NALOMA 2026

The 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA)

Proceedings of the Workshop

August 3-7, 2026

Order copies of this and other ACL proceedings from:

Association for Computational Linguistics (ACL)
317 Sidney Baker St. S
Suite 400 - 134
Kerrville, TX 78028
USA
Tel: +1-855-225-1962
acl@aclweb.org

ISBN 979-8-89176-389-0

Introduction

Welcome to the 6th edition of the Natural Language Meets Logic and Machine Learning workshop (NALOMA).

NALOMA continues to serve as a venue dedicated to bridging the gap between machine/deep learning approaches on the one hand, and symbolic/logic-based approaches to natural language understanding and reasoning on the other. A central focus of the workshop remains the development of hybrid approaches and the exploration of theoretical insights that shape and guide computational models of reasoning.

NALOMA took place in August 3-7 during ESSLLI 2026, hosted at Czech Technical University Prague, Czechia. We are deeply grateful to the ESSLLI organizers for their support. The workshop was held over a period of five days, with time slots of about one and a half hours. This year's program featured three inspiring keynotes, seven regular talks with accompanying archival papers included in this proceedings, and two contributed talks based on non-archival submissions.

We would like to thank all authors of archival or non-archival submissions, as well as the dedicated members of the program committee whose careful reviews ensured the quality of the workshop. Our thanks also go to our keynote speakers for sharing their expertise and vision.

As in prior years, NALOMA serves as a platform connecting the symbolic AI and logic communities with the machine learning community, with the dual purpose of promoting discussion and fostering joint research initiatives. We look forward to the collaborations and insights that will arise from this year's event.

NALOMA is endorsed by the Special Interest Group on Computational Semantics (SIGSEM), for which we are grateful.

Hitomi Yanaka and Lasha Abzianidze, Program Co-Chairs

Organizing Committee

Program Chairs

Hitomi Yanaka, the University of Tokyo, Japan
Lasha Abzianidze, Utrecht University, the Netherlands

Program Committee

Program Chairs

Hitomi Yanaka, the University of Tokyo
Lasha Abzianidze, Utrecht University

Program Committee

Lasha Abzianidze, Utrecht University
Stergios Chatzikyriakidis, University of Crete
Katrin Erk, University of Texas, Austin
Hai Hu, Shanghai Jiaotong University
Thomas Icard, Stanford University
Aikaterini-Lida Kalouli, Bundesdruckerei GmbH
Lawrence S. Moss, Indiana University
Valeria de Paiva, Topos Institute
Hitomi Yanaka, the University of Tokyo
Robin Cooper, University of Gothenburg
Daisuke Bekki, Ochanomizu University
Staffan Larsson, University of Gothenburg
Bill Noble, Massachusetts Institute of Technology
Masaya Taniguchi, Riken
Gijs Wijnholds, Leiden University

Keynote Talk

Reasoning in Language Models: Insights from Cognitive Science

Raffaella Bernard
Free University of Bozen-Bolzano

Abstract: Coming from the ESSLLI Logic & Language tradition, I view reasoning as a core component of language: language is the vehicle through which we express our otherwise hidden reasoning processes. Large Language Models have reached remarkable levels of linguistic proficiency. The question now is not only what they say, but what lies behind their linguistic expressions.
Reasoning is multifaceted. We deduce conclusions from premises, induce general rules from examples, ask questions to acquire information that is essential for solving a problem, and revise our beliefs when new evidence becomes available. Consequently, the question we face is equally multifaceted: which reasoning abilities have Large Language Models actually acquired, if any, and how can we determine whether they genuinely reason?
In this talk, I will present a series of recent studies from my research group investigating deductive, inductive, and information-seeking reasoning in LLMs. I will discuss the complementary evaluation methodologies we adopted, combining behavioral analyses with investigations of the models' internal reasoning processes. Finally, I will argue that understanding LLM reasoning requires moving beyond static reasoning benchmarks toward dynamic, interactive, multi-turn settings. To support this claim, I will draw on experimental paradigms that have long been used in cognitive science to study human reasoning, showing how they provide powerful tools for investigating the interplay between language and reasoning in modern language models.

Bio: Raffaella Bernardi is Associate Professor at the Faculty of Engineering, Free University of Bozen-Bolzano, Italy. She works on Computational Linguistics focusing on the interplay between Language and Reasoning as displayed during conversations and/or in multimodal contexts. She is part of the Association of Logic, Language and Information (FoLLI) for which she has served in multiple roles (as Secretary, as Logic and Language Area Chair (AC) of the ESSLLI Student Session 2001, local Program Chair (PC) of ESSLLI 2016, PC chair of ESSLLI 2021, etc.). She is an active member of the international Association of Computational Linguistics (ACL), serving as a Senior Chair, AC of *ACL conferences; she is also an active member of the Italian Association of Computational Linguistics (AILC) serving as PC chair, AC, member of the scientific committee of AILC Lectures. She has recently been the EU representative within the ACL Sponsorship Board. She has also been quite active in disseminating Computational Linguistics by means of teaching activities. She has long been the local coordinator, at unibz and then at unitn, of the Erasmus Mundus European Masters Programme in Language and Communication Technologies. She is an ELLIS (European Laboratory for Learning and Intelligent Systems) elected Fellow. Currently, she is serving as Director of the MSc in Computing for Data Science (Faculty of Engineering), member at-large of the ACL Management Board, and member of AILC Management Board.

Keynote Talk

LLMs as a Synthesis between Symbolic and Distributed Approaches to Language

Gemma Boleda
Universitat Pompeu Fabra

Abstract: Since the middle of the 20th century, a fierce battle is being fought between symbolic and distributed approaches to language and cognition. In this talk, I reflect on deep learning models of language in the context of this broader discussion, and consider the possibility that LLMs represent a synthesis between the two antagonistic traditions. This possibility is supported by the fact that deep learning architectures allow for both distributed/continuous/fuzzy and symbolic/discrete/categorical-like representations and processing; what is left to see is how modern language models make use of this flexibility. I will discuss recent research in interpretability that suggests that grammar is processed in a near-symbolic fashion in modern LLMs, while lexical semantics receives a much more distributed treatment; and I will also discuss work in progress in our group that is starting to probe this view. If time allows, I will end the talk by summarizing a recent systematic review of interpretability work on syntactic knowledge in LLMs.

