LAMBADA: Backward Chaining for Automated Reasoning in Natural Language

Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, Deepak Ramachandran
Google Research
{mehrankazemi, njkim, bhatiad, xxujasime, ramachandrand}@google.com

Abstract

Remarkable progress has been made on automated reasoning with natural text, by using Language Models (LMs) and methods such as Chain-of-Thought and Selection-Inference. These techniques search for proofs in the forward direction from axioms to the conclusion, which suffers from a combinatorial explosion of the search space, and thus high failure rates for problems requiring longer chains of reasoning. The classical automated reasoning literature has shown that reasoning in the backward direction (i.e. from the intended conclusion to supporting axioms) is significantly more efficient at proof-finding. Importing this intuition into the LM setting, we develop a *Backward Chaining* algorithm, called LAMBADA, that decomposes reasoning into four sub-modules. These sub-modules are simply implemented by few-shot prompted LM inference. We show that LAMBADA achieves sizable accuracy boosts over state-of-the-art forward reasoning methods on two challenging logical reasoning datasets, particularly when deep and accurate proof chains are required.

1 Introduction

Automated reasoning, the ability to draw valid conclusions from explicitly provided knowledge, has been a fundamental goal for AI since its early days (McCarthy, 1959; Hewitt, 1969). Furthermore, logical reasoning, especially reasoning with unstructured, natural text is an important building block for automated knowledge discovery and holds the key for future advances across various scientific domains. While in recent years tremendous progress has been made towards natural language understanding thanks to pretrained language models (LMs) (Brown et al., 2020; Chowdhery et al., 2022, *i.a.,*), the performance of these models for logical reasoning still lags behind (Rae et al., 2021; Creswell et al., 2023; Valmeekam et al., 2022) compared to the advancements in other areas such as reading comprehension and question-answering.

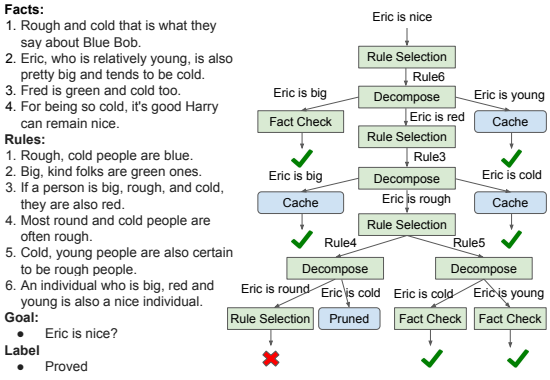


Figure 1: The search trace of LAMBADA on an example from the ParaRules subset of ProofWriter (the *Sign Agreement* and failed *Fact Check* modules are omitted for brevity).

While many problems benefit from LM scaling, scaling has been observed to provide limited benefit for solving complex reasoning problems. For example, Creswell et al. (2023) observed that for the Gopher family of LMs (Rae et al., 2021), the benefit of scaling for logic-based tasks is significantly worse than for other language tasks. Moreover, while finetuning initially seemed to enable logical reasoning in LMs (Clark et al., 2021; Tafjord et al., 2021), further exploration revealed that finetuned LMs mostly exploit spurious correlations (e.g., the correlation between the number of rules and the label) as opposed to learning to reason (Zhang et al., 2022b; Schlegel et al., 2022; Liu et al., 2023). Recently, prompting strategies such as Chain-of-Thought (Wei et al., 2022) and Scratchpad (Nye et al., 2022) have contributed to improving performance of LMs on reasoning tasks, although they have been also shown to struggle with proof planning for more complex logical reasoning problems (Saparov and He, 2023).

One solution to the aforementioned problems is to integrate the strength and reliability of classical AI models in logical reasoning with LMs (Garcez and Lamb, 2020; Marcus, 2020). In the literature,

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 6547–6568

there are two major approaches to logical reasoning (Poole and Mackworth, 2010):

1. *Forward Chaining (FC)* where one starts from the facts and rules ("theory"), and iterates between making new inferences and adding them to the theory until the goal statement can be proved or disproved,
2. *Backward Chaining (BC)* where one starts from the goal and uses the rules to recursively decompose it into sub-goals until the sub-goals can be proved or disproved based on the theory.

Previous approaches to reasoning with LMs mostly incorporate elements of FC into LMs (Tafjord et al., 2021; Creswell et al., 2023). FC requires selecting a subset of facts and rules from the entire set, which might be difficult for an LM as it requires a combinatorial search over a large space. Moreover, deciding when to halt and declare failure to prove is challenging in FC, as also noted by Creswell et al. (2023), sometimes requiring specialized modules trained on intermediate labels (Creswell and Shanahan, 2022). Indeed, the classical automated reasoning literature is heavily weighted towards BC or goal-directed strategies for proof-finding.

In this paper, we show experimentally that BC is better suited for text-based deductive logical reasoning, as it does not require a combinatorial search for subset selection and there are more natural halting criteria for it. We develop a hybrid **LA**nguage **M**odel augmented **BA**ckwar**D** ch**A**ining technique (LAMBADA), where BC drives the high-level proof planning, and the LM performs the textual understanding and individual reasoning steps. We conduct experiments with challenging datasets for LM reasoning containing examples expressed in naturalistic text. The datasets contain proof chains of up to 5 hops in depth, and examples where the goal can neither be proved nor disproved from the provided theory. We show that LAMBADA achieves substantially higher deductive accuracy, and is considerably more likely to generate valid reasoning chains compared to other techniques which find correct conclusions with spurious proof traces, while also being more query efficient than other LM-based modular reasoning approaches. Our results strongly indicate that future work on reasoning with LMs should incorporate backward chaining or goal-directed planning strategies.

2 Related Work

The deep learning based models that have been developed to solve text-based (logical) reasoning tasks can be categorized as follows (see Huang and Chang 2022 for a recent survey of the literature).

Pretraining on Relevant Tasks: Pretraining an LM on corpora relevant to the target reasoning task can lead to improvements (Hendrycks et al., 2021; Shen et al., 2021). Pretraining is, however, costly especially for larger LMs.

Implicit Reasoning: These approaches finetune LMs to produce the label directly given the input (Clark et al., 2021; Betz et al., 2021; Saeed et al., 2021; Han et al., 2022); reasoning is expected to happen implicitly in the parameters of the LM. It has been shown that finetuning LMs on logical reasoning tasks makes them learn spurious correlations (Zhang et al., 2022b; Schlegel et al., 2022), and is not robust to multi-hop reasoning (Kassner et al., 2020). Besides, finetuning large LMs is costly especially when the dataset is large, and may introduce distributional shocks to the model (Kazemi et al., 2023). In this paper, we focus on models that only take in-context examples as supervision.

Explicit Reasoning: Generating the intermediate reasoning steps such as the chain of reasoning (Wei et al., 2022; Nye et al., 2022; Dalvi et al., 2021; Zelikman et al., 2022; Zhang et al., 2022a) has shown substantial improvement for many reasoning tasks (Suzgun et al., 2022). Such chains have been explored both in the forward and the backward directions, e.g., using multiple constrained LMs for logical reasoning (Zhang et al., 2022a). Gontier et al. (2020) investigated how transformer models perform when trained to perform forward or backward chaining, and drew conclusions about their internal reasoning strategies. We compare against a popular recent prompting strategy, namely Chain-of-Thought (CoT) (Wei et al., 2022), from this category.

Verifiers: To improve CoT, some works train a verifier using chain-level labels. The verifier takes a reasoning chain produced by the model as input and judges the quality of the chain (Cobbe et al., 2021; Shen et al., 2021; Jhamtani and Clark, 2020; Zelikman et al., 2022). Using this verifier, one can then generate multiple reasoning chains (e.g., by running the algorithm multiple times with different decoding temperatures) and use the best chain according to the verifier. Since LAMBADA also

generates proofs, verifiers are also applicable to our algorithm. In this paper, we assume not having access to chain-level labels, and leave experiments with verifiers as future work.

Length generalization: A number of approaches specifically look into whether LMs can generalize from examples requiring shorter reasoning chains (shown to them either as demonstration or as finetuning data) to examples requiring longer chains (Anil et al., 2022; Tafjord et al., 2021). With our model, length generalization comes for free because the model learns the building blocks of solving the problem that are applied as many times as needed to solve the problem.

Modular Reasoning: These approaches break the problem into smaller modules and use separate LMs to solve each module (Zhou et al., 2022; Khot et al., 2023; Sprague et al., 2022; Zhou et al., 2023; Dua et al., 2022; Wang et al., 2022; Schlag et al., 2023). LM-based approaches to logical reasoning typically makes use of a single LM module; for example, in Tafjord et al. (2021), a single LM module iteratively and exhaustively infers *all* conclusions based on the facts and rules, and then the goal statement is compared against the final set of conclusions to confirm if it can be proved from the theory. Since exhaustively deriving all conclusions is computationally expensive, Creswell et al. (2023) consider a more scalable approach where the conclusions that are derived are informed by the goal; they iteratively apply two LLM modules one selecting a subset of the facts and rules informed by the goal and the other making new inferences based on the selected facts and rules and adding it back to the theory. In this paper, we compare against the second approach.

Natural Language Inference (NLI): Logical reasoning can also be understood as identifying whether a logical entailment relation holds between two propositions (premise and hypothesis; the premise is the theory and the hypothesis is the statement to be proved). In this sense, NLI models are also relevant, although inferences under NLI typically adopt a more relaxed notion of entailment rather than purely logical (Dagan et al., 2013; Bowman et al., 2015; Williams et al., 2018).

3 LAMBADA: Language Model Augmented Backward Chaining

We focus on performing automated reasoning over *facts*, i.e., natural language assertions such as "`Nice people are red`", that are coherent but not necessarily grounded in reality. A *rule* is a natural language statement that is either of the form, or can be rewritten in the form, "`If P then Q`"; e.g., "`Rough, cold people are blue`" can be rewritten as "`If a person is rough and cold, then they are blue`". `P` is called the *antecedent* and `Q` is called the *consequent* of the rule. A *theory* $\mathcal{C}$ consists of facts $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ and rules $\mathcal{R} = \{r_1, r_2, \ldots, r_m\}$. We let $\mathcal{G}$ represent a *goal* that we would like to prove or disprove based on the theory. An example theory with fictional characters and rules is demonstrated in Figure 1. Based on the theory, one should prove or disprove the goal "`Eric is nice`".

