EPiC Series in Computing

Volume 100, 2024, Pages 51–69

Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning

Automated Theorem Provers Help Improve Large Language Model Reasoning

Lachlan McGinness[1] and Peter Baumgartner[2]

[1] School of Computer Science, Australian National University and Data61, CSIRO
lachlan.mcginness@anu.edu.au

[2] Data61, CSIRO and School of Computer Science, Australian National University
peter.baumgartner@data61.csiro.au

Abstract

In this paper we demonstrate how logic programming systems and Automated first-order logic Theorem Provers (ATPs) can improve the accuracy of Large Language Models (LLMs) for logical reasoning tasks where the baseline performance is given by direct LLM solutions. We first evaluate LLM reasoning on steamroller problems using the PRONTOQA benchmark. We show how accuracy can be improved with a neuro-symbolic architecture where the LLM acts solely as a front-end for translating a given problem into a formal logic language and an automated reasoning engine is called for solving it. However, this approach critically hinges on the correctness of the LLM translation. To assess this translation correctness, we secondly define a framework of syntactic and semantic error categories. We implemented the framework and used it to identify errors that LLMs make in the benchmark domain. Based on these findings, we thirdly extended our method with capabilities for automatically correcting syntactic and semantic errors. For semantic error correction we integrate first-order logic ATPs, which is our main and novel contribution. We demonstrate that this approach reduces semantic errors significantly and further increases the accurracy of LLM logical reasoning.

1 Introduction, Background and Related Work

The release of models like GPT [3] and Gemini [28] through platforms like ChatGPT and Bard have transformed Large Language Models (LLMs) into general-purpose tools that can be used by everyone. Although designed for next token prediction, LLMs have been shown to have emergent abilities and are able to perform a wide variety of tasks without task-specific training data [3, 20, 25, 30, 31].

Unfortunately, LLMs also frequently return wrong results, such as fictitious claims ("hallucinations") or conclusions that defy common sense or (naive qualitative) physics [13, 16, 27]. Such shortcoming may or may not be obvious but in any case impact trustworthiness. A recent famous example was a lawyer who submitted a legal brief generated by ChatGPT which contained many errors and false references [5, 6]. Asking the LLM for an explanation might help, but the explanation might contain errors again and does not necessarily reflect the process used to

obtain its answer. Equipping and checking LLMs with trustworthy (logical) reasoning remains to be a current major problem [21, 22].

A general approach to address this problem equips LLMs with external functionality [8, 10, 13, 19, 21]. These equipped models are referred to as Augmented Language Models (ALMs). The general problem of combining neural networks with symbolic reasoners has attracted a lot of attention recently (a popular umbrella term is "neuro-symbolic computation"). An impressive example is the work by Trinh et al. [29] which demonstrates that a neuro-symbolic architecture that can solve International Mathematics Olympiad geometry questions at an expert level.

Other proposed combination schemes range from end-to-end differentiable architectures with tightly integrated training regimes [7, 14, 15, 32] to more loosely coupled systems where pre-trained models are linked with a reasoner through a formal language interface [23, 27]. In this paper we consider combinations of the latter kind. Pre-trained LLMs are used as black boxes tasked with translating problems that require logical reasoning into a formal logic, so that an Automated Reasoning (AR) system can be applied. We first show how accuracy can be improved with such a neuro-symbolic architecture.

This approach naturally provides excellent explainability and trustworthiness on the AR side. Therefore the correctness of the overall system critically hinges on the correctness of the LLM translation. However, engineering prompts with high correctness requires many test cases and iterations. As a result, manual inspection of test case results quickly becomes unfeasible. Knowing the types of errors that the LLM makes has the capacity to inform prompt engineers allowing optimal performance to be reached more quickly.

In order to assess the correctness of the LLM translation from natural language into logic programs, we need a reliable ground-truth logic representation for the natural language problem. To make this possible, we follow current approaches and work in a controlled setting. We chose popular "steamroller" problems, which are readily available in useful variants and can be auto-generated in any number [24]. We wrote a standard Definite Clause Grammar (DCG) parser for the required subset of English and that outputs First Order Logic (FOL) formulas, our ground truth formulas. This puts us in a position to compare the two formal logic representations; the first from the LLM and the second from the DCG. We do that in a purely semantic way using SEDAC (*Semantic Error Detection And Correction*); an algorithm that calls an Automated Theorem Prover (ATP) that is capable of deciding entailments in the considered fragment for two given formalizations.

We are interested in analyzing the correctness beyond a binary true/false status. In case of incorrectness, we make certain modifications to the given formulas and check again for entailment. Depending on the result, this allows us to conclude certain error classes and carry out automatic corrections.

We can illustrate our approach with a metaphor from text processing. Virtually every natural language text editor includes a spell-checker for (a) fixing spelling mistakes and (b) grammatical errors. More recently, (c) semantic analysis for, e.g., finding the right words for a given writing style have been added. Roughly speaking, our error categories correspond to these three levels. We have syntax errors (a), shallow semantic errors (b), and deep semantic errors (c). Like in text processing, they come at different levels of automatic detectability, fixability and the need to validate the proposed fix with the user or environment.

As far as we know, ours is the first approach of its kind. We describe it and report on practical experiments. The approach and the result statistics are valuable for at least three reasons:

(1) they provide insights into expected problem areas of ALMs that are generalizable, (2) they can give a human in the loop insights to create targeted improvements to LLM prompts, and (3) they offer the ability to 'auto-correct' some types of semantic errors made by LLMs when calling tools leading to improved performance.

Related work. The reasoning capability of LLMs is an active area of research, we refer to Huang et al. (2023) for a general overview of this field [9]. The majority of this work focuses on enhancing LLM reasoning capabilities with fine-tuning and prompt engineering but without the use of external reasoning tools. Key methods include Chain of Thought (CoT) reasoning [31], zero-shot CoT reasoning [12], Selection-Inference [4] and backward chain reasoning [11].

Although there are many works which measure the performance of LLMs on logical reasoning benchmarks [12, 16, 18], very little work has been done to classify the types of errors they make. Xu et al. [33] focuses on the emergent reasoning capabilities of LLMs (a fully sub-symbolic approach) and proposes two classes; evidence selection errors and reasoning process errors. These categories are not appropriate for neuro-symbolic approaches such as ALMs which allow models to make use of external tools for logical reasoning [16]. In these approaches, reasoning process errors are not relevant, instead the LLM is required to select the correct evidence and successfully translate it into instructions to be parsed by an external tool. Therefore for this domain we propose different error classes: syntactic errors and semantic errors, see Section 2.1.