Bio: Gemma Boleda is an ICREA Research Professor in the Department of Translation and Language Sciences of the Universitat Pompeu Fabra (Barcelona, Spain), where she co-leads the Computational Linguistics and Linguistic Theory (COLT) research group. She previously held post-doctoral positions at the University of Trento (Italy), The University of Texas at Austin (USA), and Universitat Politècnica de Catalunya (Spain). She was also a visiting researcher at the Computational Linguistics & Phonetics department (CoLi) of Saarland University and the Institute for Natural Language Processing (IMS) of the University of Stuttgart, both in Germany. In her research, Prof. Boleda uses quantitative and computational methods to investigate how languages work; in particular, how they are shaped by human cognitive constraints, on the one hand, and communicative needs, on the other.

Keynote Talk

Where Did the Symbol Go? Faithful Explanations in the Age of Large Language Models

Mateusz Lango
Charles University

Abstract: The fluency of modern large language models may lead to a perception that natural language generation is close to solved, yet a closer look reveals a recurring pattern of failure whenever a task requires exact, discrete fidelity to some underlying structure rather than fluent pattern completion. Reasoning that is straightforward in a single turn becomes surprisingly brittle across a dialogue; generation under explicit lexical or content constraints remains difficult to guarantee; and explanations produced by LLM-as-judge systems are frequently unfaithful to the actual decision process they claim to justify. I will argue that these three failures share a common shape: each is a case where an LLM must maintain or produce something exact and discrete, but is instead relying on an implicit, continuous representation not well suited to that job. This talk focuses on the third case - faithful explanation - and asks what happens when we make the intermediate representation explicit and symbolic, rather than leaving it to the model to reconstruct implicitly.
I will present two lines of work built on this idea. The first constructs faithful explanations for image classifiers via a pipeline that separates the extraction of the (symbolic, verifiable) evidence behind a decision from its realization into fluent natural language, so that plausibility and faithfulness can be optimized somewhat independently. The second uses LLMs not to generate text directly, but to author interpretable, rule-based generation systems from scratch, producing data-to-text generators that are faithful and inspectable by construction rather than by post-hoc checking. Together, these case studies suggest a general recipe for faithful explanation: let neural models handle fluency and search, and let symbolic structure carry the guarantees. I will close with some open questions about how far this division of labor can be pushed, and what it would take to test the broader thesis motivating the talk.

Bio: Mateusz Lango is a postdoctoral researcher at the Institute of Formal and Applied Linguistics, Charles University, and an assistant professor in the Machine Learning Division at Poznań University of Technology. He completed his Ph.D. in Machine Learning in 2022 under the supervision of Prof. Jerzy Stefanowski, and now works in the group of Prof. Ondřej Dušek. His research focuses on explainable methods for NLP and machine learning more broadly, particularly on producing explanations that are both plausible and faithful. He is also interested in improving natural language generation, making it more accurate, controllable, and faithful.

Table of Contents

Program

Monday, August 3, 2026

14:05 - 15:05 *Keynote: Raffaella Bernardi*

15:05 - 15:30 *Session 1*

Discovering New Theorems via LLMs with In-Context Proof Learning in Lean
Kazumi Kasaura, Naoto Onda, Yuta Oriike, Masaya Taniguchi, Akiyoshi Sannai and Sho Sonoda

Tuesday, August 4, 2026

14:00 - 15:15 *Session 2*

Neural DTS: Integrating Hyperbolic Classifiers into Natural Language Inference Systems
Honoka Kobayashi, Hinari Daido and Daisuke Bekki

Visual Question Answering and the relation between language, perception and the world
Staffan Larsson, Bill Noble and Robin Cooper

Negation in Reasoning Traces: Interpretable Signals of Correctness and Provenance
Leon Lukas Hammerla and Alexander Mehler

Wednesday, August 5, 2026

14:00 - 15:00 *Keynote: Gemma Boleda*

15:00 - 15:25 *Session 3*

Revisiting the Systematicity in Negation in the Era of In-Context Learning
Hitomi Yanaka and Taisei Yamamoto

Thursday, August 6, 2026

14:00 - 14:20 *Lost in Transfer: Argument Mining with Large Language Models — From In-Context Learning to Policy Optimization*

14:20 - 14:40 *Neural Wani: Toward Accelerating the Automated Theorem Prover wani for Dependent Type Theory*

14:40 - 15:00 *Demo Session*

Where Does Proof Search Spend Its Effort? A Visualization and Profiling Tool for Formal NLI
Koharu Saeki and Daisuke Bekki

15:00 - 15:25 *Special: Laurence S. Moss*

Friday, August 7, 2026

14:00 - 15:00 *Keynote: Mateusz Lango*

15:00 - 15:25 *Session 4*

Semantic Distance for Presburger Formula Generation
Masaya Taniguchi and Hitomi Yanaka

Revisiting the Systematicity in Negation in the Era of In-Context Learning

Hitomi Yanaka[1,2,3,*] **and Taisei Yamamoto**[1,2,*]
[1]The University of Tokyo, [2]Riken, [3]Tohoku University
{hyanaka,yamamo96}is.s.u-tokyo.ac.jp

Abstract

Understanding the meaning of negated sentences remains one of the challenges for language models, even in the era of large language models (LLMs). We analyze systematicity regarding LLM understanding of negation from two perspectives: behavioral systematicity and representational systematicity. For behavioral systematicity, we confirm that through demonstrations and in-context learning, LLMs can recognize negation expressions and scope within sentences to some extent, but they fail to achieve perfect performance. In particular, the difficulty of the negation scope recognition for models varies depending on the output format. For representational systematicity, we analyze the extent to which function vectors can be robustly constructed from in-context examples for tasks that are essential to understanding negation. The experiments suggest that while function vectors can be composed for negation cue extraction tasks, extracting function vectors for recognizing scope is more challenging.

1 Introduction

Even in the era of large language models (LLMs), language models have struggled with understanding negation (Truong et al., 2023; García-Ferrero et al., 2023; Zhang et al., 2023, *inter alia*). One reason can be the lack of *systematicity*, the capacity to comprehend an unbounded set of structured expressions from a finite set of primitive components (Fodor and Pylyshyn, 1988). In human cognition, systematicity underlies linguistic productivity: for example, understanding the sentence "The cat didn't chase the dog" immediately confers an ability to understand "The dog didn't chase the cat", due to linked representations of the constituent concepts and their combinations.