3.1 Backward Chaining

Backward chaining (BC) is a strategy for reasoning that starts from the goal and recursively breaks the goal into sub-goals based on the rules that can be applied to it, until the sub-goals can be proved or disproved based on the facts or no more rules can be applied to break down the sub-goal further.

Figure 1 shows an example of BC applied to a theory to prove a goal. Initially, BC verifies if the goal can be proved or disproved based on the facts (this step is omitted from the figure). Since none of the facts directly prove or disprove the goal, BC next selects a rule that can be applied to break down the goal into sub-goals. Whether or not a rule applies to a goal is determined by an operation called *unification* in logic; Rule6 has the same consequent as the goal so the operation can be applied, but the other rules have different consequents and it cannot be applied. Using Rule6, the goal can be broken down into three sub-goals that should be proved for the goal to be proved. BC then makes recursive calls to prove each sub-goal. The algorithm continues until either a halting criterion is reached (e.g., reaching a certain depth in search), or a sub-goal can no longer be broken down (e.g., the left sub-tree under "`Eric is rough`"), or all sub-goals are proved (e.g., the right sub-tree under "`Eric is rough`").

The outcome of BC for a goal is either PROVED, DISPROVED, or UNKNOWN; e.g., its output for the goal in Figure 1 is PROVED, for "`Fred is not green?`" is DISPROVED (because it contradicts Fact3), and for "`Fred is round?`" is UNKNOWN (because the theory does not entail or contradict it).

3.2 LM Modules in LAMBADA

To enable applying BC for text-based reasoning, we introduce four LM-based modules: *Fact Check*, *Rule Selection*, *Goal Decomposition*, and *Sign Agreement*, each implemented by showing relevant in-context demonstrations to a pretrained LM (see Appendix D.3 for details). We describe these modules and then proceed to the full algorithm.

3.2.1 Fact Check

Given a set of facts $\mathcal{F}$ from the theory and a goal $\mathcal{G}$, the *Fact Check* module verifies if there exists a fact $f \in \mathcal{F}$ such that f entails $\mathcal{G}$ (in which case the goal is proved) or f entails the negation of $\mathcal{G}$ (in which case the goal is disproved). If no such fact can be found, then the truth of $\mathcal{G}$ remains unknown.

We implement *Fact Check* with two sub-modules: the first sub-module selects a fact from the set of facts that is most relevant to the goal, and the second sub-module verifies if the goal can be proved or disproved based on that fact.[1] Since the first sub-module may fail to identify the best fact on the first try, if the truth of the goal remained unknown after one try, the selected fact can be removed and the sub-modules can be called again. This process can be repeated multiple times. In our experiments, we call the two sub-modules twice.

3.2.2 Rule Selection

Given a set of rules $\mathcal{R}$ from the theory and a goal $\mathcal{G}$, the *Rule Selection* module identifies the rules $r \in \mathcal{R}$ such that the consequent of r unifies with $\mathcal{G}$. These rules are then used for decomposing the goal into sub-goals. If no such rule can be identified, then the truth of $\mathcal{G}$ remains unknown.

As we did for *Fact Check*, we implement *Rule Selection* with two sub-modules: the first sub-module identifies the consequent of each rule (independent of the goal), and the second sub-module takes the rule consequents and the goal as input and identifies which one unifies with the goal. Note that due to the recursive nature of BC, the *Rule Selection* module may be invoked multiple times during the proof of a goal. Since identifying the consequent of each rule is independent of the goal, this sub-module only needs to be called once.

[1]Note that we select only one fact because the goals and sub-goals in the datasets we work with can be proved/disproved using single facts; The two modules can be adapted to selected multiple facts if this is not the case.

Algorithm 1 LAMBADA

Input: Theory $\mathcal{C} = (\mathcal{F}, \mathcal{R})$, Goal $\mathcal{G}$, Max-Depth D

```
factCheckResult = FactCheck(G, F)
if factCheckResult ≠ UNKNOWN then
    return factCheckResult
if D == 0 then
    return UNKNOWN
R_s = RuleSelection(G, R)
for r ∈ Rerank(R_s) do
    G = GoalDecomposition(r, G)
    if ProveSubgoals(C, G, D) then
        if SignAgreement(r, G) then
            return PROVED
        else
            return DISPROVED
return UNKNOWN
```

3.2.3 Goal Decomposition

Given a rule r and a goal $\mathcal{G}$ such that the consequent of r unifies with $\mathcal{G}$, the *Goal Decomposition* module identifies the sub-goals that need to be proved in order for $\mathcal{G}$ to be proved or disproved. The sub-goals are identified based on the antecedent of r.

3.2.4 Sign Agreement

In the case where we succeed in proving the antecedent of r, whether the goal is proved or disproved depends on whether the sign of the goal agrees or disagrees with the sign of the consequent of r. For instance, in Figure 1, for the goal "`Eric is nice.`", since the sign of the goal agrees with the sign of the consequent of Rule6 and the antecedent of the rule is proved, we conclude that the goal is proved. However, if Rule6 was "`[...] is not going to be a nice individual.`", then the sign of the goal would disagree with the sign of the consequent and so we would conclude that the goal is disproved. This motivates the fourth module, *Sign Agreement*, described below.

Given a rule r and a goal $\mathcal{G}$, the *Sign Agreement* module verifies if the sign of the consequent of r agrees or disagrees with the sign of the goal or not.

3.3 The LAMBADA Algorithm

Algorithm 1 provides a high-level description of how the four LM modules described earlier can be integrated with BC to enable text-based logical reasoning (the function calls corresponding to LM modules are color-coded).

LAMBADA can be understood as a depth-first

Algorithm 2 ProveSubgoals

Input: Theory $\mathcal{C} = (\mathcal{F}, \mathcal{R})$, Sub-Goals $\mathbf{G}$, Max-Depth D

```
for 𝒢 in G do
    result = LAMBADA(𝒞, 𝒢, D-1)
    if result ≠ PROVED then
        return False # Assuming conjunction
return True
```

search algorithm over the facts and the rules. It takes as input a theory $\mathcal{C} = (\mathcal{F}, \mathcal{R})$, a goal $\mathcal{G}$, and a depth D that defines a halting criterion for the algorithm based on the maximum allowed depth for the search. The search depth is a natural halting criterion corresponding to the maximum number of reasoning hops required for answering questions.

Initially, the algorithm uses the *Fact Check* module to check if $\mathcal{G}$ can be proved or disproved using the facts. If this is the case, then the algorithm stops and returns the result (PROVED or DISPROVED).

If $\mathcal{G}$ cannot be proved or disproved, then the algorithm checks the depth D: if $D = 0$, then the algorithm stops and returns UNKNOWN indicating that $\mathcal{G}$ could not be proved or disproved. Otherwise, the algorithm proceeds with applying rules.

The *Rule Selection* module is used to identify the rules $\mathcal{R}_s$ from $\mathcal{R}$ whose consequent unifies with $\mathcal{G}$. Once the set $\mathcal{R}_s$ is identified, if LAMBADA can start with the rules that have a higher chance of succeeding at (dis)proving the goal, it can save computations and be less error-prone. Therefore, we include a *Rerank* function in LAMBADA. Based on the intuition that shorter rules are likely to have fewer sub-goals (hence a higher chance of success), we start the search from shorter rules and proceed to longer rules if the shorter ones fail. We leave more sophisticated ranking strategies as future work.

For each selected rule, the algorithm uses the *Goal Decomposition* module to decompose $\mathcal{G}$ into a set of sub-goals $\mathbf{G}$ that need to be proved and checks whether those sub-goals can be proved by making recursive calls to the algorithm (with reduced depth). If the sub-goals can be proved, then the algorithm uses the *Sign Agreement* module to check whether the sign of the rule consequent agrees or disagrees with the sign of $\mathcal{G}$. If it does, then the algorithm returns PROVED and otherwise DISPROVED. If there is no rule for which the sub-goals can be proved, then UNKNOWN is returned.

During a proof, LAMBADA may be called multiple times with the same theory and goal; in Appendix A we explain how cycles and redundant computations can be avoided using a cache.

4 Experimental Setup

We describe our baselines and datasets here, and provide further implementation details in Appendix D. Unless stated otherwise, all experiments are based on the PaLM 540B model (Chowdhery et al., 2022).

4.1 Baselines

We compare against the following two baselines.

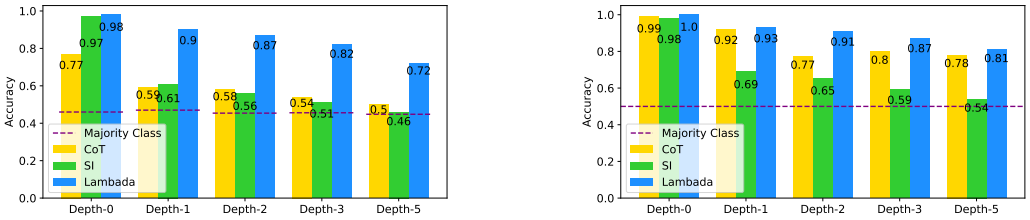
Chain-of-Thought (CoT) (Wei et al., 2022) is a popular neural approach based on demonstrating chains of inference to the LM within the in-context prompt. In addition to the few-shot demonstrations in `<INPUT>/<LABEL>` format in typical in-context learning settings, in CoT, an intermediate explanation for the label is also provided (`<INPUT>/<EXPLANATION>/<LABEL>`). In our work, the explanation corresponds to the proof.