2 Our Method

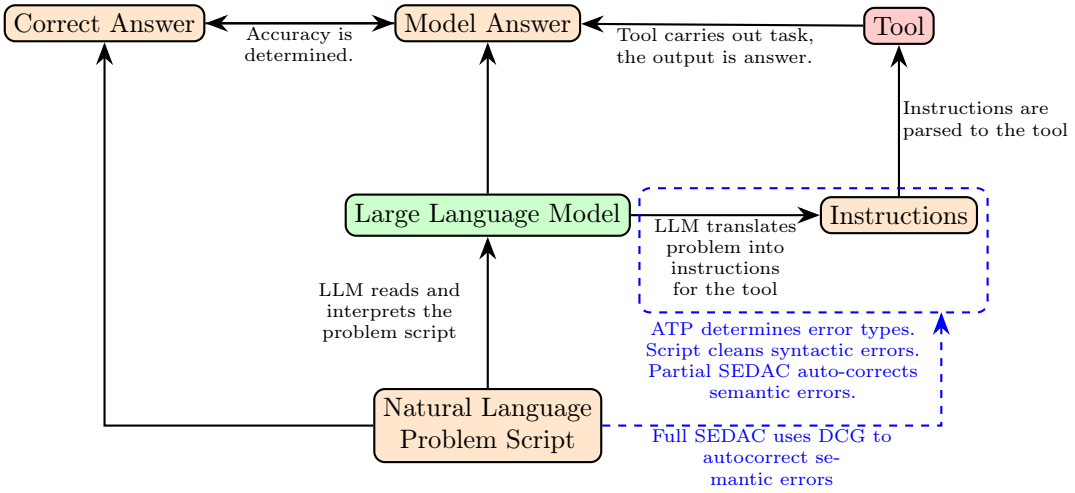
Natural Language Processing is a fast moving area with multiple new LLMs being released each year. This work focuses on only three of the best performing models at the time of the experiment; GPT3.5 [3], GPT4 [17] and Gemini-Pro [28]. This study investigates the logical reasoning skills of these models and how they could be augmented through the use of automated reasoning systems. Figure 1 provides an overview of the general architecture that we explore in our experiments.

To test these models we chose PRONTOQA [24], a logical reasoning dataset, because it has different settings ('ontologies', 'hops' and 'distractors') which can be changed to adjust the difficulty of the problems. PRONTOQA provides the Natural Language Problem Script for our specific experiments. The code for PRONTOQA questions is published but not the questions themselves, which helps prevent contamination of LLMs (reduces the likelihood that they will have seen the exact questions and answers in their training data). We generated one hundred examples of the most difficult problems ('false ontology' with 'relevant distractors') for one hop, two hops and three hops as our evaluation benchmark.

We implemented several experimental conditions for each LLM. In the baseline condition the model was given a question from the benchmark and needed to produce a 'True' or 'False' answer based on the text provided. This corresponds to the arrow pointing from the Large Language Model to the Model Answer in Figure 1. For the zero-shot condition, we provide the LLM with instructions explaining how to write a Logic Program (LP) in Prolog syntax and ask it to convert a natural language problem into such a logic program. The LP *is* the instructions shown in Figure 1.

We chose logic programs as the interchange language because their syntax is already known by the LLMs, they are easy to "teach" to a LLM in a prompt and their simple if-then structure is

Figure 1: A diagram of the general structure of Large Language Model tool use. In order to successfully use a tool an LLM must successfully generate instructions for that tool that are free of both syntax errors and semantic errors. Our contributions to improving this process including auto-correcting and error type classification are shown blue.



sufficient for our purpose. For computing a 'True' or 'False' result we used our Fusemate LP system [1].[1]

For our specific case, Fusemate is the Tool illustrated in Figure 1 that produces the Model Answer. The arrow pointing from Large Language Model to the Instructions is the pipeline that we are evaluating.

For the one-shot condition, we provide the LLM with instructions for how to write a logic program, an example natural language problem, the corresponding logic program and a new natural language problem. Once again the resulting logic program is sent to Fusemate to compute a 'True' or 'False' answer. We repeated this process for each problem in the benchmark and for each of the three LLMs.

We generated error logs from each trial which contain each problem, the corresponding model answer, the correct answer and the logic programs generated by the models for the zero-shot and one-shot conditions. These experiments and their result statistics revealed several weaknesses with these approaches in terms of the error framework introduced above.

2.1 Error Categorisation

Few systems for error categorisation currently exist in the literature [33] and these are not appropriate for categorising errors when Augmented Language Models (ALMs) call upon tools. Therefore we propose a new error categorisation which has two broad classes; syntactic errors and semantic errors.

[1]The PRONTOQA problems are designed in such a way that both an open world or closed world semantics based reasoner can be used with the same result.

A Syntactic Error is defined as an error in the LLM's instructions which prevents the tool from parsing. There are a number of different sub-categories of syntactic error which can contribute to this including:

- **Symbol Errors** - The LLM instructions contain incorrect symbols. As an example consider a logic program which contained "-?" instead of "?-" for a query. This would prevent the script from running and so the instructions can no longer be parsed.
- **Natural Language Errors** - The model includes natural language in addition to or instead of machine interpretable instructions.
- **Communication Errors** - A specific form of Natural Language Error where the model incorporates markers like "‘" or «» to separate natural language from tool instructions. A human can very easily interpret which parts are meant to be included in the instructions. This type of error can be cleaned very easily.
- **Knowledge Errors** - This is a form of evidence selection error [33]. Rather than translating the problem directly, the LLM tries to incorporate some of its own pre-existing knowledge into the instructions. An example is when the model replaces 'even number' with 'integer divisible by 2'.
- **Other Syntax Errors** - Any other syntactic error that prevents the model from parsing which does not fall into the categories above. This is depended on the tool being used.

A Semantic Error is an error in which the instructions are able to be parsed by the tool but give an incorrect output. The exact types of semantic errors are tool dependent. However we recommend breaking these into two sub-categories that will likely be helpful to developers:

- **Shallow Semantic Errors** - Errors where the semantic meaning can confidently be recovered (auto-corrected) without viewing the original natural language script. We suggest that these could be referred to as auto-correctable errors.
- **Deep Semantic Errors** - Errors where the semantic meaning cannot be recovered without viewing the original natural language script. We suggest that these could be referred to as non-auto-correctable errors.

Establishing a system of well defined error categories provides a common language and allows focus on specific common errors for the NLP community to address. This error classification is also important for developers to identify the best technique to improve the performance of LLMs. For example, if a developer discovers a large number of syntactic errors then they know to focus on techniques that can reduce these: one or few-shot prompts, fine-tuning the model with a focus on the tool's grammar or writing a script that will correct syntax on LLM instructions. When there are many semantic errors then the developer may focus on fine-tuning the model with a focus on the meaning of the natural language or choose to flag common semantic errors in the prompt.

2.2 Semantic Error Detection and Correction

In the following we describe our method for analyzing and auto-correcting errors according to our error framework. We start with a brief overview of the main ideas and its core algorithm, SEDAC (*Semantic Error Detection And Correction*) shown by the blue box in Figure 1.