In their foundational critique of connectionist models, Fodor and Pylyshyn (1988) argued that systematicity is a property of *representations* rather than behavior: only systems that encode structured representations can exhibit systematic generalization across contexts. They contend that architectures lacking such representational structure fail to guarantee systematic inference, even if they occasionally produce correct outputs on specific tasks. This distinction highlights that evaluating model behavior on benchmark tasks is insufficient to determine whether the model has acquired the underlying representational principles that support systematic generalization. Despite the theoretical importance of structured representations, previous model evaluation has focused on behavior alone, using monotonicity inference benchmarks involving negation that measure performance on held-out combinations of known inputs (Geiger et al., 2020; Yanaka et al., 2020; Goodwin et al., 2020).

Recently, Vegner et al. (2025) contrasted *behavioural systematicity*, which reflects performance patterns on specific test inputs, with *representational systematicity*, which concerns the internal encoding of compositional structure that supports systematic inference. They show that many benchmarks implicitly assume representational systematicity but only measure observable behavior, leading to ambiguous claims about models' cognitive capacities, and argue the necessity of methods that explicitly probe whether internal representations exhibit the structural regularities. Analyzing representational systematicity is particularly salient for LLMs because they can answer unseen questions in a benchmark through spurious correlations or memorized fragments, without possessing the compositional representations that would support systematic inference (Kumon and Yanaka, 2025). Thus, there is a pressing need for representational analyses that can reveal whether LLMs encode structured elements akin to those hypothesized in cognitive theories of systematicity. In addition, previous research has primarily focused on analyzing

[*] equal contribution.

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 1–8

the systematicity of pretrained language models such as BERT (Devlin et al., 2019), while the systematicity of LLMs based on in-context learning (ICL) has not been sufficiently explored.

In this paper, we analyze not only behavioral systematicity but also representational systematicity regarding LLM understanding of negation. For behavioral systematicity, we confirm that LLMs can, to some extent, recognize negation expressions and negation scope within sentences that do not appear in the demonstration. For the analysis of representational systematicity, we focus on the concept of function vectors (Todd et al., 2024) (alongside concurrently proposed task vectors (Hendel et al., 2023)), which are dense vectors extracted from the model's internal activations that summarize the input–output mapping implicitly demonstrated by a set of in-context examples. Specifically, we analyze the extent to which function vectors can be robustly formed for negation understanding tasks.

2 Related Work

2.1 Function Vectors in LLMs

Recent analyses of the mechanisms underlying ICL in LLMs have examined function vectors (concurrent studies include task vectors (Hendel et al., 2023) and in-context vectors (Liu et al., 2024)), which distill underlying function information from a few input-output demonstrations into a single vector. The vector can be extracted from the model's hidden activations and injected to trigger the same function in a different context.

Function vectors Consider an ICL prompt containing k input-output demonstrations followed by a query. When the prompt is processed by an LLM, each layer ℓ produces hidden states $\mathbf{h}_\ell \in \mathbb{R}^d$ at the last token position. The key hypothesis is that a small subset of attention heads transports a representation of the demonstrated function through the network. A set of attention heads that have a strong causal influence on the model's ability to perform the task can be identified by causal mediation analysis (see Appendix B for details).

Let $\mathcal{A}$ denote the set of such attention heads, each indexed by layer ℓ and head j. For each attention head (ℓ, j), the output activation for the final token of the demonstration context is extracted and averaged across multiple prompts demonstrating the same task. Denoting this activation by $\mathbf{a}_{\ell,j}$, the function vector is defined as the aggregated activation across the identified heads:

$$\mathbf{v}_{\text{func}} = \sum_{(\ell,j)\in\mathcal{A}} \bar{\mathbf{a}}_{\ell,j},$$

where $\bar{\mathbf{a}}_{\ell,j}$ is the mean activation over prompts. This vector summarizes the transformation encoded by the demonstration examples, effectively compressing the ICL task into a single vector representation. Empirically, these vectors tend to emerge in middle layers and remain robust across prompt variations.

After extraction, function vectors can be used to causally intervene in the model's forward pass. The intervention is performed by adding a function vector to the hidden state of a selected layer during inference. Let $\mathbf{h}_\ell$ denote the hidden state at layer ℓ for a new input without demonstrations. The function vector is injected via a residual intervention:

$$\mathbf{h}'_\ell = \mathbf{h}_\ell + \mathbf{v}_{\text{func}}.$$

The modified hidden state $\mathbf{h}'_\ell$ is then propagated through the remaining layers. Remarkably, prior research has shown that injecting the function vector robustly induces function execution for various tasks, even when applied to zero-shot contexts that do not contain any task information.

Task vectors Task vectors provide an alternative account of how ICL encodes task information in the hidden representations of an LLM. Rather than identifying a subset of causally important attention heads, task vectors are extracted directly from the residual stream activations induced by a set of demonstrations. The central hypothesis is that, when processing an ICL prompt, the model first computes a latent representation of the task specified by the demonstrations and then applies this representation to the query.

Consider the hidden states corresponding to a designated task-representative token position in a prompt (e.g., "→" in an ICL setting, where input-output demonstrations are described as small → big, and a target query is described as fast →). Hendel et al. (2023) observed that these hidden states exhibit a common component that is largely invariant across prompts for the same task. They called them task vectors, which serve as compact representations of the transformation specified by the demonstrations, abstracting away from the particular examples used to define the task. Like function vectors, task vectors can be used as interventions during inference. The task vector is injected

by swapping the residual stream activation of the final token in the target query ("→") with that in the ICL prompt. Empirical results show that the injected task vector can partially recover the behavior induced by in-context examples, suggesting that a substantial portion of the task information learned through ICL is encoded in a reusable vector representation within the model's hidden states.

If a function vector (in our paper, a function vector and task vector will be collectively referred to as a function vector) representing a task associated with arbitrary input can be constructed and it can execute the task, then LLMs might realize a kind of representational systematicity. However, existing research has primarily focused on tasks that can be easily inferred from demonstrations (e.g., the task to extract tokens related to animal names from a comma-separated sequence of tokens) and single-token output tasks. In contrast, this study attempts to construct a function vector for the task of understanding the meaning of negated sentences, then analyzes the extent to which the model can perform the task on unseen negated sentences when function vectors are applied in zero-shot settings.

