Logic-LangChain: Translating Natural Language to First Order Logic for Logical Fallacy Detection

Stanford CS224N Custom Project

Abhinav Lalwani
Department of Computer Science
Stanford University
`lalwani@stanford.edu`

Ishikaa Lunawat
Department of Electrical Engineering
Stanford University
`ishikaa@stanford.edu`

Abstract

Logical fallacies are arguments that use invalid or otherwise faulty reasoning that appear to be well-reasoned until critically examined. Automatically detecting logical fallacies has important applications in tracking misinformation and validating claims. In this paper, we design a process to reliably detect logical fallacies by translating natural language to First-order Logic (FOL) formulas by chaining Large Language Models (LLMs). We then use Satisfiability Modulo Theories (SMT) solvers to reason about the formula's validity. We also develop a novel means of utilizing LLMs to interpret the output of the SMT solver, offering insights into the counter-examples that illustrate why a given sentence is considered a logical fallacy. Our approach is robust, generalizable and does not require training data or fine-tuning. We evaluate our model on a mixed dataset of fallacies and valid sentences. The classifier achieves an F1-score of above 71% on both the base and challenge datasets, demonstrating improved performance and generalization ability compared to current state-of-the-art models.

1 Key Information to include

- Mentor: Tony Wang
- External Collaborators: Lovish Chopra, Christopher Hahn, Caroline Trippel
- Sharing project: Shared with CS 257. Refer to proposal for split details
- Team Contributions: Abhinav - Ideation, Model Design, Experiments, Writing
  Ishikaa: Experiments, Writing, Poster

2 Introduction

A logical fallacy is an argument that may sound convincing, but involves faulty reasoning, leading to an unsupported conclusion. These fallacies can be committed intentionally to manipulate or spread misinformation, and have been used to spread propaganda in news articles (Musi et al., 2022). Consequently, detecting logical fallacies in natural language text holds a very important potential application in tracking misinformation and validating claims. Recognizing fallacious arguments can make discourse more rational and instructive. In general, logical fallacies could be classified into various types (Jin et al., 2022), which are associated with the structure of the sentence. Examples of some of these fallacies are mentioned in Table 1

In the last few decades, formal reasoning tools like SAT and SMT solvers have advanced considerably Consequently, SMT solvers like Z3 (de Moura and Bjørner, 2008), CVC (Barbosa et al., 2022) have become a key tool in different kinds of program analysis and verification, including studying the satisfiability and validity of logical formulae. These formal reasoning tools allow us to precisely represent arguments symbolically and analyze them to detect logical fallacies through systematic

Fallacy Name	Example	Logical Form
Faulty Generalization	Sometimes flu vaccines don't work; therefore vaccines are useless.	$(property1(a) \land a \in b) \Rightarrow (\forall c \in b\ (property1(c)))$
False Causality	Every time I wash my car, it rains. Me washing my car has a definite effect on the weather.	$occuredAfter(a, b) \Rightarrow caused(a, b)$
Ad Populum	Everyone should like coffee: 95% of teachers do!	$manyPeopleBelieve(a) \Rightarrow isTrue(a)$

Table 1: Types of logical fallacies along with examples and their logical forms.