SEDAC takes as input a natural language script *nl* and the string representation of a logic

program *lp*. The *nl* is the original problem statement and, in this sense, holds the "ground truth". The *lp* is meant as a faithful representation of *nl* as obtained by a given LLM. The purpose of SEDAC is to assess the correctness of the *lp* wrt. the *nl* in terms of the error categories defined above. It also carries out fixes for problems spotted along the way.

SEDAC first tries to automatically fix syntactic errors. Correct or fixed statements then proceed to the semantic error detection phase; statements with un-fixable syntax errors are ignored. We distinguish between *partial* and *full* error detection (and correction). These are (potentially incomplete) operational realizations of the shallow and deep error categories introduced above, respectively. Partial error detection is concerned with unsuitable formal representation of adjectives or nouns that can be discovered confidently on linguistic grounds. Sophisticated tools like spaCy[2] can help with this process. Full error detection is concerned with discovering more speculative logical errors such as wrong introduction or removal of negation, and reversed implication. Correspondingly, discovered shallow errors are always corrected without further validation, discovered deep errors require correctness validation wrt. the given *nl* possibly in conjunction with an external trusted source for domain knowledge.

Technically, SEDAC takes the facts and rules p of *lp* and checks them one by one with a logic representation of *nl* and computes a status OK, NonFixableError or FixableError. The status of p is obtained by a soundness check: if *nl* entails p in first-order logic then p's status is OK, otherwise a *propose* function is called that returns candidate fixes for p which are again checked for soundness. Among all proposed sound fixes, if any, some "best" fix is noted with p as a FixableError. A best fix is one that maximizes the number of *nl* statements entailed by a tentatively fixed *lp*.[3] If no sound fix is produced then p's status is NonFixableError. Figure 2 shows the "full" version of SEDAC. There is also a "partial" version described below.

The *nl_to_fof* function. SEDAC calls *nl_to_fof*(s) for translating a natural language sentence s into first-order logic. We implemented *nl_to_fof* as a definite clause grammar (DCG) in (SWI-)Prolog. The grammar was reverse-engineered from PRONTOQA examples[4]; the syntactic elements nouns, verbs and adjectives we retrieved from the PRONTOQA source code. The grammar recognizes quantifiers (determiners) like "each", "any", "every", "a" and tolerates singular/plural formulations. As a side effect, the natural language parser emits first-order logic formulas. The parser adjusts nouns in plural form to singular form, as unary predicates. For example, either sentence "Cats swim." and "Every cat swims" become $\forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swim}(x)$. Adjectives are normalized into nouns, e.g., $\mathsf{even\_number}(x)$ instead of $\mathsf{even}(x)$. This way the grammar defines a canonical logical form for PRONTOQA sentences. This form is taught to and expected from the LLM translation as well.

The *lp_to_fof* function. SEDAC calls *lp_to_fof*(r) for translating a string representation *lp* of a logic program suggested by the LLM into FOL. If the program contains symbol errors, natural language errors or communication errors, an automated fix is attempted by a python script. The resulting statements are parsed and translated into FOL one by one. Parsing may fail as not all syntax errors will always be caught. If it succeeds, translation into FOL is merely syntax rewriting; if it fails then statement is ignored (taken as 'true').

[2] https://spacy.io

[3] It is tempting to instead require "completeness", i.e., the converse of the sondness entailment. This criterion would be too strong in practice in many cases, as the *lp* might lack some formulas but still entail the query.

[4] We found this easier than trying to modify the PRONTOQA code for emitting first-order logic formulas. It also more useful in view of re-usability to domains that are not synthetically made.

Figure 2: The SEDAC algorithm in pseudocode.

Algorithm 1 Semantic Error Detection and Correction, Full-SEDAC(nl, lp)

Input: A PRONTOQA problem nl and its translation into a logic program lp by an LLM.

Output: A status report for every fact and rule of lp.

```
1  nl_ax = {nl_to_fof(s) | s ∈ nl and s is not a query}   Natural language as FOL
2  lp_ax = {lp_to_fof(r) | r ∈ lp and r is not a query}   Logic program as FOL
3  lp_ax_status = {}                                      Result status maps for lp
4  for f ∈ lp_ax                                          Soundness: check if LP entailed by NL
5    if nl_ax ⊨ f                                         Check next fact or rule f
6      lp_ax_status[f] = OK                               Record OK status of f
7    else                                                 Find best modification of f, if any
8      cand_fixes = {f' ∈ propose(f) | nl_ax ⊨ f'}        Get modifications and keep sound ones
9      if cand_fixes == ∅                                 No such modifications exist
10       lp_ax_status[f] = NonFixableSemanticError
11     else
12       f_best = arg max_{f'∈cand_fixes} score(f')      f_best maximizes entailment of nl_ax
13         where score(f') = |{g ∈ nl_ax | (lp_ax \ {f}) ∪ {f'} ⊨ g}|
14       lp_ax_status[f] = FixableSemanticError(f_best)
15 return lp_ax_status
```

The *propose* function. The *propose* function takes a FOL formula f and returns a possibly empty set of *proposal formulas*. The algorithm is presented as a set of rewrite rules "⇔" and derivation rules "⇒" in Figure 3.

Starting from a singleton set comprised of a given formula, the rules are applied exhaustively, in any order, preferring rewrite rules over derivation rules. Rewrite rules replace the premise taken from the current set with its conclusion; derivation rules add to the current set. The result is the saturated set without f. It is not difficult to see that this procedure always terminates.

Rewrite rules are meant for shallow error correction. They revolve around normalization of plural into singular forms, adjectives into nouns, and proper nouns from type positions (predicates) to individuum positions (terms). The derivation rules for deep error correction are more of a speculative kind. We use them for replacing the direction of an implication and complementing literals. These are general rules, not specific to PRONTOQA, but informed by the kinds of errors we observed LLMs make.

Reasoning Complexity and Partial SEDAC. In our highly controlled and closed PRONTOQA environment with its simple formula structure, full error detection poses no problem. The FOL fragment is Bernays-Schönfinkel logic which is decided by our ATP Beagle [2]. Each entailment proof obligation was decided in very short time($<$ 1sec). The sets nl_ax and lp_ax have at most 20 formulas each for a given problem. In the worst case, four candidate fixes are proposed per rule or fact, yielding a maximum of 20 + 4*20 = 100 ATP calls. We investigated 440 problems with Full-SDEDAC which took 12h. This time could be shortened considerably by avoiding file-based ATP interface and with a faster ATP.

Figure 3: The rule system of *propose* for fixing shallow semantic errors (above the double lines) and deep semantic errors (below the double lines).