2.2 Analysis of LLMs on Negation

Negation is one of the most fundamental linguistic phenomena, which reverses the meaning of words, phrases, and sentences. Various negation understanding benchmarks have been provided to analyze the logical reasoning abilities of language models (Hossain et al., 2020a,b; Kalouli et al., 2022; Ravichander et al., 2022; Truong et al., 2023; García-Ferrero et al., 2023; Zhang et al., 2023) to evaluate the linguistic capabilities of LLMs. Some studies of probing negation understanding have shown model insensitivity to the contextual impacts of negation (Ettinger, 2020; Kassner and Schütze, 2020; Kim et al., 2025).

At least, understanding of the functional meaning in negated sentences requires (i) the recognition of the negation expression within the sentence and (ii) the comprehension of its scope. We analyze the systematic abilities of LLMs to handle negation by focusing on these two tasks.

3 Behavioral Systematicity

3.1 Settings

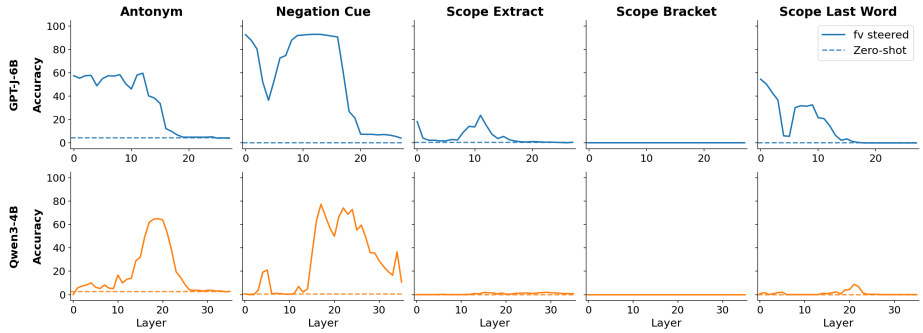
Overview We first analyze the behavioral systematicity of LLMs. Under a few-shot learning setting, we evaluate how well LLMs can correctly recognize not only the negation expression but also its scope in input sentences composed of word combinations not appearing in the demonstrations. To observe model behavior with simple and controllable settings, this study focuses on investigating the semantic understanding capabilities of LLMs in single sentences containing one negation expression. We consider (i) a negation cue extraction task that extracts a negation expression in an input sentence and (ii) a negation resolution task that predicts the part of the sentence (negation scope) affected by the extracted negation cue. As a baseline task for which function vectors can be constructed in prior research (Todd et al., 2024; Hendel et al., 2023), we also evaluate LLMs using the antonym prediction task. The task is to generate a word with an opposite meaning, given an input word (e.g., input is `old:` and its expected output is young).

I do `[NEG]`not`[/NEG]` call customer service often.	
EXTRACT	not call customer service often
BRACKET	I do not [call customer service often].
LAST-WORD	often
NER	$[0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0]$

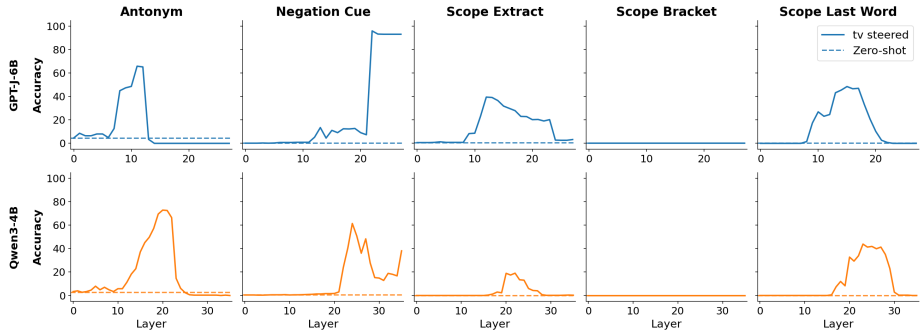
Table 1: Negation resolution task formats. To simplify the analysis, we tagged the target negation cues.

Furthermore, various output formats are conceivable for the negation resolution task. If LLMs can systematically recognize the scope of negation expressions, they should robustly predict the correct answer even when the output format differs. Thus, we investigate how well they recognize the scope of negation expressions by varying the output format. We consider four task formats shown in Table 1: EXTRACT is an extraction format that outputs the sequence of tokens within the negation scope, BRACKET is a bracket format where the negation scope is enclosed in square brackets, LAST-WORD is a last-word format that outputs the final token of the negation scope, and NER is an NER task format that returns a sequence where tokens outside the scope of negation are 0, and tokens within the scope are 1.

Dataset We used SFU Review Corpus (Konstantinova et al., 2012), which consists of 400 real-world documents of various genres manually annotated at the token level with negation expressions and at the sentence level with their linguistic scope. For simplicity of analysis, we sampled data from the corpus that contained at least one negation expression (*not*, *no*, or *n't*) followed by a scope. The



(a) Zero-shot top-1 accuracy of applying function vectors to models.



(b) Zero-shot top-1 accuracy of applying task vectors to models.

Figure 1: Representation systematicity results across tasks: antonym prediction, negation cue extraction, and negation resolution (EXTRACT, BRACKET, and LAST-WORD). The dotted line indicates the accuracy in zero-shot settings. Since we use test examples for which the model answers correctly in 10-shot settings, the accuracy in 10-shot settings is 100% in this experiment.

resulting dataset is composed of 153 training examples, 308 development examples, and 1078 test examples. For the antonym prediction task, we use the dataset developed by Todd et al. (2024), composed of 700 training examples, 90 development examples, and 210 test examples. We have confirmed that the sentences in training, development, and test sets are composed of different word combinations, respectively. For each task, we sample 10-shot in-context examples from the training set and prepend them to the test example for evaluation. Note that, since during pre-training LLMs may learn lexical items and syntactic structures that are subsequently questioned in the test set, rigorously evaluating the systematicity of LLMs is difficult. Thus, we relax the definition of systematicity; we analyze whether models obtain the ability to systematically capture the task from demonstrations without any instruction and correctly execute the task even for negated sentences containing lexical combinations not appearing in the demonstrations.