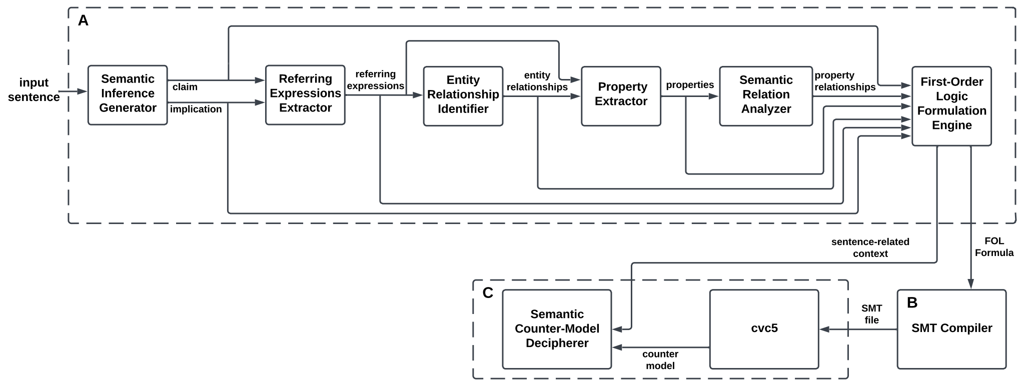


Figure 1: Proposed Logical Fallacy Detection Methodology: *Module A* converts natural language input to a first-order logic formula merged with ground truth, *Module B* compiles the negation of a given logical formula to an SMT file with well-defined sorts for variables and predicates, and *Module C* is used to run CVC on the SMT file and if the negation is satisfiable, interpret the counter-model in natural language.

checking for invalid forms of reasoning. This level of rigorous analysis is difficult for humans, so computational tools are useful supplements to scale analysis across large volumes of arguments through methodical application of the rules of deduction and logical calculus.

In order to utilize theory solvers for detecting logical fallacies, it becomes essential to first convert the given statement to logical form. Most of the existing techniques, as discussed in the next section, do not translate natural language sentences to logical form very well. We have developed an effective technique utilizing chain-of-thought prompting in LLMs and Natural Language Processing to translate a given set of statements to first-order logic. Additionally, theory solvers require context, or ground truth, to accurately distinguish logical fallacies from valid statements. This context provides a semantic interpretation of different variables and predicates, without which they have no meaning. Our methodology introduces an effective way to encode that context in a logical formula and utilize it to enrich the theory solver with the necessary context to aid in decision making.

Theory solvers are a good way to identify the validity of a given logical statement. If a set of logical reasoning arguments are invalid, these solvers can be used to obtain a counter-model to the statements. This counter-model serves the explanation behind the faulty reasoning for the statement by providing an interpretation of different variables and predicates where the claims do not lead to the given inference. Counter-models obtained from theory solvers, however, may be hard to interpret because they are in formal notation, which is incomprehensible to a layperson. We have developed an efficient way to utilize Large Language Models to provide natural language interpretation of the counter-model, which is understandable. This helps in further scaling our approach to track misinformation in the real world and making it more accessible to everyone.

In this paper, we make the following contributions:

1. We develop an explainable and few-shot method for translating Natural Language to First Order Logic by chaining LLMs.

2. We devise a first-order-logic to SMT compiler, which, given any string format first-order logic formula, converts it to an SMT file that can be used by a cvc4 solver (Barbosa et al., 2022).
3. We design an effective technique to interpret the results of cvc4 to explain the faulty reasoning behind the sentence in natural language, making it more interpretable.
4. We evaluate our methodology on numerous datasets and prove that it is highly generalizable by testing its effectiveness over a dataset consisting of real-world fallacies related to climate change: LogicClimate (Jin et al., 2022).

3 Related Work

In this section, we discuss existing research on detection of logical fallacies, converting natural language to first order logic, LLMs and theory solvers.

Logical Fallacy Detection: There have been multiple works on classification of logical fallacies, include classification of argument sufficiency (Stab and Gurevych, 2017), ad hominem fallacies from Reddit posts (Habernal et al., 2018) and dialogues (Sheng et al., 2021), rule parsers (Nakpih and Santini, 2020), structure-aware Transformers (Jin et al., 2022) and instance-based reasoning (Sourati et al., 2023). As per our knowledge, our work is the first that detects all kinds of logical fallacies(not limited to a few specific types), and is also the first to do so in a step-by-step, few-shot and explainable manner.

