sets. We form the validation set by randomly sampling 1,000 examples from the training set.

Other MWP datasets: for AQuA, we use the official training/validation/test split. For the other datasets, only the test sets are used, since we have the same prompt for GSM8K and them.

Date Understanding and **Sports Understanding**: they only have test sets. We follow Wei et al. (2022) to select the same number of examples from the test set to form the few-shot prompt and use the remaining examples as a new test set.

SayCan: Following Wei et al. (2022), we manually write 7 few-shot exemplars, since no training set is provided. We evaluate the models on our cleaned version of the test set, described in the previous subsection.

CLUTRR: this dataset is split into multiple folds. There is a training fold with $K \in \{2, 3\}$ (where K is the number of intermediate steps required to reach the answer), and one test fold for each K from 2 to 10. We construct the few-shot prompt using exemplars from the training fold, and test our method on the concatenation of all test folds.

F. Error Analysis Details

Section 6.3 detailed error analysis and conclusions for GSM8K. In this section, we continue our error analyses for the rest of the datasets we evaluate in the paper.

F.1. GSM8K

Table 7 lists generated code for the semantic understanding error example cited in Section 6.3.

Table 7. Generated code for question in Appendix F.

```
# 1. How many pounds will Martin lose per week if he eats Cheerios every day for breakfast? (independent, support: ["he'll lose 1.25 pounds/week"])
pounds_lost_cheerios = 1.25
# 2. How many pounds will Martin gain per week if he eats donuts every day for breakfast? (independent, support: ["he'll gain 1.75 pounds/week"]) pounds_gained_donuts = 1.75
# 3. How many weeks are there in 5 weeks? (independent, support: ["External knowledge: there are 7 days in a week"])
weeks_in_5_weeks = 5
# 4. How many pounds will Martin lose in 5 weeks if he eats Cheerios every day for breakfast? (depends on 1 and 3, support: [])
pounds_lost_cheerios_5_weeks = pounds_lost_cheerios * weeks_in_5_weeks
# 5. How many pounds will Martin gain in 5 weeks if he eats donuts every day for breakfast? (depends on 2 and 3, support: [])
pounds_gained_donuts_5_weeks = pounds_gained_donuts * weeks_in_5_weeks
# 6. What will be the difference in his weight at the end of 5 weeks between the two breakfast options? (depends on 4 and 5, support: [])
difference_5_weeks = pounds_gained_donuts_5_weeks - pounds_lost_cheerios_5_weeks
# 7. Final Answer: What will be the difference in his weight at the end of 5 weeks between the two breakfast options? (depends on 6, support: [])
answer = difference_5_weeks
```

F.2. StrategyQA

As mentioned in Section 5, Figure 9 shows the error type distribution on a sample of 70 instances from StrategyQA, where we specifically compare the cases where the prediction of CoT is correct whereas ours is wrong.

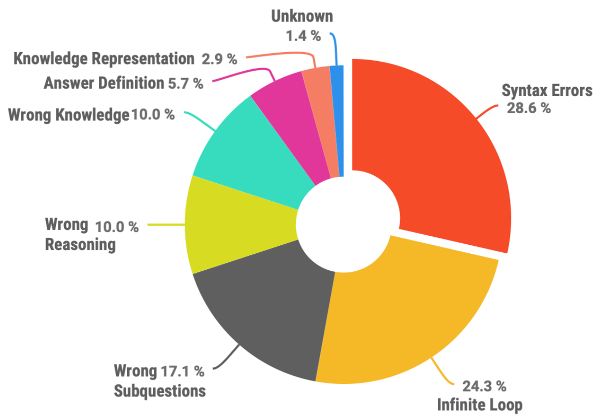

Figure 9. Error analysis for StrategyQA. For a detailed description of the error categories, see Section F.2.

There are 8 error types, from most to least frequent:

Syntax (29%): these are syntax-related errors that result in an invalid Datalog program. For example, the LM redefines a relation that is previously declared, uses an invalid string comparison operation, generates an ungrounded variable in the answer definition, and so on. With further pretraining on Datalog, we believe Codex is able to overcome these errors to a reasonable extent.