Models Following the prior research, we selected GPT-j-6B (Wang and Komatsuzaki, 2021). We also used a relatively recent LLM, Qwen3-4B (Qwen Team, 2025). We compared some baselines to verify whether models merely capture heuristics rather than understanding negation (see Appendix A).

3.2 Results

Table 2 shows accuracy for each task and format. As a general trend, Qwen3-4B performed better than GPT-j-6B. For the negation cue extraction task, both models achieved an accuracy comparable to that of the antonym prediction task. However, since this accuracy is not perfect, it suggests that they fail to acquire the systematic ability to recognize negation expressions for arbitrary negated sentences from demonstrations. For the negation resolution task, the model accuracies varied depending on the task format; models especially struggle with learning the NER format from demonstrations.

4 Representational Systematicity

4.1 Settings

Next, we investigate the representational systematicity of LLMs. We analyze the extent to which applying a function vector/task vector to each layer at the final token position of the prompt can cause

Task		GPT-j-6B	Qwen3-4B	Base.
Antonym		64.5	65.1	-
Neg-cue		62.0	64.3	64.7
Scope	EXTRACT	56.9	76.0	55.7
	BRACKET	22.6	69.8	55.7
	LAST-WORD	24.5	32.6	56.9
	NER	0.0	0.0	55.7

Table 2: Model accuracy (%) for each task and format on the 10-shot setting.

the model to perform a task in contexts that differ from the ICL contexts from which it was extracted. For each task, we compute a function vector (Todd et al., 2024) and a task vector (Hendel et al., 2023) respectively, by using 10-shot demonstrations sampled from training examples. As for a function vector setting, if the output is composed of multiple tokens, the function vector is added to all tokens following the final token.

According to Todd et al. (2024)'s evaluation setting, we only include test examples for which the model answers correctly given 10-shot in-context examples; we calculate accuracy on this filtered test set as the proportion of the model's task performance encoded by function/task vectors. We used the same datasets and models, and the NER format was excluded from the analysis due to the low model performance on the 10-shot setting.

4.2 Results

Figure 1a and Figure 1b show zero-shot top-1 accuracy of applying function vectors and task vectors to models across tasks, respectively. Regarding the negation cue extraction task, applying function vectors in middle layers and task vectors in late layers seems to have a large effect (90% accuracy for GPT-J-6B and 80% for Qwen3-4B). This indicates that a faithful function/task vector is formed for the negation cue task, similar to the antonym task.

In contrast, for the negation resolution task, the function/task vector seems to have less effect (0%-50% accuracy) than that for the negation cue task. With GPT-j-6B, there is a slight effect of steering in the EXTRACT and LAST-WORD formats, but it is minimal. This suggests that constructing function/task vectors is more challenging for the negation resolution task than for the negation cue extraction task. There are two possible reasons: (i) the task of extracting multiple tokens from sentences might be difficult to recognize, and (ii) models still struggle with capturing the concept of negation scope from in-context examples. Further investigation should reveal the bottlenecks in the LLM's systematic recognition of negation scope.

Regarding the differences between function vectors and task vectors, Figure 1a and Figure 1b suggest that task vectors can be more successfully used as interventions than function vectors in negation resolution tasks across models. In addition, task vectors tend to be more effective than function vectors at later layers. While function vectors are constructed using a small number of key attention heads from middle and late layers (see Appendix B) and represent input-output functions, task vectors are constructed from a residual stream and are thought to contain more global and abstract task information. Since later layers handle more abstract representations (Xu et al., 2026), task vectors can execute tasks more effectively at later layers. Another possible cause might be intervention differences between function vector addition and task vector replacement. Adding function vectors in later layers cannot correct information that has already been computed in previous layers. In contrast, task vectors replace the activation in the residual stream, making it possible to overwrite past information, including earlier computational results. Further analysis is needed using a wider variety of controlled tasks and formats.

5 Conclusion

We investigated LLM understanding of negation from two perspectives: behavioral systematicity and representational systematicity. For behavioral systematicity, experiments showed that through demonstrations and ICL, LLMs can recognize negation expressions and negation scope within sentences involving different word combinations to some extent, but their performance is not perfect. In particular, the difficulty of the negation resolution task for models varies depending on the output format. For representational systematicity, we analyzed the extent to which function vectors necessary for understanding negation can be robustly constructed from in-context examples. The experiments suggest that, although the function vectors for negation cue extraction can be formed, extracting the function vector for scope recognition is more challenging. Our analysis provides insights into the bottlenecks in current LLMs' systematic understanding of negation from behavior and representation perspectives, hopefully leading to improvements in LLMs' understanding of negation.

Limitations

Since LLMs may have encountered sentences from the dataset used in this study (the SFU review corpus) during pre-training or other processes, it is difficult to rigorously analyze their systematic abilities when provided with sentences involving unseen word combinations. In this study, we have adopted a slightly broader definition of systematicity and focused our analysis on the systematic abilities to capture the task from demonstrations without any instruction and execute the task for sentences involving word combinations not seen in the demonstrations.

Another limitation is that for simplicity of analysis, our data is limited to sentences involving at least one negation expression (*not*, *no*, or *n't*) followed by a scope. However, in real texts, there are various negation expressions, and sometimes the negation scope precedes the negation expression (e.g., *To Luke ten years in age difference is nothing* from the SFU review corpus). Also, as briefly introduced in Section 2.1, several function vector techniques have been explored. In future work, we will consider multiple function vector techniques and analyze models with sentences containing diverse negative expressions and scope patterns.

Lastly, although this study focuses on the model analysis in the systematic nature of negation, other systematic linguistic and inferential phenomena could also be considered. The proposed analysis framework can be extended to address polarity determination regarding monotonicity inference (Hu and Moss, 2018; Icard and Moss, 2014), enabling further analysis of LLMs. We believe that the direction of this analysis can deepen our understanding of how LLMs generalize, where they fall short, and how hidden-state interventions might yield representations that more closely resemble the compositional structure of human cognition.

Acknowledgements

We thank the anonymous reviewers for their helpful comments and constructive feedback. This work was partially supported by JST CREST Grant Number JPMJCR2565, JST BOOST Program Grant Number JPMJBY24H5, and JSPS KAKENHI Grant Number JP24H00809, Japan.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Allyson Ettinger. 2020. What bert is not: Lessons from a new suite of psycholinguistic diagnostics for language models. *Transactions of the Association for Computational Linguistics*, 8:34–48.