Natural Language to Formal Logic Conversion: Early works on natural language to formal logic conversion relied heavily on grammar-based approaches that could handle well-structured language, but struggled with more complex linguistic constructions (Purdy, 1991; Angeli and Manning, 2014; MacCartney and Manning, 2014). These works are hard to generalize because of their inability to work with random sentences without a fixed structure. More recently, deep learning has been used to translate natural language to linear temporal logic (Cosler et al., 2023; Fuggitti and Chakraborti, 2023; Liu et al., 2022) and first order logic (Singh et al., 2020; Yang et al., 2023).However, these methods do not provide a way to incorporate ground truth claims, which are necessary for distinguishing logical fallacies from valid sentences. Additionally, most of the approaches have not reached to a level where complex sentences could be accurately transformed to logical form as well as it can be done manually.

Theory Solvers: SMT solvers like Z3 (de Moura and Bjørner, 2008) and CVC (Barbosa et al., 2022) are commonly used to check the satisfiability and validity of logical formulas. They have enabled applications like system verification, program analysis, and model checking. Given a set of logical formulas, an SMT solver determines their satisfiability by applying theories and inference rules. Validity can be checked by taking the negation of the formula and testing if the negation is unsatisfiable.

4 Approach

4.1 Natural Language to First Order Logic

We translate natural language to first order logic formulas by chaining Language Models. For each step, we use an LLM with few shot prompts, unless specified otherwise. The outputs of each step are processed and then used as inputs for the next steps. We would use two simple examples to explain the algorithm: Example 1 below is a logical fallacy and Example 2 is a valid statement.

Example 1: I met a tall man who loved to eat cheese, now I believe all tall people like cheese.
Example 2: A boy is jumping on skateboard in the middle of a red bridge. Thus, the boy does a skateboarding trick.

The first step is to transform a natural language sentence to claim and implication form. Generally, a sentence can be split into some claims and some implication based upon those claims. It is also possible for a sentence to have no claim, which means that the entire sentence is being asserted with respect to the ground truth.

Example 1: *Claim:* A tall man loved to eat cheese. *Implication*: All tall people like cheese.
Example 2: *Claim:* A boy is jumping on skateboard in the middle of a red bridge. *Implication:* the boy does a skateboarding trick.

Next, we split the claim and implication into various sub-components . The first set of sub-components are referring expressions. Referring expressions are used to identify specific entities and could be any noun phrase, or surrogate for a noun phrase, whose function in discourse is to identify some objects. Additionally, we find the relationship between different entities using Zero-Shot classification via Natural Language Inference (NLI) (Yin et al., 2019). These relationships (subset / equality / not related) are generally helpful in adding appropriate quantifiers in the logical form of the sentence. For example, if the entities are 'man' and 'people', then it can be inferred that 'man' is a subset of 'people', and that the man would be bound by an existential quantifier in the generated logical form.

Example 1: *Referring Expressions:* man: x, cheese: c, people: y, $x \in y$
Example 2: *Referring Expressions:* boy: x, skateboard: s, bridge, skateboardingTrick: y

The other set of sub-components are properties, which are used describe a trait of a referring expression or a relationship between multiple referring expressions. These properties are essentially predicates in first-order logic. We also find the relationships between numerous properties. For example, in Example 1, Like and Love are equivalent. Similarly, in the valid example, 'jumping over skateboard' implies 'doing a skateboard trick'. These relationships represent a form of ground truth / context that is not directly present in the statement, but is necessary to prove the statement's validity. To identify these ground truth relationships, we run NLI between each pair of properties, i.e, by setting one property as the hypothesis and the other as the premise as the input to the NLI model. If we find that any one property entails the other, we add the relationship $property1 \Rightarrow property2$ to our ground-truth. Before running the NLI model between a pair of properties, we replace the variables in each property with the referring expressions that they represent. This adds additional context that helps the NLI model identify relations. For example, in Example 2, the NLI model is unable to find the relation between *JumpsOn(x,s)* and *Does(x,y)*, but is able to identify the relationship between *JumpsOn(boy,skateboard)* and *Does(boy,skateboardingtrick)*.