Selection-Inference (SI) (Creswell et al., 2023) is a strong modular reasoning approach based on forward chaining. SI contains two modules: (1) *selection*, which, guided by the goal, selects a subset of the facts and rules from which new conclusions can be derived toward proving the goal, and (2) *inference*, which takes the selected facts and rules and derives a new conclusion. The two modules are called iteratively, each time producing a single conclusion that is added back to the theory before the next iteration. The iterations continue until a halting criterion is met (a fixed number of steps in Creswell et al. 2023).

4.2 Datasets

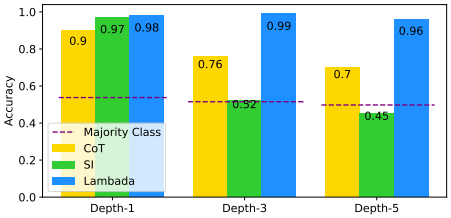
We experiment with challenging deductive logical reasoning datasets outlined below.

ProofWriter (Tafjord et al., 2021) is a commonly used synthetic dataset for testing logical reasoning when facts and rules are expressed in naturalistic text. It contains two subsets: an open-world assumption (OWA) subset and a closed-world assumption (CWA) subset. In this paper, we use the OWA subset. Each example is a (*theory, goal*) pair and the label is one of {PROVED, DISPROVED, UNKNOWN} where UNKNOWN indicates that the goal can neither be proved nor disproved. The dataset has five parts, each part requiring 0, ≤ 1, ≤ 2, ≤ 3 and ≤ 5 hops of reasoning, respectively. We report two sets of results on this dataset: (1)

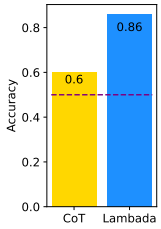


(a) ProofWriter-PUD

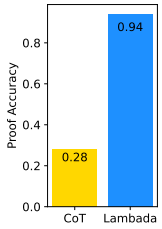
(b) ProofWriter-PD



(c) PrOntoQA



(d) ParaRules



(e) ProofWr. (d5)

Figure 2: Prediction accuracy results on (a) ProofWriter-PUD (b) ProofWriter-PD, (c) PrOntoQA, and (d) ParaRules datasets. (e) The proof accuracy of CoT and LAMBADA on ProofWriter (Depth-5) for a set of randomly sampled examples for which the models correctly predicted if the goal can be proved or disproved.

with examples labeled UNKNOWN removed (for compatibility with previous work), and (2) with all three labels. Note that intermediate proof chains from ProofWriter are not used by our models in making predictions. For both cases, due to the cost of inference, we used the first 1000 examples in the test set. Hereafter, we refer to these two subsets as *ProofWriter-PD* and *ProofWriter-PUD*.

PrOntoQA (Saparov and He, 2023) is a synthetic dataset created to analyze the capacity of LM-based approaches for logical reasoning. Compared to ProofWriter, PrOntoQA has lower natural language diversity and less 1 fact/rule variations (e.g., no conjunctions). However, the search traces typically contain multiple paths with only one of them leading to the proof, thus enabling testing the proof planning of different models. This dataset has multiple versions; we use the *fictional characters* version, which is one of the hardest versions according to Saparov and He (2023). Similarly to ProofWriter, each version of PrOntoQA is divided into different parts depending on the depth of reasoning chains required (1, 3, and 5 hops).

ParaRules (Tafjord et al., 2021) is a version of ProofWriter where the synthetically generated sentences in the theory are rewritten by crowdworkers to increase diversity and naturalness of the text. This lets us move beyond evaluating reasoning with templatic expressions, which is a key limitation of the other datasets. Each fact in ParaRules may be a combination of several sub-facts (see Fig. 1 for an example). The examples require proof depths of up to 5 and the label can be PROVED, DISPROVED, or UNKNOWN. We found some minor quality issues in ParaRules; we manually verified and fixed the first 500 examples of the test set (see Appendix D.2) and used this set for evaluation.

5 Results

We now describe the results and compare LAMBADA and the baselines in detail.

5.1 Label Prediction Accuracy

The results are reported in Figure 2, (a)–(d).[2] LAMBADA significantly outperforms the baselines, especially on ProofWriter-PUD which contains UNKNOWN labels (44% relative improvement compared to CoT and 56% compared to SI on Depth-5), the higher depths of PrOntoQA (37% relative improvement compared to CoT and 113% compared to SI on Depth-5), and the ParaRules dataset (43% relative improvement compared to CoT). These results overall show the merit of LAMBADA for logical reasoning. We highlight that the reasoning capacity of LAMBADA robustly generalizes to more naturalistic expressions, as demonstrated by the high accuracy on ParaRules, which is exactly

[2]Due to the low performance of SI on ProofWriter and PrOntoQA and its high number of LM calls (see Figure 7), we only compared LAMBADA against CoT for ParaRules.

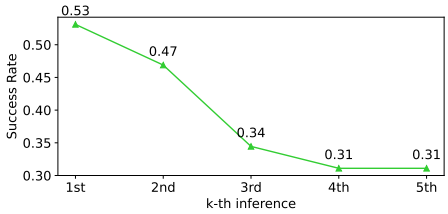


Figure 3: The success rate of the k-th inference of SI on PrOntoQA (Depth-5) for different values of k. As k increases, the size of the input theory becomes larger and the success rate decreases.

the desired outcome of combining the strengths of an LM and a symbolic reasoning algorithm.

The results in Figure 2(a) reveal a shortcoming of the CoT approach in dealing with UNKNOWN labels. That is, unlike the examples for which the label is PROVED or DISPROVED, there is no natural chain of thought for the examples whose labels are UNKNOWN. Nevertheless, the performance of CoT is competitive for the ProofWriter-PD dataset, and the accuracy does not diminish substantially with increasing depth. We investigate the reason for this behaviour of CoT in the next section.

5.2 Proof Accuracy

To understand the reason behind the high accuracy of CoT on higher depths of ProofWriter-PD, we randomly selected 50 examples from Depth-5 of the dataset where CoT predicted the label correctly, and manually verified if the proof chain is correct or not. For comparison, we also manually verified the proofs generated by LAMBADA following a similar procedure. The results are reported in Figure 2(e).

While LAMBADA mostly produces correct chains, CoT produces correct chains only for 28% of the examples. We find that hallucination is the main source of error (48% of the examples; see Appendix B.2 for other prominent failure modes). The hallucinated facts and rules mostly resulted in shortcuts to the correct answer. This hints at the possibility of spurious correlations in ProofWriter-PD that can be exploited by CoT (see Appendix B.2, Figure 10 for examples). This result is consistent with previous work showing that when LMs are asked to solve logical reasoning end-to-end, they rely on spurious correlations (Zhang et al., 2022b). Note that for modular approaches like SI and LAMBADA, the intermediate modules are impervious to the spurious correlations between the input and the label and do not suffer from this issue.

5.3 Forward vs. Backward Chaining

As previously explained, SI is based on forward chaining and its selection module requires a combinatorial search to find the right subset of facts and rules (see Appendix C), and the search space becomes progressively larger in each iteration of the algorithm as new inferences are added to the theory. To verify whether the increase in the search space makes forward chaining progressively harder, we measured the success rate of the k-th inference of SI for different values of k on Depth-5 of PrOntoQA (see Appendix B.3 for details). From the results in Figure 3, we can see that the success rate indeed decreases in the later inferences of the model, where the size of the input theory is larger and therefore a larger space needs to be searched to find the right combination of facts and rules. Note that none of the components in LAMBADA require selecting a *subset*, hence no combinatorial search is required (see Appendix C for more details).

SI also suffers from inferring redundant facts. Figure 4 reports the number of unique inferences from SI for the examples in ProofWriter-PD (Depth-5) where SI incorrectly predicted UNKNOWN (i.e., examples where a proof exists but SI failed to find it). The result shows that SI inferences contained no redundant facts only 29% of the time; in 7% of the cases, all 5 inferred facts were identical, and in another 10%, only two unique inferences were made. This shows that SI, and maybe more generally forward-chaining approaches, suffer from redundant inference.

SI also over-predicts DISPROVED in the binary case and UNKNOWN in the three-way classification case (see Appendix B.4), performing even worse than the majority class for Depth-5 of PrOntoQA which has more PROVED labels than DISPROVED.

These results, together with Figure 2, show that backward chaining (which is the backbone of reasoning in LAMBADA) is a better choice compared to forward chaining (the backbone in SI).

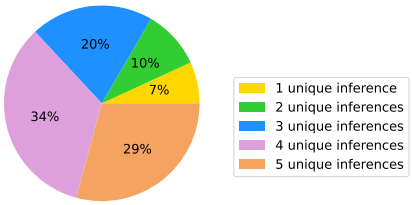
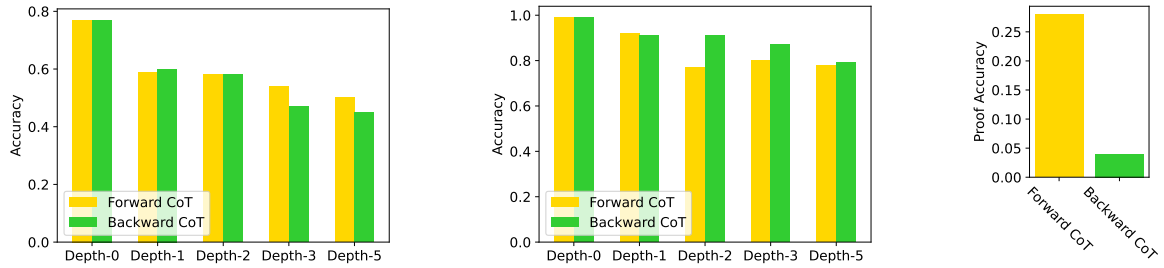


Figure 4: Number of unique inferences generated by SI for Depth-5 of ProofWriter-PUD when selection and inference modules are called five times.



(a) (b) (c)

Figure 5: Prediction accuracy results on (a) ProofWriter-PUD and (b) ProofWriter-PD with forward and backward CoT. (c) compares the proof accuracy of forward and backward CoT on ProofWriter (Depth-5) for a set of randomly sampled examples for which the models correctly predicted the proof label.