Jerry A. Fodor and Zenon W. Pylyshyn. 1988. Connectionism and cognitive architecture: A critical analysis. *Cognition*, 28(1–2):3–71.

Iker García-Ferrero, Begoña Altuna, Javier Alvez, Itziar Gonzalez-Dios, and German Rigau. 2023. This is not a dataset: A large negation benchmark to challenge large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8596–8615, Singapore. Association for Computational Linguistics.

Atticus Geiger, Kyle Richardson, and Christopher Potts. 2020. Neural natural language inference models partially embed theories of lexical entailment and negation. In *Proceedings of the Third BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP*, pages 163–173, Online. Association for Computational Linguistics.

Emily Goodwin, Koustuv Sinha, and Timothy J. O'Donnell. 2020. Probing linguistic systematicity. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1958–1969, Online. Association for Computational Linguistics.

Roee Hendel, Mor Geva, and Amir Globerson. 2023. In-context learning creates task vectors. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 9318–9333, Singapore. Association for Computational Linguistics.

Md Mosharaf Hossain, Antonios Anastasopoulos, Eduardo Blanco, and Alexis Palmer. 2020a. It's not a non-issue: Negation as a source of error in machine translation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3869–3885, Online. Association for Computational Linguistics.

Md Mosharaf Hossain, Venelin Kovatchev, Pranoy Dutta, Tiffany Kao, Elizabeth Wei, and Eduardo Blanco. 2020b. An analysis of natural language inference benchmarks through the lens of negation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 9106–9118, Online. Association for Computational Linguistics.

Hai Hu and Larry Moss. 2018. Polarity computations in flexible categorial grammar. In *Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics*, pages 124–129, New Orleans, Louisiana. Association for Computational Linguistics.

Thomas F. Icard and Lawrence S. Moss. 2014. Recent progress on monotonicity. *Linguistic Issues in Language Technology*, 9.

Aikaterini-Lida Kalouli, Rita Sevastjanova, Christin Beck, and Maribel Romero. 2022. Negation, coordination, and quantifiers in contextualized language models. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 3074–3085, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Nora Kassner and Hinrich Schütze. 2020. Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7811–7818, Online. Association for Computational Linguistics.

Jinsung Kim, Seonmin Koo, and Heuiseok Lim. 2025. Semantic inversion, identical replies: Revisiting negation blindness in large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 21434–21471, Suzhou, China. Association for Computational Linguistics.

Natalia Konstantinova, Sheila C.M. de Sousa, Noa P. Cruz, Manuel J. Maña, Maite Taboada, and Ruslan Mitkov. 2012. A review corpus annotated for negation, speculation and their scope. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3190–3195, Istanbul, Turkey. European Language Resources Association (ELRA).

Ryoma Kumon and Hitomi Yanaka. 2025. Analyzing the inner workings of transformers in compositional generalization. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 8529–8540, Albuquerque, New Mexico. Association for Computational Linguistics.

Sheng Liu, Haotian Ye, Lei Xing, and James Y. Zou. 2024. In-context vectors: Making in context learning more effective and controllable through latent space steering. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 32287–32307. PMLR.

Qwen Team. 2025. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*. Version 1.

Abhilasha Ravichander, Matt Gardner, and Ana Marasovic. 2022. CONDAQA: A contrastive reading comprehension dataset for reasoning about negation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8729–8755, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Eric Todd, Millicent L. Li, Arnab Sen Sharma, Aaron Mueller, Byron C. Wallace, and David Bau. 2024. Function vectors in large language models. In *Proceedings of the 2024 International Conference on Learning Representations*. ArXiv:2310.15213.

Thinh Hung Truong, Timothy Baldwin, Karin Verspoor, and Trevor Cohn. 2023. Language models are not naysayers: an analysis of language models on negation benchmarks. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 101–114, Toronto, Canada. Association for Computational Linguistics.

Ivan Vegner, Sydelle de Souza, Valentin Forch, Martha Lewis, and Leonidas A. A. Doumas. 2025. Behavioural vs. representational systematicity in end-to-end models: An opinionated survey. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 31842–31856, Vienna, Austria. Association for Computational Linguistics.

Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Ningyu Xu, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. 2026. Emergent structured representations support flexible in-context inference in large language models. *Preprint*, arXiv:2602.07794.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, and Kentaro Inui. 2020. Do neural models learn systematicity of monotonicity inference in natural language? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6105–6117, Online. Association for Computational Linguistics.

Yuhui Zhang, Michihiro Yasunaga, Zhengping Zhou, Jeff Z. HaoChen, James Zou, Percy Liang, and Serena Yeung. 2023. Beyond positive scaling: How negation impacts scaling trends of language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 7479–7498, Toronto, Canada. Association for Computational Linguistics.

A Baseline Setting

We prepared a baseline for each task. For the negation cue task, we calculated the accuracy using the negation cue *not* as the baseline. For the negation resolution task (EXTRACT, BRACKET, and NER formats), we calculated the accuracy using the negation cue and all subsequent tokens as the baseline. For the LAST-WORD format, we calculated the accuracy using the last word of input as the baseline.

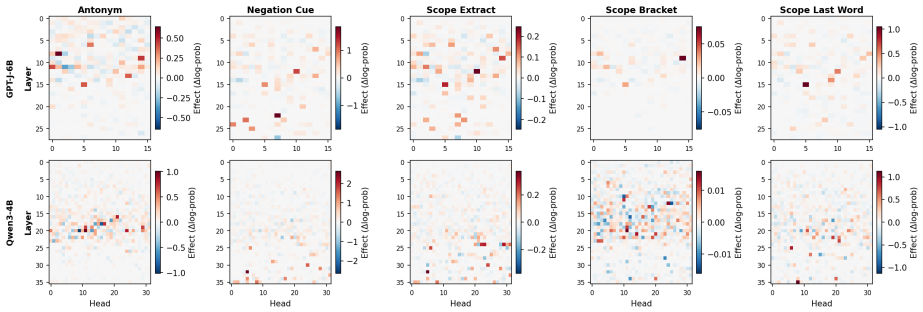


Figure 2: Average indirect effect across tasks for each attention head.