Example 1: *Properties:* Tall, Love, Like
Relationships: Tall(x), Love(x, c)
Ground truth:

- $\forall x(Like(x, c) \Rightarrow Love(x, c))$
- $\forall x(Love(x, c) \Rightarrow Like(x, c))$
- $x \in y$

Example 2: *Properties:* JumpsOn, inMiddleOf, Red, Does
Relationships: JumpsOn(b, s), inMiddleOf(b, bridge), Red(bridge), Does(b, y)
Ground truth:

- $\forall x(JumpsOn(x, s) \Rightarrow Does(x, y))$

Finally, we combine all of the sub-components and their relationship between numerous properties and entities to obtain the first-order logical form of the sentence. For a logical fallacy, the negation of the formula is expected to be satisfiable. For a valid statement, the negation of the formula should be unsatisfiable. This leads us to the next step, which is to feed the formula to an SMT solver. [1]

[1]The prompts we used for each step are available alongside the code submission

Example 1: *First-Order Logic:*
$((\forall x(Like(x,c) \Rightarrow Love(x,c))) \land (\forall x(Love(x,c) \Rightarrow Like(x,c))) \land (\exists x(Tall(x) \land Love(x,c)))) \Rightarrow (\forall y(Tall(y) \Rightarrow Like(y,c)))$
Example 2: *First-Order Logic:*
$((\forall x JumpsOn(x,s) \Rightarrow Does(x,y)) \land Red(bridge) \land inMiddleOf(b, bridge) \land JumpsOn(b,s)) \Rightarrow Does(b,y)$

4.2 First Order Logic to SMT Solving

Our next step involves automatically creating an SMT file for the negation of the first-order logical formula. Given a logical formula, while one can easily write an SMT file for the same manually, generating one automatically for an arbitrary formula is something that has not been done before, and is one of our major contributions.

We have developed an efficient compiler for parsing a given logical formula and converting it into a SMT file that can be given as input to CVC, as described in Algorithm 1. Some of the major challenges involved in designing the compiler were designing a recursive infix-to-prefix algorithm and devising a novel algorithm (Algorithm 2) to identify and unify sorts (domains of variables/properties).

Algorithm 1 Logical Formula to SMT Compilation

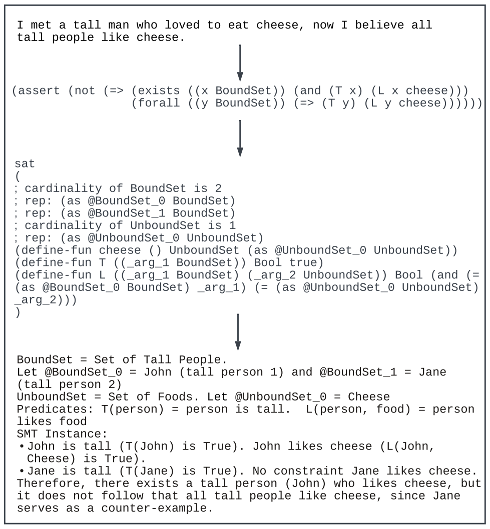
```
1. Split the formula across any operator, parentheses, or commas into tokens.
2. Process tokens to instances of operators, variables and predicates. For predicates, identify all arguments
   and recursively process tokens for the arguments separately.
3. Convert the main logical formula from infix to prefix form. For predicates, recursively convert the
   arguments to prefix form.
4. Identify sorts of all variables and predicates using $unify\_sort$ described in Algorithm 2.
5. Parenthesize the prefix form formula to bring it into SMT format appropriately.
6. Create the SMT file by declaring appropriate sorts, variables and predicates using $(declare - sort)$
   and $(declare - fun)$. Assert negation of the logical formula. Add $(check - sat)$ and $(get - model)$
   to the SMT file.
```