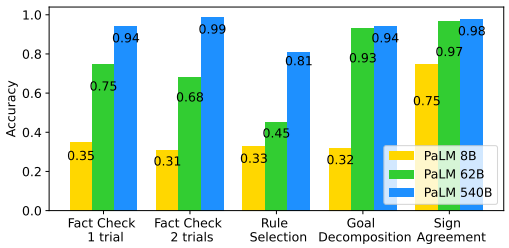


Figure 6: ProofWriter (val) performance of modules in LAMBADA in isolation, for different LM sizes.

5.4 Does Backward CoT Suffice?

Our results may raise the question of whether it is enough to directly incorporate the steps of backward chaining into CoT prompts, or if modularity (as in LAMBADA) is also needed. To answer this question, we experiment with a backward version of CoT where the proofs are written in the backward direction from the goal to the premises. The label accuracies are presented in Figure 5(a)–(b) for ProofWriter-PUD and ProofWriter-PD, and their proof accuracy on ProofWriter-PD (Depth-5) in Figure 5(c). The label accuracy of forward and backward CoT are comparable, but forward CoT leads to better performance on PUD and backward CoT leads to better performance on PD. For proof accuracy, however, we see a clear difference between the two versions where backward CoT produces substantially lower quality proofs compared to forward chaining. This result is consistent with the observations of Gontier et al. (2020) for fine-tuned LMs.

The above results show that a modular formulation (as in LAMBADA) is key to successful logical reasoning and simply providing CoT in the backward direction does not suffice. We note, however, that future work can use the traces of our model to finetune (smaller) language models (e.g., Zelikman et al. 2022), or use the traces as training data in future language models to improve their performance with CoT prompting.

Taking the label and proof accuracy results together, there is also a potential that backward CoT models are more heavily relying on spurious correlations for the PD case where backward CoT outperformed CoT, as backward CoT achieves a similar label accuracy as forward CoT but with a much lower proof accuracy.

5.5 Qualitative Analysis

In Figure 1, we show the search trace created by LAMBADA for an example from ParaRules, where the answer was predicted correctly. From the figure, one can see how backward chaining helps LAMBADA effectively search and create the reasoning chain and how the LM helps fact checking, rule selection, goal decomposition, and sign agreement checking. In Appendix B.1, we include an example that has a much larger search trace.

5.6 Individual Module Analysis

To understand which components in LAMBADA are responsible for the failure cases, we computed the individual accuracy of the four modules described in Section 3. For this purpose, we created four datasets from the validation set of ProofWriter, each measuring only the performance of one module in isolation (see Appendix D.1 for details).

Based on the results of the PaLM 540B model in Figure 6, *Rule Selection* is the lowest performing module followed by *Goal Decomposition*. It is possible that the *Rule Selection* module (partially) fails for some examples but LAMBADA still arrives at

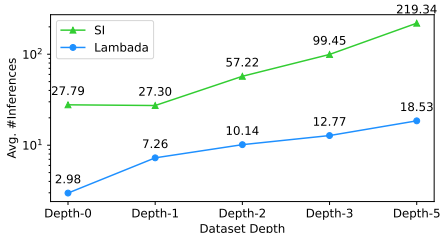


Figure 7: Comparing LAMBADA and SI w.r.t. the average number of inference calls they make per example for different subsets of the ProofWriter-PUD dataset.

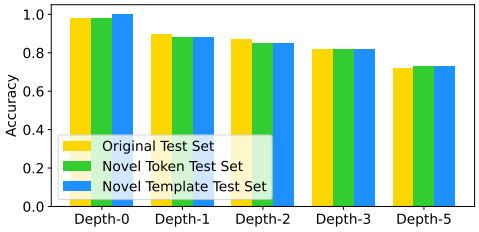


Figure 8: The performance of LAMBADA on ProofWriter-PUD for the original, novel token, and novel template test sets.

the correct conclusion and proof (e.g., if in Figure 1 the third call to *Rule Selection* only returned Rule5). For *Fact Check*, when we allow the model to only select one fact, the accuracy is 0.94 but when we allow the model to select two facts, the accuracy is near perfect. The *Sign Agreement* module also shows near-perfect accuracy.

5.7 The Role of Scale

We repeat the experiment from Section 5.6 with PaLM 62B and 8B to examine the effect of LM scale on LAMBADA. According to the results in Figure 6, when we use PaLM 62B, the performance of the *Goal Decomposition* and *Sign Agreement* modules remain comparable, but the performance for the *Fact Check* and *Rule Selection* modules drop substantially. Unlike the first two modules, the second two rely on a one-to-many comparison between the goal and each of the facts/rules which may require a larger model capacity. Moreover, we observe that in PaLM 8B, the accuracy for all components drops significantly, in some cases becoming close to random prediction.

We argue that the extent to which the higher-level reasoning algorithm breaks the problem into sub-problems should be dependent on the scale and power of the base LMs. If smaller LMs are used, then one may need finer-grained problem decomposition (e.g., further decomposing the one-to-many comparisons in the selection module). And as LMs become larger and stronger in the future, one could rely on them to solve problems with a coarser-grained decomposition of the problem.

5.8 Number of Inference Calls

Another advantage of LAMBADA is its efficiency compared to other approaches that require multiple LM inference calls per example such as SI. In Figure 7, we compare the average number of LM calls per example, for different depths of ProofWriter-PUD. LAMBADA requires much fewer calls compared to SI, especially at higher depths: for Depth-1, LAMBADA requires 3.8x fewer calls whereas for Depth-5 it requires 11.8x fewer calls.

5.9 Lexical Robustness

To analyze the lexical sensitivity of LAMBADA, we modified the test set of ProofWriter-PUD by replacing various lexical items (names, adjectives, and verbs) with novel tokens and the rule templates with novel ones. We then compared the performance of LAMBADA on the original and the modified test sets using the same few-shot examples. The details of the modifications are in Appendix B.5. As can be seen in Figure 8, the performance of LAMBADA remains almost unchanged, demonstrating robustness to lexical and templatic variations.

6 Conclusion and Future Directions

We developed LAMBADA, an algorithm for deductive logical reasoning with natural language that combines the capacity of LMs to handle naturalistic text input with the backward chaining algorithm for robust symbolic reasoning. We showed that LAMBADA achieves significant improvements over competitive approaches on challenging benchmarks, both in terms of label accuracy (predicting if a statement can be proved or disproved based on a theory) and proof accuracy. Importantly, this improvement was also observed in a dataset that expresses the theory in more naturalistic expressions, clearly illustrating the benefit of combining an LM with reasoning modules. We also demonstrated the query efficiency and lexical robustness of LAMBADA. Although in this paper we only experiment with formal reasoning problems and datasets, we believe our key insight on the efficacy of backward, goal-directed reasoning with LMs has broader implications and can be adapted to other NLP tasks where multi-step inference is required.

Limitations

We identify some limitations and risks with our current work that can be addressed in future work.

- The current work is mainly applicable to logical entailment problems, where one needs to solve a classification problem of whether a goal can be proved, disproved, or neither proved nor disproved based on a theory. Future work can extend LAMBADA to non-classification cases, e.g., where one needs to apply logical reasoning to answer questions such as "`What color is Fiona?`".
- The current work assumes all the rules are given as input and the rule set is small enough to be included in the prompt. Future work can extend LAMBADA to the cases where not all the rules are provided as input and part of the knowledge has to come from the LM itself, as well as the case where not all the rules can be included in the prompt due to the limitation in the prompt length.
- The current work is limited to deductive reasoning with the *modus ponens* rule; future work can expand the applicability of LAMBADA on datasets with other types of rules such as proof by contradiction, disjunction elimination, etc.
- The calls made to the LM modules in LAMBADA are dependent on the value from the previous call. That is, we need to wait for the results from one call before we decide what the next call must be. Since making batch calls to the LMs is typically easier and faster, future work can find ways to implement LAMBADA with batch LM calls.
- While we showed that LAMBADA is more efficient than SI in terms of the number of inference calls it makes to the LM, it still requires many more calls to the LM compared to approaches such as CoT, hence increasing the required computation and cost.

References

Cem Anil, Yuhuai Wu, Anders Andreassen, Aitor Lewkowycz, Vedant Misra, Vinay Ramasesh, Ambrose Slone, Guy Gur-Ari, Ethan Dyer, and Behnam Neyshabur. 2022. Exploring length generalization in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 38546–38556. Curran Associates, Inc.

Gregor Betz, Christian Voigt, and Kyle Richardson. 2021. Critical thinking for language models. In *Proceedings of the 14th International Conference on Computational Semantics (IWCS)*, pages 63–75, Groningen, The Netherlands (online). Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling language modeling with pathways. *arXiv:2204.02311*.

Peter Clark, Oyvind Tafjord, and Kyle Richardson. 2021. Transformers as soft reasoners over language. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, IJCAI'20.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv:2110.14168*.

Antonia Creswell and Murray Shanahan. 2022. Faithful reasoning using large language models. *arXiv:2208.14271*.

Antonia Creswell, Murray Shanahan, and Irina Higgins. 2023. Selection-inference: Exploiting large language models for interpretable logical reasoning. In *The Eleventh International Conference on Learning Representations*.

Ido Dagan, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. Recognizing textual entailment: Models and applications. *Synthesis Lectures on Human Language Technologies*, 6(4):1–220.

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2021. Explaining answers with entailment trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7358–7370, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. 2022. Successive prompting for decomposing complex questions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1251–1265, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Artur d'Avila Garcez and Luis C Lamb. 2020. Neurosymbolic ai: the 3rd wave. *arXiv:2012.05876*.

Nicolas Gontier, Koustuv Sinha, Siva Reddy, and Chris Pal. 2020. Measuring systematic generalization in neural proof generation with transformers. In *Advances in Neural Information Processing Systems*, volume 33, pages 22231–22242. Curran Associates, Inc.