B Identifying Attention Heads for Function Vectors

Following Todd et al. (2024), we identify a set of attention heads that exhibit strong causal effects on the model's ability to perform the task. To estimate the causal effect of each head, we replace its output with $\bar{\mathbf{a}}_{\ell,j}$, the mean activation of that head computed over the development set prompts, and evaluate the model's zero-shot performance on the development set. The causal effect is defined as the average increase in the probability assigned to the correct tokens relative to the base mode (i.e., without replacement). We then select the top ten heads to construct the function vector.

Figure 2 shows the average indirect effect across tasks for each attention head in GPT-j-6B and Qwen3-4B. We can see that the attention heads with high average indirect effect appear from the middle layers to the late layers.

Neural DTS: Integrating Hyperbolic Classifiers into Natural Language Inference Systems

Honoka Kobayashi and Hinari Daido * **and Daisuke Bekki**
Ochanomizu University
{kobayashi.honoka | daido.hinari | bekki}@is.ocha.ac.jp

Abstract

Dependent Type Semantics (DTS) provides a highly rigorous framework for natural language inference (NLI), yet its scalability is severely bottlenecked by the need for manually created world knowledge. To overcome this knowledge acquisition bottleneck, we present a novel neuro-symbolic NLI system that integrates Hyperbolic Entailment Cones for automated conceptual hierarchy discovery. By exploiting the geometric properties of hyperbolic space, our model efficiently learns lexical entailment relations and dynamically injects them as logical axioms during the DTS proof-search process. Evaluations on our constructed diagnostic dataset show that our hybrid approach broadens the coverage of complex lexical variations and paraphrases without manual engineering.

1 Introduction

Computational approaches to natural language semantics can be broadly categorized into two paradigms: logical methods based on formal linguistics and data-driven methods based on large-scale neural language models. The former offers the advantage of strictly formalizing the correspondence between sentence structure and meaning, thereby guaranteeing the validity of inference through formal proofs. However, these "symbolic" methods face significant practical constraints, as essential world knowledge for reasoning must be manually provided as axioms. Conversely, neural language models exhibit a highly flexible capacity for knowledge acquisition, yet their inference processes remain opaque, making it difficult to guarantee logical consistency or explainability. From the perspective of theoretical linguistics, it has long been posited that human language comprehension involves a distinct "language faculty" that cannot be reduced solely to "general cognitive processes". Simultaneously, it is undeniable that general cognitive processes—such as association, pattern recognition, and knowledge acquisition—contribute to language understanding. Therefore, a fundamental challenge in capturing the full scope of human language comprehension lies in formally defining the interface between these two domains: integrating the structural and principled theory of the language faculty (formal linguistics) with the flexible and continuous knowledge processing mechanisms (neural embedding models).

Neural DTS is a research project aimed at elucidating this interaction between the language faculty and general cognitive processes, both theoretically and computationally. It achieves this by centering on a proof-theoretic theory of meaning, Dependent Type Semantics (DTS) (Bekki and Mineshima, 2017), as a formal linguistic model, while incorporating externally acquired neural embeddings as a model of general cognition. Toward this goal, this paper proposes and implements a novel neuro-symbolic framework that dynamically supplements world knowledge regarding conceptual entailment—which is required during the DTS proof construction process—using a *hyperbolic* classifier. Specifically, we represent conceptual entailment, such as hyponymy-hypernymy relations, using Hyperbolic Entailment Cones (Ganea et al., 2018). We introduce a classifier as an "oracle" to determine the validity of these relations on-the-fly during the proof search. By injecting the outputs of this externally trained geometric embedding into the inference rules of DTS, we extend the system's reasoning capacity into a unified framework where the continuous nature of human knowledge acquisition and the discrete, logically explainable nature of formal reasoning are integrated into a coherent whole.

*This work was conducted independently and does not reflect the views or positions of Amazon.

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 9–18

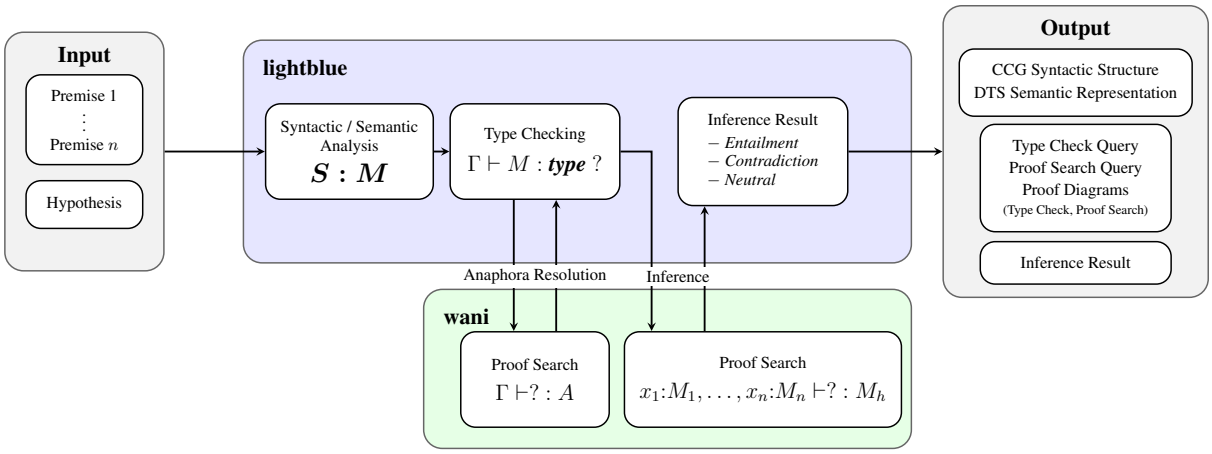


Figure 1: Natural Language Inference Pipeline

2 Previous Work

2.1 Natural Language Inference Pipeline

One of the latest advancements in linguistic-based natural language inference (NLI) is a system proposed by Tomita et al. (2025), illustrated in Figure 1. This system is composed of the CCG (Combinatory Categorial Grammar) (Steedman, 1996, 2000) syntactic parser `lightblue`[1] and the theorem prover `wani` (Daido and Bekki, 2020; Daido, 2022), which utilizes a sub-system of DTS.