Premise	Kind	Conclusion	Condition	Example
$\forall x\ p(x) \rightarrow f$	$\Leftrightarrow$	$\forall x\ (x = p) \rightarrow f$	p is proper noun	$\forall x\ \mathsf{tom}(x) \rightarrow \mathsf{swims}(x) \Leftrightarrow \forall x\ (x = \mathsf{tom}) \rightarrow \mathsf{swims}(x)$
$[\neg]p(ns)$	$\Leftrightarrow$	$\forall x\ n(x) \rightarrow [\neg]p(x)$	ns is the plural form of a noun n	$\neg\mathsf{swims}(\mathsf{cats}) \Leftrightarrow \forall x\ \mathsf{cat}(x) \rightarrow \neg\mathsf{swims}(x)$
$[\neg]p(n)$	$\Leftrightarrow$	$\forall x\ n(x) \rightarrow [\neg]p(x)$	n is a singular noun	$\neg\mathsf{swims}(\mathsf{cat}) \Leftrightarrow \forall x\ \mathsf{cat}(x) \rightarrow \neg\mathsf{swims}(x)$
$[\neg]p(a)$	$\Leftrightarrow$	$\forall x\ n(x) \rightarrow [\neg]p(x)$	a is an adjective form of a noun n	$\mathsf{floral}(\mathsf{even}) \Leftrightarrow \forall x\ \mathsf{even\_number}(x) \rightarrow \mathsf{floral}(x)$
$\forall x\ ns(x) \rightarrow f$	$\Leftrightarrow$	$\forall x\ n(x) \rightarrow f$	ns is the plural of noun $n \neq ns$	$\forall x\ \mathsf{cats}(x) \rightarrow \mathsf{swims}(x) \Leftrightarrow \forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swims}(x)$
$\forall x\ f \rightarrow [\neg]ns(x)$	$\Leftrightarrow$	$\forall x\ f \rightarrow [\neg]n(x)$	*same*	$\forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swims}(x) \Leftrightarrow \forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swim}(x)$
$[\neg]a(t)$	$\Leftrightarrow$	$[\neg]n(t)$	a is an adjective form of a noun n	$\mathsf{even}(\mathsf{tom}) \Leftrightarrow \mathsf{even\_number}(\mathsf{tom})$
$\forall x\ a(x) \rightarrow f$	$\Leftrightarrow$	$\forall x\ n(x) \rightarrow f$	*same*	$\forall x\ \mathsf{even}(x) \rightarrow \mathsf{swim}(x) \Leftrightarrow \forall x\ \mathsf{even\_number}(x) \rightarrow \mathsf{swim}(x)$
$\forall x\ f \rightarrow [\neg]a(x)$	$\Leftrightarrow$	$\forall x\ f \rightarrow [\neg]n(x)$	*same*	$\forall x\ \mathsf{floral}(x) \rightarrow \mathsf{even}(x) \Leftrightarrow \forall x\ \mathsf{floral}(x) \rightarrow \mathsf{even\_number}(x)$
$\forall x\ f \rightarrow p(t)$	$\Rightarrow$	$\forall x\ f \rightarrow \neg p(t)$	*none*	$\forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swim}(x) \Rightarrow \forall x\ \mathsf{cat}(x) \rightarrow \neg\mathsf{swim}(x)$
$\forall x\ f \rightarrow \neg p(t)$	$\Rightarrow$	$\forall x\ f \rightarrow p(t)$	*none*	$\forall x\ \mathsf{cat}(x) \rightarrow \neg\mathsf{swim}(x) \Rightarrow \forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swim}(x)$
$\forall x\ f \rightarrow g$	$\Rightarrow$	$\forall x\ g \rightarrow f$	*none*	$\forall x\ \mathsf{cat}(x) \rightarrow \mathsf{swim}(x) \Rightarrow \forall x\ \mathsf{swim}(x) \rightarrow \mathsf{cat}(x)$

More realistic settings have open-world character where the problem statement does not contain full domain information and "ground truth oracles" may not be available. This let us chose first-order logic semantics for the soundness tests; a closed world semantics seems too credulous for entailments (let alone having a highly undecidable entailment problem). As a trivial example, a formula with a syntactic error is always dropped and, this way, could support an unintended entailment with a default negation inference. While the "tool" could, say, employ logic programming for query answering, deep error fixes should be proposed cautiously and only if deductively valid.

These considerations motivated us to evaluate two versions of SEDAC: the full version defined above, and a *partial* version for shallow error correction. More precisely, partial-SEDAC differs from Full-SEDAC in that it receives the *lp* only (no *nl*) and then immediately calls *propose* restricted to rewriting-rule error correction only. The result of the partial-SEDAC call is the result of the *propose* call if not empty (i.e., propose was effective), otherwise it is the given *lp*. (We do not provide pseudo-code here.) These two version allowed us to assess the tradeoffs in effectiveness and expressivity. We report on the results in Section 3 below.

Example. We demonstrate Partial-SEDAC and Full-SEDAC with a small example that we compiled from actual PRONTOQA problems and LLM translations. The example consists of

the sets nl and lp shown on the left of the following table, which are converted to nl_ax and lp_ax shown on the right, respectively, in the first steps of (Full-)SEDAC. Here and below, FOL formulas are written in TPTP FOF syntax [26].

			$\_ax$	
nl	1	`Each integer is not fruity.`	1	`! [A] : (integer(A) => ~ fruity(A))`
	2	`Negative numbers are brown.`	2	`! [A] : (negative_number(A) => brown(A))`
	3	`Wren is an integer.`	3	`integer(wren)`
	4	`True or false: Wren is not fruity.`	4	`% Query ~ fruity(wren) ignored`
lp	1	`even(X) :- integer(X), 0 is X mod 2.`	1	`% Syntax error line ignored`
	2	`integer(X) :- fruity(X).`	2	`! [X] : (fruity(X) => integer(X))`
	3	`integer(wren).`	3	`integer(wren)`
	4	`integer(X).`	4	`! [X] : integer(X)`
	5	`brown(negative).`	5	`brown(negative)`
	6	`?- \+ fruity(wren).`	6	`% Query ~ fruity(wren) ignored`

Our parser for the FOL versions of nl connects adjectives/noun pairs into single-name predicates, e.g., as in `negative_number(X)`. Shallow error correction is designed to align logic programs with this convention. Notice the attempt to bring in "background knowledge" `0 is X mod 2` by the LLM on line 1 of lp_ax without instructing to doing so; we classify this into the sub-category of Knowledge Error.

The FOL resulting from the SEDAC runs are as follows:

	Partial-SEDAC(lp)		Full-SEDAC(nl, lp)
1	`% Syntax error line ignored`	1	`% Syntax error line ignored`
2	`! [X] : (fruity(X) => integer(X))`	2	`! [X] : (fruity(X) => ~ integer(X))`
3	`integer(wren)`	3	`integer(wren)`
4	`! [X] : integer(X))`	4	`% ! [X] : integer(X) is NonFixableError`
5	`! [I] : (negative_number(I) => brown(I))`	5	`! [I] : (negative_number(I) => brown(I))`
6	`% Query ~ fruity(wren) ignored`	6	`% Query ~ fruity(wren) ignored`

It is instructive to compare the results of partial and full SEDAC. Partial-SEDAC(lp) differs from lp_ax only on line 5 by noun and adjective corrections. Full-SEDAC(nl, lp) includes this fix as well. In addition, it fixes the formula $f =$ `! [X] : fruity(X) => integer(X)` on line 2 of pl_ax by negating its conclusion. This happens in three steps. First, the entailment check on line 5 in Full-SEDAC finds $nl\_ax \not\models f$. Then, $propose(f)$ returns four variants of f but only $f' =$ `! [X] : fruity(X) => ~ integer(X)` satisfies $nl\_ax \models f'$. Scoring is irrelevant in this case. The status for f, hence, is FixableError. As a further difference, the formula $f =$ `! [X] : integer(X)` on line 4 of lp_ax has status NonFixableError as $nl\_ax \not\models f$ and no fix is proposed.