Simeng Han, Hailey Schoelkopf, Yilun Zhao, Zhenting Qi, Martin Riddell, Luke Benson, Lucy Sun, Ekaterina Zubova, Yujie Qiao, Matthew Burtell, et al. 2022. FOLIO: Natural language reasoning with first-order logic. *arXiv:2209.00840*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran.

Carl Hewitt. 1969. Planner: A language for proving theorems in robots. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, page 295–301, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Jie Huang and Kevin Chen-Chuan Chang. 2022. Towards reasoning in large language models: A survey. *arXiv:2212.10403*.

Harsh Jhamtani and Peter Clark. 2020. Learning to explain: Datasets and models for identifying valid reasoning chains in multihop question-answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 137–150, Online. Association for Computational Linguistics.

Nora Kassner, Benno Krojer, and Hinrich Schütze. 2020. Are pretrained language models symbolic reasoners over knowledge? In *Proceedings of the 24th Conference on Computational Natural Language Learning*, pages 552–564, Online. Association for Computational Linguistics.

Mehran Kazemi, Sid Mittal, and Deepak Ramachandran. 2023. Understanding finetuning for factual knowledge extraction from language models. *arXiv:2301.11293*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*.

Gary Marcus. 2020. The next decade in AI: four steps towards robust artificial intelligence. *arXiv:2002.06177*.

John McCarthy. 1959. Programs with common sense. In *Proceedings of the Teddington Conference on the Mechanization of Thought Processes*, pages 75–91, London. Her Majesty's Stationary Office.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2022. Show your work: Scratchpads for intermediate computation with language models. In *Deep Learning for Code Workshop*.

David L Poole and Alan K Mackworth. 2010. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press.

Jack W. Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, Eliza Rutherford, Tom Hennigan, Jacob Menick, Albin Cassirer, Richard Powell, George van den Driessche, Lisa Anne Hendricks, Maribeth Rauh, Po-Sen Huang, Amelia Glaese, Johannes Welbl, Sumanth Dathathri, Saffron Huang, Jonathan Uesato, John Mellor, Irina Higgins, Antonia Creswell, Nat McAleese, Amy Wu, Erich Elsen, Siddhant Jayakumar, Elena Buchatskaya, David Budden, Esme Sutherland, Karen Simonyan, Michela Paganini, Laurent Sifre, Lena Martens, Xiang Lorraine Li, Adhiguna Kuncoro, Aida Nematzadeh, Elena Gribovskaya, Domenic Donato, Angeliki Lazaridou, Arthur Mensch, Jean-Baptiste

Lespiau, Maria Tsimpoukelli, Nikolai Grigorev, Doug Fritz, Thibault Sottiaux, Mantas Pajarskas, Toby Pohlen, Zhitao Gong, Daniel Toyama, Cyprien de Masson d'Autume, Yujia Li, Tayfun Terzi, Vladimir Mikulik, Igor Babuschkin, Aidan Clark, Diego de Las Casas, Aurelia Guy, Chris Jones, James Bradbury, Matthew Johnson, Blake Hechtman, Laura Weidinger, Iason Gabriel, William Isaac, Ed Lockhart, Simon Osindero, Laura Rimell, Chris Dyer, Oriol Vinyals, Kareem Ayoub, Jeff Stanway, Lorrayne Bennett, Demis Hassabis, Koray Kavukcuoglu, and Geoffrey Irving. 2021. Scaling language models: Methods, analysis & insights from training Gopher. *arXiv:2112.11446*.

Mohammed Saeed, Naser Ahmadi, Preslav Nakov, and Paolo Papotti. 2021. RuleBERT: Teaching soft rules to pre-trained language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1460–1476, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Abulhair Saparov and He He. 2023. Language models are greedy reasoners: A systematic formal analysis of chain-of-thought. In *The Eleventh International Conference on Learning Representations*.

Imanol Schlag, Sainbayar Sukhbaatar, Asli Celikyilmaz, Wen-tau Yih, Jason Weston, Jürgen Schmidhuber, and Xian Li. 2023. Large language model programs. *arXiv preprint arXiv:2305.05364*.

Viktor Schlegel, Kamen Pavlov, and Ian Pratt-Hartmann. 2022. Can transformers reason in fragments of natural language? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11184–11199, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. 2021. Generate & rank: A multi-task framework for math word problems. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2269–2279, Punta Cana, Dominican Republic. Association for Computational Linguistics.

Zayne Sprague, Kaj Bostrom, Swarat Chaudhuri, and Greg Durrett. 2022. Natural language deduction with incomplete information. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8230–8258, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv:2210.09261*.

Oyvind Tafjord, Bhavana Dalvi, and Peter Clark. 2021. ProofWriter: Generating implications, proofs, and abductive statements over natural language. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3621–3634, Online. Association for Computational Linguistics.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. 2022. Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.

Boshi Wang, Xiang Deng, and Huan Sun. 2022. Iteratively prompt pre-trained language models for chain of thought. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2714–2730, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. STaR: Bootstrapping reasoning with reasoning. In *Advances in Neural Information Processing Systems*, volume 35, pages 15476–15488. Curran Associates, Inc.

Hanlin Zhang, Ziyang Li, Jiani Huang, Mayur Naik, and Eric Xing. 2022a. Improved logical reasoning of language models via differentiable symbolic programming. In *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*.

Honghua Zhang, Liunian Harold Li, Tao Meng, Kai-Wei Chang, and Guy Van den Broeck. 2022b. On the paradox of learning to reason from data. *arXiv:2205.11502*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc V Le, and Ed H. Chi. 2023. Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

Hattie Zhou, Azade Nova, Hugo Larochelle, Aaron Courville, Behnam Neyshabur, and Hanie Sedghi. 2022. Teaching algorithmic reasoning via in-context learning. *arXiv:2211.09066*.

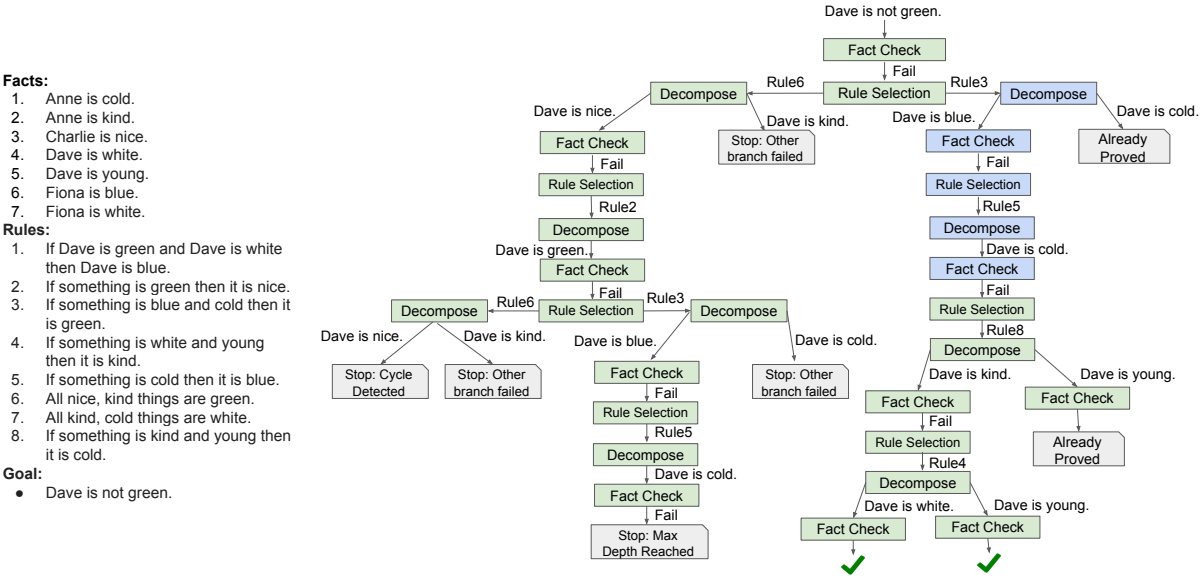


Figure 9: The search trace of LAMBADA on an example from ProofWriter with depth=5 where the answer was predicted correctly. The sign agreement module has been omitted for brevity. The modules color-coded with blue represent the calls where the module retrieved the value from the cache instead of calling the LM.

A Caching and Avoiding Loops for LAMBADA

Since LAMBADA is a recursive algorithm, during the proof of an example Algorithm 1 may be called with the same goal multiple times. For instance, consider the goal "`Eric is nice`" for the theory in Figure 1. Applying Rule6 breaks the goal into three sub-goals. The first one is "`Eric is big`" which is proved using the *Fact Check* module. For the second sub-goal, Rule3 is used to compose it into three sub-goals the first of which we have proved before. Since we have already proved this sub-goal, we can save a *Fact Check* call if we cache previous results.

Note that the result of a call to LAMBADA can be different depending on the input max depth. For example, the algorithm may return UNKNOWN when called for the theory and goal in Figure 1 with max depth 0, and return PROVED when called with max depth 3. Specifically, if we can prove/disprove a goal at depth d, we can conclude that it can be proved/disproved at depths $\geq d$ as well and we can get the value from the cache. Moreover, if the algorithm returns UNKNOWN for a goal at depth d, we can conclude that it will also return UNKNOWN at depths $< d$. Therefore, if the algorithm is called for a theory and goal at depth d, we also check other depths to see if we have the results for other depths that apply to this case. Besides having a cache for the entire algorithm that avoids redundant computations when the truth of a goal has been previously computed for a theory, each individual module can also have its own cache as it is possible that the module is called for the same theory and goal. We show one such example in Figure 9 (to be discussed in Section B).

LAMBADA may sometimes run into loops. For example, to prove a (sub-)goal "`Fiona is round?`", after recursively identifying rules that unify with it and decomposing it into sub-goals, the algorithm may arrive at a point where it needs to prove the "`Fiona is round?`" sub-goal, which is equivalent to the initial goal. To avoid such loops, for each path in the proof trace, we keep track of the (sub-)goals that are to be proved and stop further exploring that branch of the search trace when a loop is identified.