Infinite Loop (24%): the LM generates an infinite loop of subquestions, which are oftentimes rather philosophical. For example, for the question "Are potatoes native to the European continent?", the LM generates a list of 40 subquestions (potentially more, if not truncated by the token limit):

1. Where are potatoes native to?
Potatoes are native to the Andes.
2. Which continent is the Andes in?
The Andes is in South America.
3. Which continent is South America in?
South America is in the Americas.
4. Which continent is the Americas in?
The Americas is in the Western Hemisphere.
5. Which hemisphere is the Western Hemisphere in?
The Western Hemisphere is in the Earth.
6. Which continent is the Earth in?
The Earth is in the Solar System.
......
11. Which continent is the Universe in?
The Universe is in the Multiverse.
12. Which continent is the Multiverse in?
The Multiverse is in the Omniverse.
13. Which continent is the Omniverse in?
The Omniverse is in the Metaverse.
14. Which continent is the Metaverse in?
The Metaverse is in the Hyperverse.
......
39. Which continent is the Quattuordecaverse in?

The Quattuordecaverse is in the Quindecaverse.
40. Which continent is the Quindecaverse in?

We notice that this type of errors also happen if the prompt does not include Datalog code but just the NL subquestions, but much less often.

Wrong Subquestions (17%): The LM generates an irrelevant subquestion to the original question. For instance, given the question, "Would toast for a vegan have margarine instead of butter?", one subquestion that is generated is "What is the difference between a vegan and a vegetarian?", which does not help with reaching the final answer.

Wrong Reasoning (10%): The subquestions are correct, but the reasoning from the retrieved facts to the final answer is problematic. For example, to answer the question, "Do solo pianists require a conductor?", the model correctly generates the relevant subquestions and their respective answers:

1. What is the definition of a solo pianist?
A solo pianist is a pianist who performs alone.
2. What is the definition of a conductor?
A conductor is a person who leads an orchestra.
3. What is the definition of an orchestra?
An orchestra is a large instrumental ensemble that contains string, brass, woodwind, and percussion instruments.

But in deriving the final answer, the reasoning is wrong: "Now, we derive the final answer: Do solo pianists require a conductor? The answer is Yes only if a solo pianist is not a conductor."

Wrong Knowledge (10%): the LM fails to retrieve the correct knowledge to answer the subquestions. For example, given the original question "Is the largest city in New Mexico also known as Yootó?", the model correctly generates the subquestions "What is the largest city in New Mexico?" (answer: Albuquerque) and "Is Albuquerque also known as Yootó?". However, when answering the second subquestion, it retrieves a wrong piece of knowledge ("Albuquerque is also known as Yootó.", whereas in reality, it should be "Santa Fe" that is known as Yootó).

Answer Definition (6%): In our prompt, we always derive the answer in the format of "The answer is Yes only if ...", which is followed by a Datalog rule containing conditions that should be satisfied for the answer to be true. However, the LM sometimes generates this as "The answer is No only if ...", which outputs the reversed answer.

Knowledge Representation (3%): The retrieved knowledge in NL is correct, but the representation of it in Datalog is wrong. For example, for the piece of knowledge "The Lucy Show is not the same TV series as JAG (TV series)", the model represents it as follows:

```
.decl Same_TV_series(TV_series1:symbol, TV_series2:symbol)
Same_TV_series("The Lucy Show", "JAG (TV series)")."
```

which actually means the reverse (they are the same).

Unknown (1%): There is a very small proportion of errors (1 out of 70) where we are unsure of the cause. Specifically, we expect the solver to output True, but it outputs False instead.

F.3. Date Understanding

Unlike GSM8K, we only have 69 errors out of the 359 test examples, so we annotate them all, as shown in Figure 10. The error categories for date understanding are similar to GSM8K, except that we do not see any generation errors in the samples, but we see questions with ambiguous phrasing allowing both the gold and predicted answers to be correct based on interpretation.