Now consider the query `True or false: Wren is not fruity`. The correct answer is `True` as $nl\_ax \models q$ where $q =$ `~ fruity(wren)`. The LLM translation cannot show that ($lp\_ax \not\models q$), neither can the partial fix (Partial-SEDAC(lp) $\not\models q$) but the full fix can (Full-SEDAC(nl, lp) $\models q$).

3 Results

Table 1 shows the overall accuracy of all three models with each experimental condition described in Section 2. The results show that the use of the LP system, Fusemate, increased the accuracy of each LLM by between 10% and 25% of the possible total.

Table 1: Accuracy for each technique for each model type. Random guessing would be expected to achieve an accuracy of 0.5 ± 0.05. The error values are half the range across three trials.

Prompt Strategy	GPT3	GPT4	Gemini-Pro
Normal	0.48 ± 0.06	0.83 ± 0.12	0.47 ± 0.04
Chain of Thought + one-shot	0.65 ± 0.15	0.94 ± 0.04	0.74 ± 0.12
Fusemate	0.66 ± 0.05	0.94 ± 0.015	0.57 ± 0.03
Fusemate + one-Shot	0.76 ± 0.06	0.94 ± 0.015	0.67 ± 0.03
Fusemate + one-shot + syntax fix	0.83 ± 0.06	0.95 ± 0.02	0.74 ± 0.02
Fusemate + one-shot + partial fix	0.87 ± 0.05	0.983 ± 0.005	0.77 ± 0.04
Fusemate + one-shot + full fix	0.98 ± 0.01	0.995 ± 0.005	0.96 ± 0.04

The SEDAC auto-correction successfully reduced errors in all cases. The syntactic fix alone reduced the number of errors of each model by $15 - 30\%$. The partial and full semantic fixes reduced the number of model errors by $45 - 72\%$ and $88 - 92\%$ respectively. In addition to error correction, the SEDAC algorithm also classifies the types of errors.

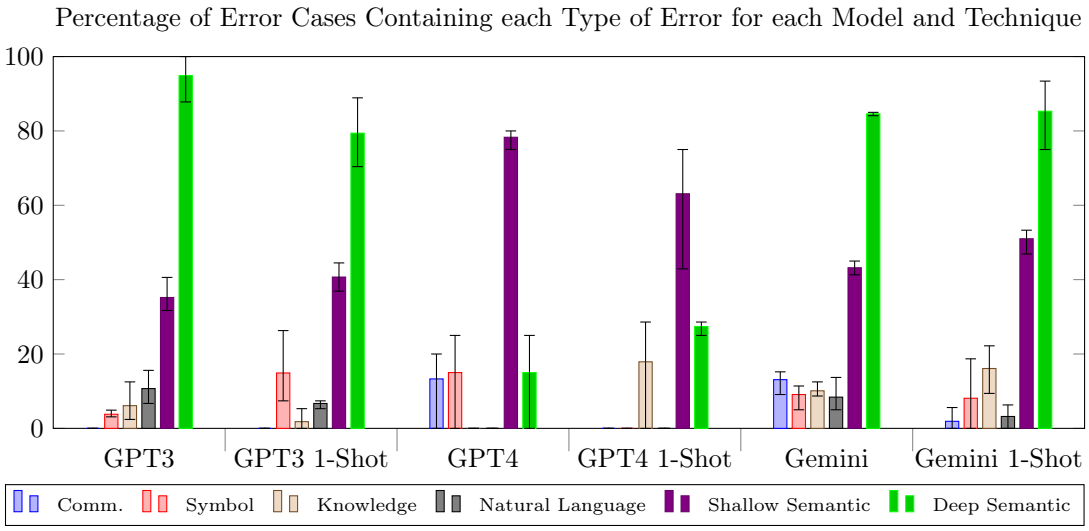
Table 2: Error breakdown for each model type. For each entry, the left value is the average number of each type of error from 100 problems. The values given on the right are half of the range across the three trials for each experimental condition.

Techniques	Communication Errors	Symbol Errors	Knowledge Errors	Natural Language Errors	Other Syntax Errors	Shallow Semantic Errors	Deep Semantic Errors	Total Instances with Errors
GPT3	0.0±0.0	1.3±0.5	2.0±1.5	3.7±1.5	0.3±0.5	12.0±1.5	32.3±3.0	34.3±5.5
1ShotGPT3	0.0±0.0	3.3±1.5	0.3±0.5	1.7±0.5	0.0±0.0	10.0±2.5	19.3±4.5	24.3±4.0
GPT4	0.7±0.5	1.0±1.0	0.0±0.0	0.0±0.0	0.0±0.0	4.7±1	1±1	6.0±1.5
1ShotGPT4	0.0±0.0	0.0±0.0	1.0±1.0	0.0±0.0	0.0±0.0	3.7±1.0	1.67±0.5	6.0±1.5
Gemini	5.7±1.5	4.0±1.5	4.3±0.5	3.7±2.0	1.0±1.0	18.7±0.5	36.7±2.5	43.3±3.0
1ShotGemini	0.7±1.0	2.7±3.0	5.3±2.5	1.0±1.0	0.0±0.0	16.7±2	27.7±0.5	32.7±3.0

For each of the Fusemate methods, the types of errors were determined as described in the Sections 2.1 and 2.2. Table 2 shows the average frequency of each error type across $n = 100$ test examples. For each of the three models the most common error type is the Shallow Semantic Errors. Communication Errors, Symbol Errors, Natural Language Errors and Other Syntax Errors were decreased by introducing the example prompt. The one-shot case did not reduce the number of semantic errors for the GPT4 model, however it did reduce semantic errors by approximately 30% for GPT3 and Gemini.

Figure 4 shows the percentage of error cases which contained each error type for each model and technique. This graph shows that the most common errors for GPT3 and Gemini were Deep Semantic Errors, which occur in 75% to 100% of cases. For GPT4 the most common error was Shallow Semantic errors which occurred in approximately $60\% - 80\%$ of cases. Note that as the graphed results are normalised, they do not allow for direct comparison of the models' ability to translate the semantic meaning from natural language to logic programs.

Figure 4: A graph of the percentage of error cases that contained each error type for each model. Note that this is an indication of the relative frequency of each error type for a given model and experimental condition. Error bars show the minimum and maximum values across the three trials.