Note that in Algorithm 1, for clarity of the algorithm we did not include the caching and loop avoidance operations. Also note that caching and loop avoidance mainly help with reducing the number of inference calls.

B Additional Results and Analyses

In this section, we provide some more in-depth qualitative and quantitative analysis of the results from our model and the baselines.

B.1 Qualitative Analysis

In Figure 9, we provide the search trace of LAMBADA for an example in ProofWriter (Depth-5) for

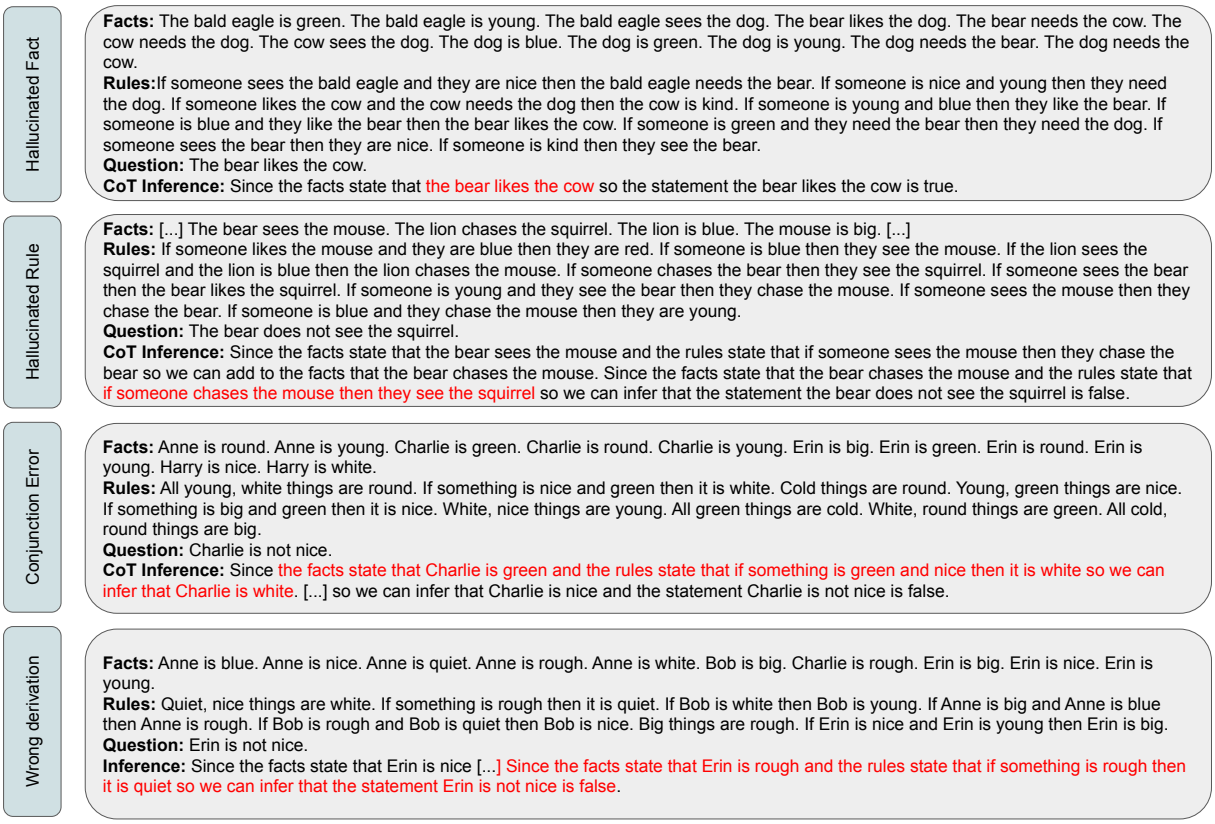


Figure 10: Examples of wrong CoT proof chains from four different categories. The erroneous part is marked in red.

which LAMBADA correctly predicted that the goal is disproved based on the theory. We deliberately selected an example with a large search trace to demonstrate the various aspects of LAMBADA.

LAMBADA starts by calling the *Fact Check* module on the goal which fails to prove or disprove it. So *Rule Selection* is called which identifies two rules that can be applied: Rule3 and Rule6. Since Rule6 is shorter, the reranker ranks it higher; LAMBADA starts with this rule and calls the *Goal Decomposition* module which breaks the goal into two sub-goals: "`Dave is nice.`" and "`Dave is kind.`". Starting with the first sub-goal, *Face Check* fails on it so *Rule Selection* is called which selects Rule2 and *Goal Decomposition* decomposes the sub-goal into "`Dave is green.`".

Note that if the cycle checking was smart enough to understand that this sub-goal is the negation of the root goal, we could stop further searching this branch. However, we currently only do cycle matching for exact matches so the algorithm continues the search trace.

Fact Check fails again so *Rule Selection* is called which selects Rule3 and Rule6 again, and since Rule6 is shorter the algorithm continues with that rule. *Goal Decomposition* breaks the sub-goal into "`Dave is nice.`" and "`Dave is kind.`". Considering the first sub-goal, the algorithm identifies a cycle and stops the search. The second sub-goal is also ignored as there is a conjunction between the sub-goals.

The algorithm then continues with calling *Goal Decomposition* for Rule3 which breaks the sub-goal into "`Dave is blue.`" and "`Dave is cold.`". Starting with the first sub-goal, since *Fact Check* fails the algorithm calls *Rule Selection* which selects Rule5 and *Goal Decomposition* breaks the sub-goal into "`Dave is cold.`". *Face Check* fails on this sub-goal and since the maximum depth is reached, the algorithm stops expanding this branch. Moreover, the branch for "`Dave is cold.`" is no longer pursued because there was a conjunction between the sub-goals and one of them failed.

Moving on to the right branch in Figure 9, the algorithm calls the *Goal Decomposition* module for the goal and Rule3. Since we have previously computed it, the sub-goals "`Dave is blue.`" and "`Dave is cold.`" are returned from the cache. *Fact Check* is called on "`Dave is blue.`" and since it has been computed before, the result (fail-

ure) is retrieved from the cache. The *Rule Selection* module is called, where the result (Rule5) is again retrieved from the cache. *Goal Decomposition* is then called and the sub-goal "`Dave is cold.`" is retrieved from the cache. *Fact Check* fails again (retrieved from the cache), *Rule Selection* selects Rule8 and *Goal Decomposition* produces two sub-goals: "`Dave is kind.`" and "`Dave is young.`". For "`Dave is kind.`", *Fact Check* fails, *Rule Selection* selects Rule4 and *Goal Decomposition* produces two sub-goals: "`Dave is white.`" and "`Dave is young.`". For both of these sub-goals, *Fact Check* succeeds in proving them. The algorithm then also checks "`Dave is young.`" for the right branch, but since this sub-goal has already been proved, it just gets the result from the cache. The algorithm then checks "`Dave is cold.`" for the rightmost branch, but since this sub-goal has already been proved, it just gets the result from the cache.

The model also calls the *Sign Agreement* module for rules on the right branch (not shown in the Figure) and finds out that the sign of the rules and the sub-goals agree for all cases, except for the very first rule selected (Rule3) so it correctly concludes that the goal is disproved.

B.2 Further Analysis of CoT

In Figure 2(e), we observed that CoT mostly produces wrong proof chains even when the predicted label is correct. Through manually analyzing 50 examples for which CoT predicted the correct label, we identified three dominant reasons for the chains being wrong: 1- hallucinating rules or facts, 2- not understanding conjunction, and 3- making invalid derivations. In Figure 10, we show failure examples from each category. Notice that, e.g., in the example with a hallucinated rule, CoT relies on a rule "`if someone chases the mouse then they see the squirrel`" which not only does not appear in the provided set of rules, but cannot even be derived with a combination of the rules.