4.3 Interpretation of SMT Solver Results

We send the SMT file that we generate to the cvc4 solver (Barrett et al., 2011) to get the result (sat / unsat), and if it is satisfiable, then it returns a model, i.e, a concrete assignment of values to the variables in the formulas that makes the formulas true. Since we assert the negation of the actual logical formula, this model acts as a counter-example to the original formula, proving that the given formula is a logical fallacy

Generally, it is difficult to understand the model generated by the SMT solver, especially for a layperson. In order to explain the counter-example better to prove that the reasoning is faulty, it is essential to explain the counter-example in natural language. To do so, we prompt an LLM with the claim, implication, referring expressions, properties, first-order logical formula and the counter-model generated by cvc4. The transformer model is then utilized to interpret the counter-model using natural language as depicted in the figure.

A simplified example for the same is given in Figure 2. As evident, the SMT result is hard to understand because it uses technical terminology that can generally be only understood by those who understand cvc4 and SMT well. Therefore, we developed a pipeline to convert the cvc4 results back to natural language to explain why the reasoning is faulty.



```
I met a tall man who loved to eat cheese, now I believe all
tall people like cheese.
```

```
(assert (not (=> (exists ((x BoundSet)) (and (T x) (L x cheese)))
                 (forall ((y BoundSet)) (=> (T y) (L y cheese))))))
```

```
sat
(
; cardinality of BoundSet is 2
; rep: (as @BoundSet_0 BoundSet)
; rep: (as @BoundSet_1 BoundSet)
; cardinality of UnboundSet is 1
; rep: (as @UnboundSet_0 UnboundSet)
(define-fun cheese () UnboundSet (as @UnboundSet_0 UnboundSet))
(define-fun T ((_arg_1 BoundSet)) Bool true)
(define-fun L ((_arg_1 BoundSet) (_arg_2 UnboundSet)) Bool (and (=
(as @BoundSet_0 BoundSet) _arg_1) (= (as @UnboundSet_0 UnboundSet)
_arg_2)))
)
```

```
BoundSet = Set of Tall People.
Let @BoundSet_0 = John (tall person 1) and @BoundSet_1 = Jane
(tall person 2)
UnboundSet = Set of Foods. Let @UnboundSet_0 = Cheese
Predicates: T(person) = person is tall.  L(person, food) = person
likes food
SMT Instance:
 •John is tall (T(John) is True). John likes cheese (L(John,
  Cheese) is True).
 •Jane is tall (T(Jane) is True). No constraint Jane likes cheese.
Therefore, there exists a tall person (John) who likes cheese, but
it does not follow that all tall people like cheese, since Jane
serves as a counter-example.
```

Figure 2: Interpretation of results from a counter-example

5 Experiments

5.1 Data

We use the following three datasets to evaluate the effectiveness of our approach:

1. LOGIC (Jin et al., 2022): consists of 2,449 common logical fallacies.
2. LOGICClimate (Jin et al., 2022): consists of 1,079 logical fallacies from climate change news from the Climate Feedback website.
3. Stanford Natural Language Inference (SNLI) Corpus (Bowman et al., 2015): contains over 170,000 valid sentences generated by combining 'sentence 1' and 'sentence 2' from the entailment data points to form a sentence where claim entails the implication.

Since the LOGIC and LOGICClimate datasets contain only logical fallacies, we randomly sample equal number of valid statements from the SNLI corpus to balance the datasets.

5.2 Evaluation method

We consider the F1, precision, recall, and accuracy metrics. We compare our method with NLI Zero Shot Classifiers (BART MNLI) and the Llama 7b LLM with an end-to-end few-shot prompt. We also perform a qualitative analysis as shown in the Analysis section.