Appendix C contains a correlation matrix for each of the different error types. The matrix shows that most correlations are very weak (magnitude < 0.11) with only three exceptions. Knowledge Errors show a correlation of 0.23 with Shallow Semantic Errors, Symbol errors have a 0.31 correlation with Natural Language Errors and Shallow Semantic Errors anti-correlate (-0.37) with Deep Semantic Errors.

Finally, we investigated the effectiveness of our error correction mechanisms. These are 'syntactic fixes only', Partial-SEDAC and Full-SEDAC. As said earlier, the PRONTOQA problems are agnostic of the reasoning type; with an error-free translation the LP (*lp* in Figure 2) is always sufficiently complete in the sense that default reasoning (specifically, default negation) does not enable more conclusions than after reformulation wrt. classical first-order logic. This is no longer true if the transformation is not correct and the error correction is imperfect. In particular, the corrected *lp* may miss relevant rules, which not only removes positive literal conclusions but also adds negative literal conclusions.

For this reason we re-evaluated question answering for different correction scenarios and both open-world and closed-world reasoning. For that, we considered the problems with wrong answers ($n = 440$). The results are summarized in Figure 5 which expands on the summarised results in Table 1. This table shows that the precision values are systematically higher for the open world semantics compared to the closed world semantics.

4 Discussion

The results clearly show that during the time period of the experiments (December 2023), the accuracy of GPT4 on all experimental conditions was significantly higher than GPT3 and Gemini-Pro which were comparable in their performance. Using an AR tool improved the performance comparable with Chain of Thought techniques and our method has the added

Figure 5: Re-running divergent problems after syntax only, Partial-SEDAC and Full-SEDAC corrections wrt. open-world (classical first-order logic) and closed-world (LP) semantics.

	Open-world (FOL)			**Closed-world (LP)**		
	Recall	Precision	Accuracy	Recall	Precision	Accuracy
Syntax errors fixed	0.22	0.57	0.53	0.18	0.16	0.17
Partial-SEDAC	0.38	0.72	0.63	0.35	0.32	0.34
Full-SEDAC	0.80	0.98	0.89	0.85	0.81	0.83

bonus of trustworthy explainability; AR tools can produce a proof for any answer they produce.

For all models, semantic errors were more common than syntax errors. Semantic errors occurred in more than 80% of error cases. Therefore the SEDAC algorithm showed greater error reduction for semantic errors than syntactic errors. Note that although the Full-SEDAC reduced the total number of errors by 90%, most real world scenarios would not have a full semantic fix available. Even in these cases the SEDAC algorithm is useful as it allows for classification of errors to rapidly improve prompting.

As expected, one-shot examples reduced the number of communication errors. Intuitively, providing an example allows the model to better know the required output format. We also expected one-shot to reduce the number of Symbol Errors, this was the case for GPT4, however it made little difference to Gemini and including examples unexpectedly increased the number of Symbol Errors for GPT3.

Syntax errors occur in a relatively small number of cases compared to semantic errors. This indicates that the capability of state of the art LLMs to produce correct syntax exceeds their ability to express the correct semantics to a tool. This demonstrates the importance of AR tools to enhance the models' reasoning capabilities. We speculate that the 'reasoning capacity' of an LLM may be effectively measured by the Chain of Thought accuracy as the corresponding error rate is similar to the total semantic error rate.

There is currently no prevalent system of classifying types errors in LLM use of tools. There is however one more general error structure which exists in the literature which has some relevance [33]. See Appendix A for a comparison between this and our error classification.

The results in Figure 5 confirm our expectations that error correction increases recall consistently for open-world and closed-world semantics. Roughly speaking, recall depends mostly on deductive reasoning, which is not as affected by the change of semantics as precision. A high precision value requires a low false positive rate. In our scenario, false positive are often conclusions in the form of negative literals ("True or false: Tom is a not cat") that become provable by default reasoning when relevant rules are removed by errors. This leads to significantly lower precision than with the open-world semantics. Note that in practice the choice of semantics is mostly likely to be determined by the application domain.

The well defined structure of the natural language in PRONTOQA allows a DCG to achieve 100% performance. However DCGs are not robust even to small deviations from the assumed structure. Testing LLMs on the PRONTOQA dataset allowed for automated measurement of the frequency and type of LLM errors. We hypothesise that LLMs will be significantly more robust to small changes in wording than DCGs and one area for future work is to test LLM reasoning on unstructured natural language.

Local LLMs were not used in this study, instead we utilised APIs for pre-trained remote models. Measuring the computational cost is therefore challenging as the structure and number of parameters in each model is not known. Run-time does not provide a reliable measure of computational cost as data transfer and network latency make a varying and significant contributions. A typical response time was 0.5-5 seconds and most responses contained on the order of 100 tokens. One area for future work is to perform similar experiments using local models to accurately determine the computational cost.

5 Conclusions

In this study we have investigated the intersection of Automated Reasoning and Large Language Models in three different ways. Firstly we have explored the capability of LLMs as stand alone reasoning engines. Secondly we have tried coupling LLMs with external Automated Reasoning systems. Thirdly we have implemented automated reasoning technology to debug LLM reasoning.

We have demonstrated that augmenting an LLM with an AR system improves its reasoning by a similar level to Chain of Thought prompting but with the added bonus of reliable explainability. Furthermore we have introduced the SEDAC algorithm which can act as an auto-correct to reduce LLM errors by at least 15% and up to 90% for problems where a DCG is able to parse the ground truth.

An error classification system was introduced for evaluating interactions between ALMs and their tools. It provides a systematic way to determine the types of errors that LLMs make when interacting with tools. Diagnosing error types provides insight and guidance into which strategies should be implemented to improve model performance. This classification is broad enough that it can be generalised for any external tool while still providing specific information to improve ALM prompts. As the popularity of ALMs rises focus on types of errors gives developers of LLMs a clear direction for improvement.

One key finding from the paper is that semantic errors are far more common than syntactic errors when LLMs call external tools. This is significant for developers who are interested in deploying LLMs for real-world applications. When prompting their models to use external tools, focus should be placed on enhancing model reasoning and semantics not just syntax.

This study considers only a restricted domain of steamroller problems which have highly predictable structures. An area for future research is to apply and evaluate these techniques to a broader class of problems or real-world application and to determine their computational cost.

References

[1] Peter Baumgartner and Elena Tartaglia. Bottom-Up Stratified Probabilistic Logic Programming with Fusemate. *Electronic Proceedings in Theoretical Computer Science*, 385: 87–100, September 2023. ISSN 2075-2180. doi: 10.4204/EPTCS.385.11. URL http://arxiv.org/abs/2308.15862v1.

[2] Peter Baumgartner, Joshua Bax, and Uwe Waldmann. Beagle – A Hierarchic Superposition Theorem Prover. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25*, Lecture Notes in Computer Science, pages 367–377, Cham, 2015. Springer International Publishing. ISBN 978-3-319-21401-6. doi: 10.1007/978-3-319-21401-6_25.