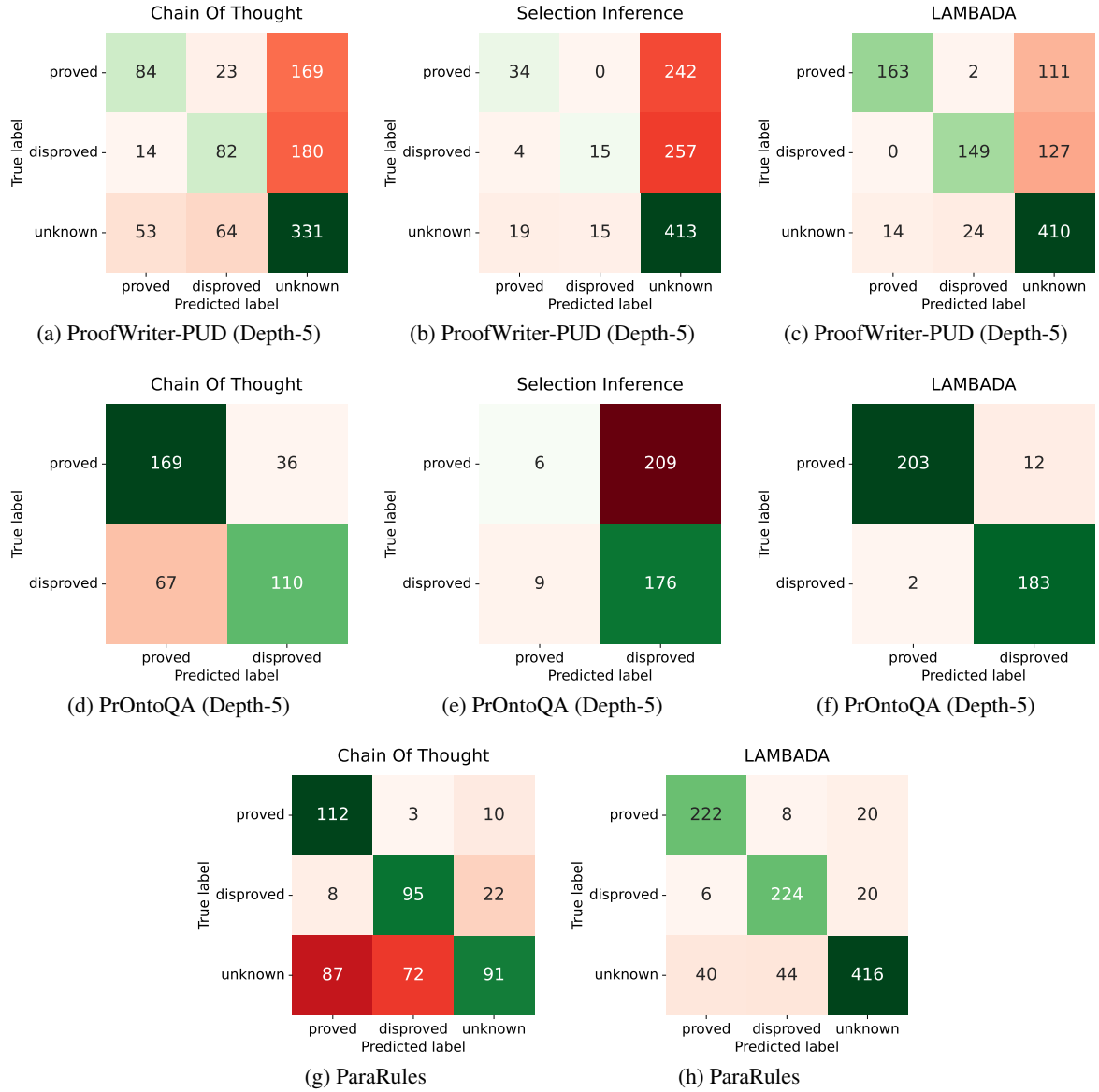
The high label accuracy of CoT and its low proof accuracy on ProofWriter-PD hint at the possibility of spurious biases that can be exploited by CoT. For example, we found that in 9.2% of the examples which require 1+ reasoning hops, the consequent of one of the rules in the theory is the same as the goal to be proved, and for 98.9% of these examples the label is PROVED. In several of these examples, CoT simply concluded that the goal can be proved in 0 hops based on a hallucinated fact. Moreover, the existence of the word "`not`" in the goal is highly predictive of the label: goals having "`not`" are mostly DISPROVED and goals not having "`not`" are mostly PROVED. The PUD case solves the latter issue to a large extent as the label for a good portion of the examples with or without "`not`" in UNKNOWN. The spurious correlations also explain the fluctuations in the CoT performance across different depths, as the performance depends on how much those correlations appear in the few-shot demonstrations.

We reiterate that for SI and LAMBADA, such spurious correlations between the input and the label cannot be exploited because the intermediate modules are impervious to the correlations between the input and the label.

B.3 Forward Chaining Becomes Progressively More Difficult

Algorithms such as SI that are based on forward chaining require a combinatorial search of the theory to find the right subset of facts and rules in each step of the reasoning. The search space becomes progressively larger as the algorithm makes new inferences and those inferences are added back to the theory. For example, if the initial size of the theory (i.e. the number of facts plus the number of rules) is $|\mathcal{C}|$, when making the k-th inference the size of the theory is $|\mathcal{C}| + k - 1$.

Conceptually, as the model produces more inferences, the distance to the goal (in terms of the number of hops remaining between the goal and the facts) should reduce and so the later inferences should be more accurate. However, we hypothesize that the increase in the size of the theory (and hence the size of the search space) may result in lower success rates in the later inferences of the SI model. To verify this experimentally, we further analyzed the results of SI on depth-5 of PrOntoQA as follows. We extracted the subset of examples where the label was PROVED but SI failed to find a proof (these are examples where at least one of the inferences is not on the proof chain). Then, as a proxy for measuring the responsibility of the k-th inference of the model for the failure, we measured the percentage of times the k-th inference was on the proof chain (the proof chain for each test example is provided as part of the dataset). Notice that it is possible that, e.g., the first inference is not on the proof chain, but the rest of the inferences



(a) ProofWriter-PUD (Depth-5) (b) ProofWriter-PUD (Depth-5) (c) ProofWriter-PUD (Depth-5)

(d) PrOntoQA (Depth-5) (e) PrOntoQA (Depth-5) (f) PrOntoQA (Depth-5)

(g) ParaRules (h) ParaRules

Figure 11: Confusion matrices.

are. The results are reported in Figure 3 in the main text. The results show that the chance of producing inferences that are on the proof chain progressively decreases in the later inferences of the model where the size of the input theory (and hence the search space) is larger.

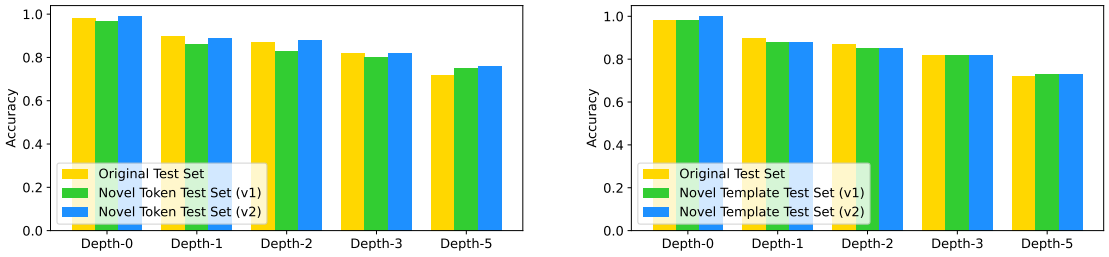
B.4 Confusion Matrices

We reported the overall model accuracies in the main text. Here, we report finer-grained confusion matrices that help better understand the biases of the model. Figure 11 reports the confusion matrices for our datasets. According to the results, we observe that whenever LAMBADA predicts PROVED or DISPROVED, the prediction is mostly correct. The accuracy is slightly more on cases where the prediction is PROVED than DISPROVED. We believe this is because DISPROVED cases typically involve negation that makes the reasoning more complex. However, there are several examples for which the label is PROVED or DISPROVED, whereas the model predicts UNKNOWN.

CoT and SI also show similar behaviour as LAMBADA on ProofWriter-PUD but with a larger bias toward prediction UNKNOWN. Moreover, SI shows a large tendency toward predicting DISPROVED for PrOntoQA.

B.5 Lexical Sensitivity Analysis

To analyze the lexical sensitivity of LAMBADA, we created a new test for ProofWriter-PUD which contains tokens that do not appear in demonstra-



(a) (b)

Figure 12: The performance of LAMBADA on ProofWriter-PUD for (a) the original and the novel token test sets, (b) the original and the novel template test sets. The results show that LAMBADA is robust to lexical and template modifications.

tion examples. Specifically, we manually created a pool of entity names, animal names, adjectives, and verbs (all of them previously not appearing in the ProofWriter dataset) and then made the following modifications for each example: 1- identified all entity names and mapped each entity name to a randomly selected name from the pool, 2- identified all animals and mapped each of them to a randomly selected animal from the pool, 3- identified all adjectives and mapped each of them to a randomly selected adjective from the pool, and 4- identified all verbs and mapped each of them (except the *to be* verbs) to a randomly selected verb from the pool. As an example, dog may be mapped to `bison` in one example and to `camel` in another. Then, using the same few-shot examples as before, we tested the performance of LAMBADA on this modified test set and compared the results to the original test set.

We also analyzed the sensitivity to the templates used for the rules. Toward this goal, we identified the templates used for the rules in the ProofWriter dataset and replaced each template with another template (previously not appearing in the ProofWriter dataset). For example, we changed the template "`[X] things are [Y]`" to "`It is a truth that [X] things are always [Y] as well`". Then, using the same few-shot examples as before, we tested the performance of LAMBADA on this modified test set and compared the results to the original test set.

We repeated the aforementioned experiments twice for each analysis each time using a different set of tokens/templates. The results in Figure 8 in the main text demonstrate the average accuracy across two runs. The results for individual runs are presented in Figure 12(a), (b) for the two analyses respectively. According to the results, while we observe some variations in the total accuracy (for some depths the performance goes slightly down and for some depths goes slightly up), the performance stays in the same ballpark, showing the robustness of LAMBADA. Moreover, comparing the results on the modified test set with those of the baselines reported in the main text, we observe that even on this modified test set, LAMBADA performs significantly better than the baselines tested on the original test set.

C Combinatorial Search Issue in Forward Chaining

Consider a simple fictional theory with the following facts:
[`Anne is cold.`, `Anne is nice and pink.`, `Anne is kind.`, `Anne is green.`, `Anne is big and young.`, `Anne is rough.`, `Anne is round.`]
the following rules:
[`Cold, red people are white.`, `Nice, blue people are white.`, `Kind, green people are white.`, `Cold, round people are white.`, `Big, green people are white.`]
and the goal "`Anne is white.`". An approach based on forward chaining requires selecting a subset of the facts and rules from the theory from which this goal can be proved. Specifically, it needs to select "`Anne is cold.`", "`Anne is round.`", and `Cold, round people are white.` from the theory. Such a selection requires a combinatorial search where different combinations of facts and rules should be tested to see which one can lead to proving the goal. An LM may fail to search this space effectively in a single inference call.

SI uses an approximation to reduce the search space: it first makes an inference call to an LM to

select one fact/rule, then it makes another inference call to select the next fact/rule based on the first one, and continues to make inference calls until a halting criterion is met. This approximation reduces the search space from a combinatorial space to a linear space. Since the facts/rules are not selected jointly, however, the chances of selecting the wrong combinations of facts and rules increase because repairing a wrong first choice is not possible, and this leads to low performance as evidenced in our experimental results.

With a backward chaining approach such as LAMBADA, on the other hand, no combinatorial search (or approximations to it) is required: the *Rule Selection* module verifies each rule independently to see which one is applicable (i.e. a linear scan), the *Goal Decomposition* module breaks goals into sub-goals based on each selected rule independently of the other selected rules, and the *Fact Check* module verifies the existence of a fact that entails or contradicts the goal with a linear search over the facts.