5.3 Experimental details

We use Meta's open-source Llama 2 (7B-parameters) Touvron et al. (2023) for LLM prompting and BART (140M parameters) Lewis et al. (2020) finetuned on MNLI Williams et al. (2018) for identifying the relationships between properties and referring expressions. We run the experiments on a V100 GPU, and one run costs around 2 GPU hours.

5.4 Results

The experiment results are highlighted in Table 2.

Based upon the methodology, we found that LLMs are a great way to identify properties and referring expressions in the sentence, and natural language inference can be used to identify the relationships between properties and entities quite well.

Metric	ExplainNLI (Few Shot)	BART-MNLI (Zero Shot)	Llama-7b (Few Shot)
Accuracy	0.63	0.58	0.41
Precision	0.58	1	0.45
Recall	0.92	0.15	0.82
F1-Score	0.71	0.26	0.58

Table 2: Comparison of Metrics for ExplainNLI with end-to-end approaches for the LOGIC+SNLI dataset

We found the accuracy of the method to be 63%, which outperforms end-to-end few-shot and zero-shot classification techniques using the same models. We also observed a high recall, which means that the model can flag fallacies effectively.

Our challenge set, LOGICClimate+SNLI, is a set of real-world logical fallacies from climate change news. The results obtained are shown in Table 3. The results are highly similar to the results of the LOGIC dataset. This indicates that our system is highly robust and effectively generalizes to real-world text, even when it contains large amounts of domain-specific context, making it effective in identifying and preventing misinformation.

Metric	ExplainNLI (Few Shot)	BART-MNLI (Zero Shot)	Llama-7b (Few Shot)
Accuracy	0.66	0.57	0.31
Precision	0.6	1	0.38
Recall	0.94	0.14	0.62
F1-Score	0.73	0.25	0.47

Table 3: Comparison of Metrics for ExplainNLI with end-to-end approaches for the LOGICClimate+SNLI dataset

6 Analysis

As evident from the results, proving a statement to be valid is harder than identifying it as a logical fallacy, contributing to the lower precision of the model. This is because it is inherently difficult to prove the negation of a statement as unsatisfiable compared to satisfiable. This challenge arises because the model may not have articulated some semantics or ground truth in the first-order logical formula that may be necessary to prove validity. If this context is not well established in the SMT code explicitly, we cannot prove validity, because it would be easy to build a counter-example. The SMT needs full context, and any gaps in contextual information can cause a valid statement to be mistakenly identified as a logical fallacy.

One such case is present in example 4 of the Table 4. In this case, the NLI model does not identify a required ground-truth relation. If this context were identified and added to the claim of the logical formula, then it would have predicted the statement to be valid. Furthermore, our current approach is limited to discerning the NLI relationship between two properties at a time, rather than handling multiple relationships concurrently. For example, consider example 6 in Table 4. In the given example, the semantic claim involves the conjunction of two properties entailing the third, while the NLI model only checks if one property entails the other. Finding such complex extra context requires more advanced NLP techniques or human intervention, and including this can further improve the precision of the model.

Nonetheless, it is important to clarify that these examples do not imply a general inefficiency of NLI in identifying property relations. An interesting illustration of where they work well can be found in Example 5 from Table 4. In this instance, our model identifies additional context by establishing relationships such as IsBaseballPlayer implying IsPlayingBaseball, and IsNearOutfieldFence implying IsOutdoors. These contextual connections help in effectively proving the validity of the statement.