F.4. SayCan

Since SayCan only has 120 test examples and Faithful CoT produces 7 errors, we annotate all 7 of them, as shown in Figure 11. These 7 examples can be categorized into two types:

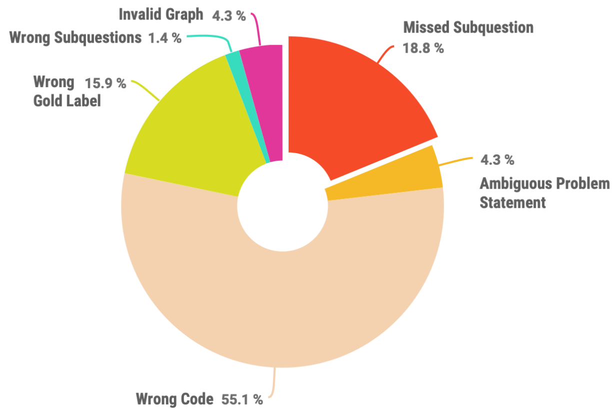

Figure 10. Error analysis for Date Understanding. For a detailed description of the error categories, see Section F.3.

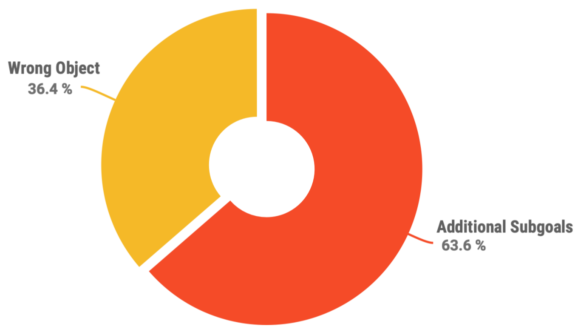

Figure 11. Error analysis for SayCan. For a detailed description of the error categories, see Section F.4.

Additional Subgoals (64%): Cases where the model generated unnecessary subgoals in the decomposition of the original task, leading the planner astray. This is illustrated by the request "Clear the jalapeno chips off the counter":

```
(:goal
    (and
            (not (at jalapeno-chips counter))
            (not (at jalapeno-chips table))
            (not (at jalapeno-chips trash))
            (not (at jalapeno-chips bowl))
            (not (at jalapeno-chips user))
    )
)
```

Wrong Object (36%): Here the model generates the wrong object/object types in the goal. For example, a request such as "I opened a pepsi earlier. How would you bring me an open can?" fails because the model generates actions with water instead of Pepsi.

F.5. CLUTRR

For CLUTRR, we group all error cases by K, the number of steps in their gold reasoning chain, as a proxy for problem complexity, and perform importance sampling on these groups to select 100 examples. Our annotation of these examples reveals 5 error categories, as shown in Figure 12:

Inversed Relation (41%): This stands out as the majority of the errors. These errors are caused by the reversal of directional relationships for the actors in the problem, i.e., predicting "mother" or "nephew" when the answer is "daughter" or "uncle" respectively.

Wrong Relation (30%): Here the model extracts the relation incorrectly (not even the inverse). For example, for the subquestion "How is [Donald] related to [Jason]?" with the correctly identified support "[Jason] is father of their father", the

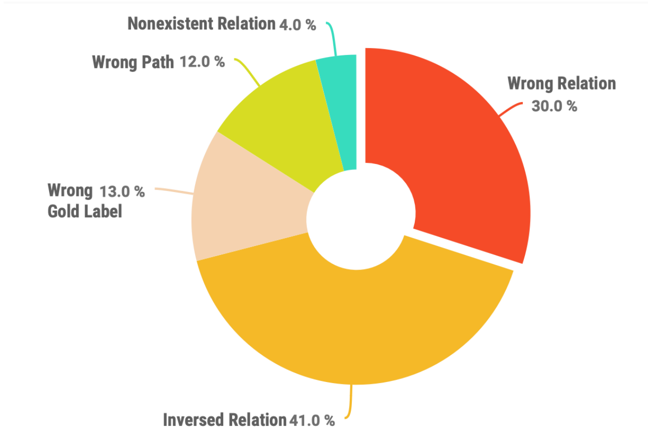

Figure 12. Error analysis for CLUTRR. For a detailed description of the error categories, see Section F.5.

model produces `relation(Donald, Jason) = son` when the correct relation should be "grandson".
Nonexistent Relation (4%): The model hallucinates a non-existent relation (e.g. "adopted" for daughter).
Wrong Path (12%): Here, the model does not generate a correct reasoning path from target entity A to target entity B in the question.
Wrong Gold Label (13%): These are annotation errors in the CLUTRR dataset. In one example, for the sentence, "[Gloria] asked her mother [Laura] if she could go outside and play with her friends.", the annotation says Laura is Gloria's grandmother.