D Implementation Details

For our experiments, we used the PaLM 540B model (Chowdhery et al., 2022) for all the models (both LAMBADA and the baselines) served on a 4×4 TPU v4 architecture. The decoding temperature was set to zero. For testing CoT on PrOntoQA, we used the same demonstration examples as the original work but slightly changed the wording by adding conjunctive words such as "`Since`" and "`So`" to make the chains have a better flow. The reason for this modification was that we found when working with PaLM, prompts that have a better flow result in better predictions. This can be viewed from Figure 13 where we compare the performance for the original prompts vs. the prompts with the conjunctive words added. It can be viewed that while the latter slightly underperforms on Depth-1 (where the reasoning flow is not as important), it substantially improves the results for higher depths (especially Depth-5). For ProofWriter, we wrote similar few-shot examples.

For SI, we used the same demonstration examples as in the original work for ProofWriter; for PrOntoQA we wrote few-shot examples following a similar pattern to those for ProofWriter. For each dataset depth we used/wrote specific few-shot examples (e.g., when working with a subset of the data that has examples requiring at most k hops of reasoning, our CoT demonstrations also require only k hops of reasoning), except for ProofWriter Depth-5 where, following the original work, we used it for testing length-generalization and only included examples with chains up to 3 hops. For running CoT on ProofWriter-PUD, we included extra few-shot examples where the label is UNKNOWN; the explanation for these examples is that the goal cannot be proved or disproved with a combination of the facts and the rules. For running SI on ProofWriter-PUD, after obtaining the inferences by running SI, we give the inferences and the goal to our *Fact Check* module which decides if the goal can be proved, disproved, or neither. Since *ProofWriter-PD* and *PrOntoQA* are binary datasets but LAMBADA makes three-way predictions (PROVED, DISPROVED, and UNKNOWN), to test LAMBADA on these datasets, similar to SI we combine the UNKNOWN and DISPROVED predictions into one class.

D.1 Datasets for Individual Module Evaluation

For creating datasets for measuring the performance of individual modules in LAMBADA, we proceeded as follows. For *Fact Check*, we randomly selected 100 examples from the Depth-0 examples. We count a model prediction to be correct if it produces the same label as the one specified in the ProofWriter dataset. For *Rule Selection*, we randomly selected 100 examples and manually enumerated every rule whose consequent unifies with the goal. A model prediction is considered correct if it predicts *all* such rules correctly. For *Goal Decomposition*, we randomly selected 100 rules and goals such that the consequent of the rule unifies with the goal and then manually wrote the subgoals. A model prediction is considered correct if it predicts *all* the sub-goals correctly. For *Sign Agreement*, we re-used the same examples from the *Goal Decomposition* module and manually labeled them with respect to their sign agreement/disagreement.

D.2 Quality Issues in ParaRules

We found the ParaRules dataset to has a high amount of variation in the text, in the facts, and in the rules thus making it a valuable benchmark for evaluating text-based logical reasoning. We also found a few quality issues in the ParaRules dataset that were introduced when annotators converted facts and rules into natural language form. Here, we describe some of the main issues that we

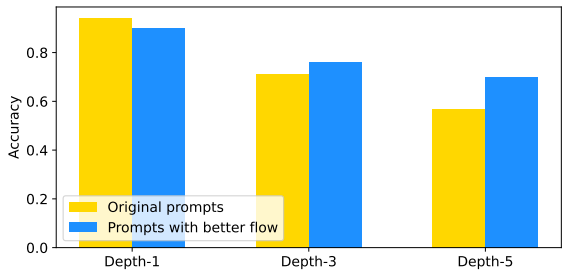


Figure 13: CoT results on PrOntoQA with the original prompts vs. the prompts with conjunctive words added to make the sentences flow better.

found and fixed.

- **Changing antecedents and consequents:** We found that in some cases where the rule was "`X and Y imply Z`", the natural language version of the rule produced by annotators was written as if "`X implies Y and Z`" or "`X implies Y or Z`". As an example, the rule "`Cold, nice people are red.`" was written in natural language form as "`Some cold people can be nice at times,and red at at other times.`". For such cases, we modified the text to make the antecedents and consequent match the original rule.
- **Introducing new antecedents:** In some cases, the annotator introduced new antecedents in the rule. For example, for a rule where the antecedents were "`green`", "`red`" and "`rough`", the annotator added another antecedent "`naive`" ("`If someone is green and naive ...`"). For such cases, we removed the extra antecedents.
- **Turning general rules to specific ones:** In some cases, the natural language version of a general rule was written for only a specific entity. For example the rule "`Rough, young, green people are very round.`" was written as "`Tom is a rough, young person to know ...`". We removed the specific entities and made the rule generally applicable.
- **Introducing pronouns:** For some of the facts, we found that the annotator replaced the name of the entity with a pronoun. As an example, "`Dave is ...`" was annotated as "`He is ...`". We replaced the pronouns with the original entity name in the theory.

D.3 Prompts

We provide an overview of the prompts we used for each of the four components of our model for the ProofWriter dataset.

Algorithm 3 FactCheck

Input: Facts $\mathcal{F}$, Goal $\mathcal{G}$, Number of trials n

for n times do **do**
  f = FactSelection($\mathcal{F}$, $\mathcal{G}$)
  result = FactVerifier(f, $\mathcal{G}$)
  if result $\neq$ UNKNOWN **then**
    return result
  $\mathcal{F} = \mathcal{F} - f$
return UNKNOWN

Algorithm 4 RuleSelection

Input: Rules $\mathcal{R}$, Goal $\mathcal{G}$

I = RuleImplications($\mathcal{R}$)
selected = SelectRules(**I**, $\mathcal{G}$)
return selected

The pseudo-code for the *Fact Check* module is provided in Algorithm 3. For selecting a fact in *Fact Check*, our prompt looks like the following:

```
Example 1
Fact1: <FACT1> Fact2: <FACT2> ... Factn: <FACTn>
Question: <QUESTION>
Inference: For the question <QUESTION> the most relevant fact is Facti (<FACTi>).
...
Example K
Fact1: <FACT> Fact2: <FACT> ... Factm: <FACT>
Question: <QUESTION>
Inference:
```

For verifying if the goal/question can be derived from the selected fact, we use the following prompt:

```
Example 1
Fact: <FACT>
Question: <QUESTION>
Inference: The fact <FACT> [X1] the question <QUESTION> so [X2].
...
Example K
Fact: <FACT>
Question: <QUESTION>
Inference:
```

In the case where the goal can be proved from the fact, we replace `[X1]` with "`is equivalent to`" and `[X2]` with "`so the answer is "yes"`". In the case where the goal can be disproved from the

fact, we replace [X1] with "is the negation of" and [X2] with "so the answer is "no"". And in the case where the goal can neither be proved nor disproved, we replace [X1] with "is neither equivalent nor the negation of" and [X2] with "so the question cannot be inferred from the fact".

The pseudo-code for the *Rule Selection* module is provided in Algorithm 4. For finding the implication/consequent of the rules, we use the following prompt:

```
Example 1
Rule1: <RULE1>, Rule2: <RULE2> ... Rulen: <RULEn>
Inference: Rule1 implies [X1], ..., Rulen implies [Xn].
...
Example K
Rule1: <RULE1>, Rule2: <RULE2> ... Rulem: <RULEm>
Inference:
```

[Xi]s depend on the consequent of each rule. For rules such as "Rough, nice people are red." we write [Xi] as "(is; red)", and for rules such as "If the cat chases the dog then the cat sees the dog." we write [Xi] as "(cat; chase; dog)".

For rule selection based on the implications, we use the following prompt:

```
Example 1
Rule1 implies <IMLP1>, Rule2 implies <IMPL2>, ..., Rulen implies <IMPLn>
Question: <QUESTION>
Inference: The question is about <IMPLq>: Rule1 <IMPL1> [X1] <IMPLq>, ..., <IMPLn> [Xn] <IMPLq>.
...
Example K
Rule1 implies <IMLP1>, Rule2 implies <IMPL2>, ..., Rulem implies <IMPLm>
Question: <QUESTION>
Inference:
```

where each [X1] is either "is applicable to" or "not applicable to" depending on whether the rule can be applied or not.

For goal decomposition, we use the following prompt:

```
Example 1
Rule: <Rule>
Question: <QUESTION>
Inference: The question subject is <SUBJq> and the rule premises are <PRM>*, so the question breaks down to <SUBQ>*.
...
Example K
Rule: <RULE>
Question: <QUESTION>
Inference:
```

where <SUBJq> indicates the subject of the question, <PRM>* indicates the premises/antecedents in the rule (the * indicates that there might be multiple premises), and <SUBQ>* indicates the sub-goals.

Finally, for sign agreement, we use the following prompt:

```
Example 1
Rule: <Rule>
Question: <QUESTION>
Inference: The rule implication <IMLPr> is [Xr], the question <IMPLq> is [Xq], so signs [Xd].
...
Example K
Rule: <RULE>
Question: <QUESTION>
Inference:
```

where <IMLPr> shows the implication of the rule and <IMPLq> indicates the implication of the question. [Xr] and [Xq] are either "positive" or "negated" depending on the sign of the implication. [Xd] is either "agree" or "disagree" depending on whether the signs agree or not.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations section on p9

☑ A2. Did you discuss any potential risks of your work?
Limitations section on p9

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract and Section 1 (Introduction)

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

Section 3 creates a new artifact.

☑ B1. Did you cite the creators of artifacts you used?
We used three datasets referenced in Section 4 (Datasets)

☒ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
The licenses can be found publicly on the corresponding websites: 1- ProofWriter https://allenai.org/data/proofwriter, 2- PrOntoQA: https://github.com/asaparov/prontoqa

☒ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
The datasets were used in the way they were used in the original works.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
Not applicable. Left blank.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Not applicable. Left blank.

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Appendix (implementation details)

C ☑ Did you run computational experiments?

Section 4

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Appendix (Implementation details)

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix (Implementation details)

☒ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
We ran our experiments only once, but there is no randomness in the experiments so running them multiple times gives the same result as running once.

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
We used PaLM (see appendix - implementation details)

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.