The examples in Table 4 prove that few shot prompting with LLMs is a great way to convert a sentence to first-order logic in a step-by-step manner, improving the efficiency of conversion. It

S.No.	Type	Sentence	Logical Form	Prediction
1	LF	X has been around for years now. Y is new. Therefore, Y is better than X.	(IsNew(a) ∧ ∼ IsNew(b)) ⇒ (IsBetterThan(a,b))	LF: Correct prediction
2	LF	Jimmy isn't at school today. He must be on a family trip.	(∼IsAtSchool(a)) ⇒(IsOnFamilyTrip(a))	LF: Correct prediction
3	LF	Everyone is doing the Low-Carb Diet.	(∃ b (∃ a (IsDoing(b,a)))) ⇒ (∃ c (∃ a (IsDoing(c,a))))	Valid: Incorrect prediction: Wrong translation when there was no claim given
4	V	Two dogs are fighting in a field. Consequently, the two dogs are outside.	(∃ b (∃ a (IsFighting(a, b) ∧ IsInField(b) ∧ IsInField(b)))) ⇒ (∃ a (IsOutside(a)))	LF: Incorrect prediction: Missing semantic ground truth claim: ∀ a (IsInField(a) ⇒ IsOutside(a))
5	V	A baseball player gets ready to catch a fly ball near the outfield fence. Therefore, a person is playing baseball outdoors.	(∃ a (IsGettingReady(a) ∧ (IsABaseballPlayer(a) ∧ IsCatchingFlyBall(a) ∧ IsNearOutfieldFence(a))) ∧ (∀ e (IsABaseballPlayer(e) ⇒ IsPlayingBaseball(e))) ∧ (∀ f (IsPlayingBaseball(f) ⇒ IsABaseballPlayer(f))) ∧ (∀ g (IsNearOutfieldFence(g) ⇒ IsOutdoors(g)))) ⇒ (∃ c (∃ a (IsPlayingBaseball(a) ∧ IsOutdoors(c))))	Valid: Correct Prediction
6	V	A woman sits alone on a park bench in the sun. Hence, a women is in a park.	(IsSittingOn(a, b) ∧ isParkBench(b) ∧ IsInSun(a)) ⇒ (IsInPark(a))	LF: Incorrect prediction: Missing semantic ground truth claim: $\forall a \forall b$ (IsSittingOn(a, b) ∧ isParkBench(b) ⇒ IsInPark(a))

Table 4: Some sample results from our model run: Type *LF* indicates *Logical Fallacy* and *V* indicates *Valid statement*

can be seen that most of these examples identify referring expressions and properties quite well, and are able to produce syntactically correct expressions. Various examples like example 1 and 2 correctly derect logical fallacies. Consequently, we also obtained a very high recall, beating the baseline techniques by a significant amount.

Among the few logical fallacies where our model incorrectly predicted a logical fallacy to be a valid statement, most of these predictions failed due to the imprecision of the LLM, leading to false translations and incorrect results. Example 3 is a prominent case where the input does not have any claim, rather just jumps to an implication. However, the model is not able to identify that the example has no claim. As a result, we get an incorrect translation from our technique. We believe that utilizing more advanced LLMs in future experiments will help prevent these issues and improve our statistics further.

7 Conclusion

In conclusion, we presented an automatic and effective solution for detecting fallacies and tackling misinformation. We developed a strategy to distinguish logical fallacies from valid statements, which involves a chaining approach to convert a sentence to first-order logic using Large Language Models, followed by using SMT solvers to identify whether the first-order logical statement is valid or not, and if not, interpret the counter-model generated by the SMT solver in natural language. Our proposed technique showed promising results in identifying logical fallacies and valid statements. The primary bottleneck is the natural language to first-order logic conversion, which is ongoing research. One potential improvement is the incorporation of more advanced LLMs. Additionally, the methodology can be improved in future work to classify logical fallacies into different categories alongside detection. The step-by-step nature of our approach also enables the incorporation of human feedback into the pipeline.

References

Gabor Angeli and Christopher D Manning. 2014. Naturalli: Natural logic inference for common sense reasoning. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 534–545.

H. Barbosa et al. 2022. cvc5: A versatile and industrial-strength smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2022. Lecture Notes in Computer Science*, volume 13243, Cham. Springer.

Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. 2011. Cvc4. In *Computer Aided Verification*, pages 171–177, Berlin, Heidelberg. Springer Berlin Heidelberg.

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.