G. Prompts

Due to the space limit, we show one exemplar in the prompt for each dataset here. Our full prompts can be found in the Supplementary Materials.

Among all the MWP datasets, our prompt for AQuA is different from the rest, because the answers are in a multiple-choice format instead of integers. To produce a multiple-choice answer, we take a two-step approach by first producing a numerical answer in the same way as for the other math datasets. Then, we perform an additional step of converting the numerical answer into an answer choice by again prompting the language model to generate which answer choice is closest to the previously produced numerical answer. An exemplar of this 2-step prompt is shown in Table 8.

Table 8. An exemplar from our prompt for AQuA.

EXEMPLAR FOR AQUA

Step 1: Answer Prediction

```
# Question: In a flight of 600 km, an aircraft was slowed down due to bad weather. Its average speed for the trip was reduced by 200 km/hr and the time of flight increased by 30 minutes. The duration of the flight is:

# Answer option: ['A)1 hour', 'B)2 hours', 'C)3 hours', 'D)4 hours', 'E)5 hours']
# Write Python code to solve the following questions. Store your result as a variable named 'answer'.
# 1. What was the duration of the flight? (independent, support: ["The duration of the flight is"])
duration = Symbol('duration', positive=True)
# 2. What is the delay of the flight? (independent, support: ["the time of flight increased by 30 minutes"])
delay = 30 / 60
# 3. What was the total flight distance? (independent, support: ["In a flight of 600 km"])
total_distance = 600
# 4. What was the original speed? (depends on 1 and 3, support: ["External knowledge: speed is distance over time"])
original_speed = total_distance / duration
# 5. What was the reduced speed? (depends on 1, 2, and 3, support: [])
reduced_speed = total_distance / (duration + delay)
# 6. What was the duration of the flight if the original speed was 200 km/hr faster than the reduced speed? (depends on 4, 5, and 1, support: [])
solution = solve_it(original_speed - reduced_speed - 200, duration)
answer = solution[duration]
```

Step 2: Multiple Choice Conversion

```
# Question: In a flight of 600 km, an aircraft was slowed down due to bad weather. Its average speed for the trip was reduced by 200 km/hr and the time of flight increased by 30 minutes. The duration of the flight is:

# Answer option: ['A)1 hour', 'B)2 hours', 'C)3 hours', 'D)4 hours', 'E)5 hours']
# Prediction: 1.00000000000000
# Closest Option: A
```

Table 9. An exemplar from our prompt for GSM8K, SVAMP, MultiArith, and ASDiv.

EXEMPLAR FOR GSM8K, SVAMP, MULTIARITH, AND ASDIV

```
# Q: There are 15 trees in the grove. Grove workers will plant trees in the grove today. After they are done, there will be 21 trees. How many trees did the grove workers plant today?

# To answer this question, write a Python program to answer the following subquestions:
# 1. How many trees are there in the beginning? (independent, support: ["There are 15 trees"])
trees_begin = 15
# 2. How many trees are there in the end? (independent, support: ["there will be 21 trees"])
trees_end = 21
# 3. How many trees did the grove workers plant today? (depends on 1 and 2, support: [])
trees_today = trees_end - trees_begin
# 4. Final Answer: How many trees did the grove workers plant today? (depends on 3, support: [])
answer = trees_today
```

Table 10. An exemplar from our prompt for StrategyQA.