[3] 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. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[4] Antonia Creswell, Murray Shanahan, and Irina Higgins. Selection-Inference: Exploiting Large Language Models for Interpretable Logical Reasoning. In *Conference on Learning Representations*. ICLR2023, September 2022. URL https://openreview.net/forum?id=3Pf3Wg6o-A4.

[5] Matthew Dahl, Varun Magesh, Mirac Suzgun, and Daniel E. Ho. Hallucinating Law: Legal Mistakes with Large Language Models are Pervasive, January 2024. URL https://hai.stanford.edu/news/hallucinating-law-legal-mistakes-large-language-models-are-pervasive.

[6] Matthew Dahl, Varun Magesh, Mirac Suzgun, and Daniel E. Ho. Large Legal Fictions: Profiling Legal Hallucinations in Large Language Models, January 2024. URL http://arxiv.org/abs/2401.01301. arXiv:2401.01301 [cs].

[7] Lennert De Smet, Pedro Zuidberg Dos Martires, Robin Manhaeve, Giuseppe Marra, Angelika Kimmig, and Luc De Raedt. Neural Probabilistic Logic Programming in Discrete-Continuous Domains, March 2023. URL http://arxiv.org/abs/2303.04660.

[8] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided Language Models, 2022.

[9] Jie Huang and Kevin Chen-Chuan Chang. Towards Reasoning in Large Language Models: A Survey. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1049–1065, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.67. URL https://aclanthology.org/2023.findings-acl.67.

[10] Nora Kassner, Oyvind Tafjord, Ashish Sabharwal, Kyle Richardson, Hinrich Schuetze, and Peter Clark. Language Models with Rationality, October 2023. URL http://arxiv.org/abs/2305.14250. arXiv:2305.14250 [cs].

[11] Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. LAMBADA: Backward Chaining for Automated Reasoning in Natural Language. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6547–6568, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.361. URL https://aclanthology.org/2023.acl-long.361.

[12] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large Language Models are Zero-Shot Reasoners. In *36th Conference on Neural Information Processing Systems*. NeurIPS, 2022.

[13] Ruibo Liu, Jason Wei, Shixiang Shane Gu, Te-Yen Wu, Soroush Vosoughi, Claire Cui,

Denny Zhou, and Andrew M Dai. MIND'S EYE: GROUNDED LANGUAGE MODEL REA- SONING THROUGH SIMULATION. In *The Eleventh International Conference on Learning Representations*. ICLR, 2023.

[14] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural Probabilistic Logic Programming, December 2018. URL http://arxiv.org/abs/1805.10872.

[15] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. Neural probabilistic logic programming in DeepProbLog. *Artificial Intelligence*, 298:103504, September 2021. ISSN 00043702. doi: 10.1016/j.artint.2021.103504. URL https://linkinghub.elsevier.com/retrieve/pii/S0004370221000552.

[16] Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. Augmented Language Models: a Survey. *Transactions on Machine Learning Research*, 2023.

[17] OpenAI. GPT-4 Technical Report. Technical report, 2023. URL https://api.semanticscholar.org/CorpusID:257532815.

[18] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language Models as Knowledge Bases? In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2463–2473, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1250. URL https://aclanthology.org/D19-1250.

[19] Gabriel Poesia, Kanishk Gandhi, Eric Zelikman, and Noah D. Goodman. Certified Deductive Reasoning with Language Models, November 2023. URL http://arxiv.org/abs/2306.04031. arXiv:2306.04031 [cs].

[20] Stanislas Polu and Ilya Sutskever. Generative Language Modeling for Automated Theorem Proving, September 2020. URL http://arxiv.org/abs/2009.03393. arXiv:2009.03393 [cs, stat].

[21] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah Smith, and Mike Lewis. Measuring and Narrowing the Compositionality Gap in Language Models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5687–5711, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.378. URL https://aclanthology.org/2023.findings-emnlp.378.

[22] Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. Reasoning with Language Model Prompting: A Survey. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5368–5393, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.294. URL https://aclanthology.org/2023.acl-long.294.

[23] Abhiramon Rajasekharan, Yankai Zeng, Parth Padalkar, and Gopal Gupta. Reliable Natu-

ral Language Understanding with Large Language Models and Answer Set Programming. *Electronic Proceedings in Theoretical Computer Science*, 385:274–287, September 2023. ISSN 2075-2180. doi: 10.4204/EPTCS.385.27. URL http://arxiv.org/abs/2302.03780. arXiv:2302.03780 [cs].

[24] Abulhair Saparov and He He. Language Models Are Greedy Reasoners: A Systematic Formal Analysis of Chain-of-Thought. In *The Eleventh International Conference on Learning Representations*, March 2023. URL https://openreview.net/forum?id=qFVVBzXxR2V.

[25] Aarohi Srivastava and et al. Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=uyTL5Bvosj.

[26] Geoff Sutcliffe. The logic languages of the TPTP world. *Logic Journal of the IGPL*, 31(6):1153–1169, November 2023. ISSN 1367-0751. doi: 10.1093/jigpal/jzac068. URL https://doi.org/10.1093/jigpal/jzac068.

[27] Oyvind Tafjord, Peter Clark, Matt Gardner, Wen-tau Yih, and Ashish Sabharwal. QUAREL: A Dataset and Models for Answering Questions about Qualitative Relationships. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7063–7071, July 2019. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33017063. URL https://ojs.aaai.org/index.php/AAAI/article/view/4687.

[28] Gemini Team. Gemini: A Family of Highly Capable Multimodal Models, 2023. URL http://arxiv.org/abs/2312.11805. arXiv:2312.11805 [cs].

[29] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):476–482, January 2024. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-023-06747-5. URL https://www.nature.com/articles/s41586-023-06747-5.

[30] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL https://openreview.net/forum?id=yzkSU5zdwD.

[31] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *36th Conference on Neural Information Processing Systems*. NeurIPS, 2022.

[32] Thomas Winters, Giuseppe Marra, Robin Manhaeve, and Luc De Raedt. DeepStochLog: Neural Stochastic Logic Programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):10090–10100, June 2022. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v36i9.21248. URL https://ojs.aaai.org/index.php/AAAI/article/view/21248.

[33] Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. Are Large Language Models Really Good Logical Reasoners? A Comprehensive Evaluation and Beyond, August 2023. URL http://arxiv.org/abs/2306.09841. arXiv:2306.09841 [cs].

A Comparison with Existing Error Classification Systems

Xu et al. [33] have two major error categories for determining LLM reasoning capability; evidence selection errors and reasoning process errors. The evidence selection process category is divided into two sub categories which are defined as [33]:

- *Wrong Selection* - 'LLMs select the wrong facts or ignore the necessary facts from the beginning of the reasoning.'
- *Hallucination* - 'LLMs select the evidence which contradicts the given context or cannot be verified from the context.'

Note that these categories combined roughly correspond to Knowledge Errors and Deep Semantic Errors.