M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel. 2023. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In *Computer Aided Verification. CAV 2023. Lecture Notes in Computer Science*, volume 13965, Cham. Springer.

Francesco Fuggitti and Tathagata Chakraborti. 2023. NL2LTL – a python package for converting natural language (NL) instructions to linear temporal logic (LTL) formulas. In *AAAI*. System Demonstration.

Ivan Habernal et al. 2018. Before name-calling: Dynamics and triggers of ad hominem fallacies in web argumentation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 386–396, New Orleans, Louisiana. Association for Computational Linguistics.

Zhijing Jin, Abhinav Lalwani, Tejas Vaidhya, Xiaoyu Shen, Yiwen Ding, Zhiheng Lyu, Mrinmaya Sachan, Rada Mihalcea, and Bernhard Schoelkopf. 2022. Logical fallacy detection. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 7180–7198, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

M. Lewis et al. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 2020 Association for Computational Linguistics (ACL)*.

J.X. Liu, Z. Yang, B. Schornstein, S. Liang, I. Idrees, S. Tellex, and A. Shah. 2022. Lang2ltl: Translating natural language commands to temporal specification with large language models. In *Workshop on Language and Robotics at CoRL 2022*.

Bill MacCartney and Christopher D Manning. 2014. Natural logic and natural language inference. In *Computing Meaning: Volume 4*, pages 129–147. Springer.

Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg. Springer Berlin Heidelberg.

E. Musi et al. 2022. Developing fake news immunity: fallacies as misinformation triggers during the pandemic. *Online Journal of Communication and Media Technologies*, 12(3):e202217.

Callistus Ireneous Nakpih and Simone Santini. 2020. Automated discovery of logical fallacies in legal argumentation. *International Journal of Artificial Intelligence and Applications (IJAIA)*, 11.

William C Purdy. 1991. A logic for natural language. *Notre Dame Journal of Formal Logic*, 32(3):409–425.

Emily Sheng et al. 2021. “nice try, kiddo”: Investigating ad hominems in dialogue responses. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 750–767, Online. Association for Computational Linguistics.

Hrituraj Singh, Milan Aggrawal, and Balaji Krishnamurthy. 2020. Exploring neural models for parsing natural language into first-order logic.

Zhivar Sourati, Vishnu Priya Prasanna Venkatesh, Darshan Deshpande, Himanshu Rawlani, Filip Ilievski, Hông Ân Sandlin, and Alain Mermoud. 2023. Robust and explainable identification of logical fallacies in natural language arguments.

Christian Stab and Iryna Gurevych. 2017. Recognizing insufficiently supported arguments in argumentative essays. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 980–990, Valencia, Spain. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

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

Yuan Yang, Siheng Xiong, Ali Payani, Ehsan Shareghi, and Faramarz Fekri. 2023. Harnessing the Power of Large Language Models for Natural Language to First-Order Logic Translation. *arXiv e-prints*, page arXiv:2305.15541.

Wenpeng Yin, Jamaal Hay, and Dan Roth. 2019. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *CoRR*, abs/1909.00161.

A Appendix

A.1 Unify Sort Algorithm

Algorithm 2 unify_sort for one predicate, say A(x, y)

```
1. Declare current sort of A: (NULL, NULL, Bool)
2. For each instance of predicate A:
    (a) Find the sort of arguments based upon the instance (instance sort):
        i. If argument is a formula, then sort(arg) = Bool.
        ii. If argument is a variable, then sort(arg) = sort(variable) [may be null]
    (b) Unify current sort with instance sort:
        i. If sorts of an argument in the current sort and instance sort are not NULL and different, then
           raise Error (incompatible sorts).
        ii. If current argument sort is NULL and corresponding instance sort is not NULL, then update
            current argument sort = instance sort.
        iii. If instance argument sort is NULL and corresponding current sort is not NULL, then update
             the sort of the corresponding variable to current sort.
```