EXEMPLAR FOR STRATEGYQA

```
// Q: Would a pear sink in water?

// To answer this question, we answer the following subquestions:
// 1. What is the density of a pear?
// The density of a pear is about 0.6g/cm^3.
// 2. What is the density of water?
// Water has a density of 1g/cm^3.

// Then, we represent these answers in Datalog:
// 1. The density of a pear is about 0.6g/cm^3.
.decl Has_density(Object:symbol, Density:float)
Has_density("pear", 0.6).
// 2. Water has a density of 1g/cm^3.
Has_density("water", 1).

// Now, we derive the final answer: Would a pear sink in water?
// The answer is Yes only if the density of a pear is more than the density of water.
.decl Answer()
Answer() :- Has_density("pear", density1), Has_density("water", density2), density1 > density2.
.output Answer
```

Table 11. An exemplar from our prompt for Date Understanding.

EXEMPLAR FOR DATE UNDERSTANDING

```
# Q: Yesterday was April 30, 2021. What is the date tomorrow in MM/DD/YYYY?

# To answer this question, we write a program to answer the following subquestions:
# import relevant packages
from datetime import date, time, datetime
from dateutil.relativedelta import relativedelta
# 1. What is the date yesterday? (independent, support: ["Yesterday was April 30, 2021"])
date_yesterday = date(2021,4,30)
# 2. What is the date today? (depends on 1, support: ["Yesterday was April 30, 2021"])
date_today = date_yesterday + relativedelta(days=1)
# 3. What is the date tomorrow? (depends on 2, support: [])
date_tomorrow = date_today + relativedelta(days=1)
# 4. Final Answer: What is the date tomorrow in MM/DD/YYYY? (depends on 3, support: [])
answer = date_tomorrow.strftime("%m/%d/%Y")
```

Table 12. An exemplar from our prompt for Sports Understanding.

EXEMPLAR FOR SPORTS UNDERSTANDING

```
# Q: Is the following statement plausible? Sam Darnold passed the puck

# To answer this question, write a Python program to answer the following subquestions:
# 1. Sam Darnold is a player in which sport? (independent, support: ["Sam Darnold is an NFL Quarterback", "NFL is the National Football League"])
player_sport = "football"
# 2. The phrase "passed the puck" implies playing which sport? (independent, support: ["Players pass the puck in hockey"])
playing_sport = "hockey"
# 3. Is the following statement plausible? Sam Darnold passed the puck (depends on 1 and 2, support: ["Sam Darnold is an NFL Quarterback", "NFL is the National Football League", "Players pass the puck in hockey"])
plausibility = (player_sport == playing_sport)
# 4. Is the following statement plausible? Sam Darnold passed the puck (depends on 3, support: [])
answer = int(plausibility)
```

Table 13. An exemplar from our prompt for SayCan.

EXEMPLAR FOR SAYCAN

User query: Bring me something not sweet to eat.

```
Goal in PDDL:
(:goal
    ; I need to find a snack
    (exists (?s - snack)
        ; it has to satisfy the following conditions
        (and
            ; the snack must not be sweet
            (not (is-sweet ?s))
            ; bring it to the user
            (at ?s user)
        )
    )
)
```

Table 14. An exemplar from our prompt for CLUTRR.

EXEMPLAR FOR CLUTRR

```
# Context: [Jason] always had some great adventure planned for his granddaughter [Guillermina] when she came to visit. So, naturally, when [Myrna] told her daughter [Guillermina] that they would be going to visit [Jason] she could hardly contain herself.

# Question: How is [Jason] related to [Myrna]?
# To answer this question, we write a program to answer the following subquestions:
# 1. How is [Jason] related to [Guillermina]? (independent, support: "[Jason] always had some great adventure planned for his granddaughter [Guillermina] when she came to visit.")
relation(Jason, Guillermina) = grandfather
# 2. How is [Guillermina] related to [Myrna]? (independent, support: "So, naturally, when [Myrna] told her daughter [Guillermina] that they would be going to visit [Jason] she could hardly contain herself.")
relation(Guillermina, Myrna) = daughter
# 3. Final answer: How is [Jason] related to [Myrna]? (depends on 1, 2)
relation(Jason, Myrna) = relation(Jason, Guillermina) @ relation(Guillermina, Myrna)
```