Furthermore the reasoning process errors are divided into three sub-categories; no reasoning, perspective mistake and process mistake. In our context the model is not required to reason per se, instead it is required to translate natural language to a logic program. This best approximates the Shallow Semantic Errors as these clearly indicate a failure in logical reasoning. The communication, symbol and natural language errors have no equivalent error in the system proposed by Xu et al. As the two systems of errors only have rough corresponding categories, any comparison of the frequency error categories should only be a rough approximation. This breakdown would give the results displayed in Table 3.

Table 3: This table compares the relative frequency of error categories found by this experiment and those reported by Xu et al. [33]. Note no uncertainty values were reported for the relative frequency of the corresponding error categories. Note that only the GPT3 results were included in this comparison as they most accurate reflection of the models in the review.

Literature Error Categories	Relative Frequency	Corresponding Error Types	Average Relative Frequency for GPT-3
Hallucination and Wong Selection	60.7%	Knowledge Errors and Deep Semantic Errors	$73 \pm 15\%$
Perspective Mistake	44.5%	Shallow Semantic Errors	$52 \pm 6\%$

Note that the results reported by Xu et al. would not consider syntactic errors types (except for knowledge and other syntactic errors) as they do not indicate any error in reasoning, only interfacing with an external tool [33]. Their study found that the total number of types of errors per failure was 1.61; our result for this value is comparable at 1.55 ± 0.06.

B Example LLM Prompt

One of the PRONTOQA steamroller problems reads as follows:[5]

> *Each composite number is not liquid. Every composite number is a fraction. Every composite number is a number. Negative numbers are not large. Every fraction is large. Each fraction is a real number. Fractions are integers. Integers are temperate. Each number is slow. Each even number is loud. Even numbers are natural numbers. Alex is an even number. Alex is a composite number.*

[5] `GPT3Fusemate_hops_2.txt, Iteration: 50`

True or false: Alex is large.

The correct answer is "True". This can be seen from two modus ponens inferences. Several irrelevant utterances can be ignored.

The following instructions are given to the LLMs:

```
Convert the given English statements into a Prolog program.
Use the format and rules below, including an example for guidance.
**Format and Rules:**
- Specific statements become facts: 'Whiskers is a cat' -> 'cat(whiskers).'
- General 'is a' statements become rules: 'All cats are birds' -> 'bird(X) :- cat(X).'
- Use '\+' for negations: 'No bird swims' -> '\+swims(X) :- bird(X).'
- Frame queries with '?-': 'Does Whiskers swim?' -> '?- swims(whiskers).'

**Example:**
English: 'All cats are birds. No bird swims. Whiskers is a cat. Does Whiskers swim?'
Problog: cat(whiskers).
bird(X) :- cat(X).
\+swims(X) :- bird(X).
?- swims(whiskers).
Now, convert the following statements into a Prolog program:
Question: 'Each composite number is not liquid. ...'
Problog Program:
```

(The concrete problem is given in full and only listed in abbreviated form above.)

For example, GPT3 answers with the lines:

```
composite(X) :- number(X), \+prime(X).
liquid(X) :- \+composite(X).
fraction(X) :- composite(X).
number(X).
large(X) :- \+negative(X), fraction(X).
real_number(X) :- fraction(X).
integer(X) :- fraction(X).
temperate(X) :- integer(X).
slow(X) :- number(X).
loud(X) :- even(X).
even(X) :- number(X), X mod 2 =:= 0.
natural_number(X) :- even(X).
alex is large.
```

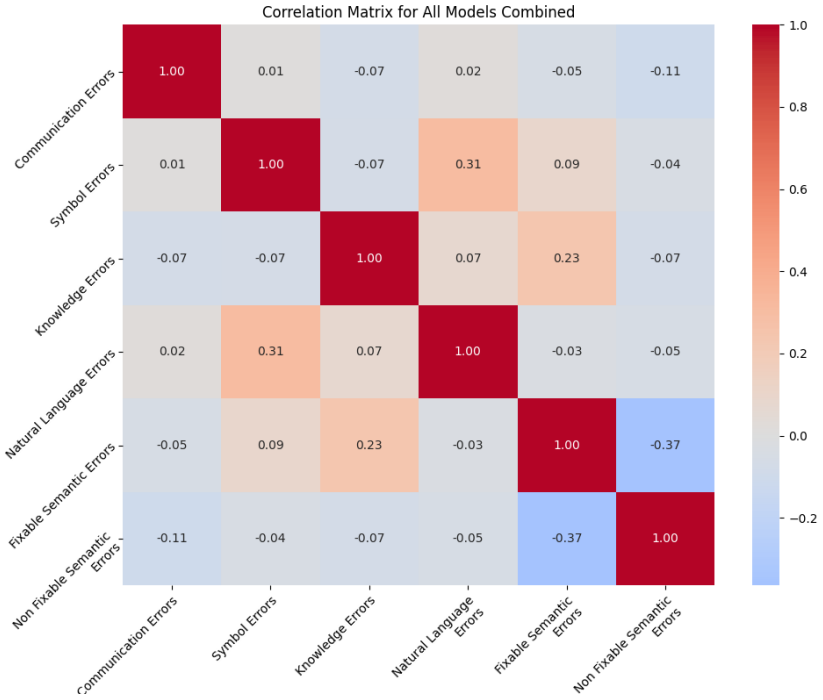
Notice the question is translated incorrectly in the last line, it should be a prolog query `?- large(alex).`. Also the rule for *even* is unexpected and considered a syntax error.

C Correlation Matrix

Figure 6 shows the correlation between error types for our experiments. Note that most of the examples that contained errors came from experiments using GPT3 and Gemini, so GPT4 is underrepresented. The correlation between Natural Language Errors and Symbol Errors can be explained by the experimental conditions. In zero-shot examples the model is more likely to make both natural language errors and symbol errors as shown in Figure 4, while the models

make less of these errors in one-shot exmaples. Therefore we would naturally expect to see a correlation between these error types when considering all examples.

Figure 6: Error Type Correlation Matrix. This shows that there only two significant correlations and one anti-correlation between the types of errors. There is a strong anti-correlation between Shallow Semantic Errors and Deep Semantic Errors, indicating that there are many examples where only one of these two types occurred. There is a correlation between Natural Language Errors and Symbol Errors and also a correlation between Shallow Semantic Errors and Knowledge Errors. All other correlations between errors types are close to 0.



The correlation between Knowledge Errors and Shallow Semantic has an interesting explanation; it is a feature of the dataset, not the error classification system. Knowledge Errors are syntactic errors that cannot be corrected. Therefore when SEDAC investigates semantic errors, these lines will always be disregarded. The results show that for the remaining lines, higher likelihood that there will be Shallow Semantic Errors. This can be explained by looking at the most common cause of knowledge errors: inclusion of mathematical expressions such as `even(X) :- mod2(x)=0`. These problems are also the most likely problems to mistake adjectives as nouns; for example `prime(X)` instead of `prime_number(X)` which can also fixed by partial SEDAC.