When a set of premises and a hypothesis are provided as input, the process begins with `lightblue`, which performs syntactic parsing using CCG and semantic composition based on DTS. Subsequently, a type-checking process is conducted to verify whether the semantic representations of the sentences possess the sort "type". Following this, a proof search is executed by `wani`. In this stage, `wani` searches for a proof term that takes the semantic representations of the premises as assumptions and the semantic representation of the hypothesis as the conclusion. Based on the outcome of the proof search, the system yields an inference result: *Yes* if a proof term exists, *No* if a proof term exists for the negation of the hypothesis, and *Unknown* otherwise.

2.2 Neural DTS

DTS is a framework for formal semantics based on Dependent Type Theory (DTT) (Martin-Löf, 1984), capable of handling complex linguistic phenomena—such as negation, conditionals, quantification, anaphora, and presupposition—in a unified and rigorous manner. Since inference in DTS is formulated as the construction of proof diagrams within the type theory, the validity of the inference results is guaranteed. However, the cost of manually describing world knowledge as axioms, including lexical entailment relations, has been identified as a practical challenge.

To address this issue, Neural DTS (Bekki et al., 2021, 2022) was proposed as a framework to integrate DTS with neural embedding models. Bekki et al. (2021, 2022) presented the mathematical theory and learning algorithms for Neural DTS. Furthermore, Iinuma et al. (2023) worked on the implementation of Neural DTS, utilizing neural architectures such as Multi-Layer Perceptrons (MLP) and Neural Tensor Networks (Socher et al., 2013).

While their work demonstrated the practical feasibility of Neural DTS, there remains significant room for development. In particular, existing implementations relying on standard neural embedding models often operate in Euclidean spaces, which are inherently limited in their capacity to embed complex, hierarchical conceptual structures (e.g., hypernymy and hyponymy). This limitation underscores the need for a more geometrically appropriate representation of lexical entailment to further improve classification accuracy and reasoning efficiency, paving the way for the hyperbolic approach proposed in this paper.

3 The Hyperbolic Classifier

In this paper, we propose a novel approach for Neural DTS that integrates a hyperbolic classifier utilizing Hyperbolic Entailment Cones into a natural language inference system based on DTS.

[1] https://github.com/DaisukeBekki/lightblue

3.1 Advantages of Hyperbolic Embeddings

Lexical hypernymy-hyponymy relations possess a hierarchical tree structure. In such structures, the number of nodes increases exponentially with depth; however, the volume of Euclidean space grows only polynomially relative to its radius. Consequently, low-dimensional Euclidean spaces fail to arrange nodes appropriately, leading to significant structural distortion. In contrast, hyperbolic space possesses the property that its volume expands exponentially with respect to its radius. This allows for the embedding of hierarchical structures with extremely low distortion, even in low-dimensional spaces. In particular, Hyperbolic Entailment Cones, which we employ in this study, represent concepts as cones to geometrically define the asymmetric relationship of entailment. As reported by Ganea et al. (2018), this method achieves high performance in entailment detection tasks using resources such as WordNet (Bond et al., 2012).

3.2 Mathematical Definition of the Embedding Model

In Hyperbolic Entailment Cones, each concept u is embedded as a point $\boldsymbol{u}$ in the Poincaré ball $\mathbb{D}^n$, the open unit ball that serves as a model of n-dimensional hyperbolic space (throughout, we write the non-bold u for a concept and the bold $\boldsymbol{u}$ for its embedding vector). Although $\mathbb{D}^n$ is bounded in the ordinary Euclidean sense, its boundary lies infinitely far from the center O under the hyperbolic metric; consequently, moving a point outward toward the boundary corresponds to descending the taxonomy into ever more specific concepts. To avoid a singularity at the center, where the aperture defined below is undefined, an open ball $\mathcal{B}^n(O, \varepsilon)$ ($\varepsilon > 0$) around O is excluded. Intuitively, each concept u then owns a cone that fans out from its point $\boldsymbol{u}$ toward the boundary; every concept falling inside this cone is treated as a more specific instance (a hyponym) of u, so that *dog is-a animal* holds exactly when the point for *dog* lies inside the cone of *animal*. A learnable $K > 0$ parameterizes the cone's half-aperture $\psi(\boldsymbol{u})$, that is, how wide this cone opens:

$$\psi(\boldsymbol{u}) = \sin^{-1}\left(K\frac{1-\|\boldsymbol{u}\|^2}{\|\boldsymbol{u}\|}\right) \tag{1}$$

Geometrically, this aperture shrinks as u moves away from the origin, so that deep, specific concepts near the boundary own narrow cones, whereas general concepts near the center own wide cones that subsume many descendants. To ensure a valid arcsine domain, we impose the constraint:

$$K \le \frac{\varepsilon}{1-\varepsilon^2} \tag{2}$$

The geometric condition for a concept v (hyponym) to be entailed by a concept u (hypernym) is that the geodesic angle $\Xi(\boldsymbol{u}, \boldsymbol{v})$ between the two vectors must fall within the aperture of the cone formed by the hypernym u. Specifically, this is expressed by the following condition:

$$\Xi(\boldsymbol{u}, \boldsymbol{v}) \le \psi(\boldsymbol{u}) \tag{3}$$

Here the geodesic angle $\Xi(\boldsymbol{u}, \boldsymbol{v})$ measures the direction of v as seen from u relative to the cone's axis, which points radially outward from the center O: it is small when v lies straight ahead inside the cone and large when it points away. Concretely, writing $\angle Ouv$ for the angle at vertex u in the triangle Ouv, it is defined as:

$$\Xi(\boldsymbol{u}, \boldsymbol{v}) := \pi - \angle Ouv = \cos^{-1}\left(\frac{N}{D}\right) \tag{4}$$

where N and D are defined as:

$$N = \langle \boldsymbol{u}, \boldsymbol{v}\rangle(1+\|\boldsymbol{u}\|^2) - \|\boldsymbol{u}\|^2(1+\|\boldsymbol{v}\|^2)$$
$$D = \|\boldsymbol{u}\|\|\boldsymbol{u}-\boldsymbol{v}\|\sqrt{1+\|\boldsymbol{u}\|^2\|\boldsymbol{v}\|^2 - 2\langle \boldsymbol{u}, \boldsymbol{v}\rangle}$$

Here, $\|\cdot\|$ and $\langle\cdot,\cdot\rangle$ denote the Euclidean norm and the standard inner product, respectively. We reproduce this closed form from Ganea et al. (2018) for completeness; for the intuition that follows, it suffices to read $\Xi(\boldsymbol{u}, \boldsymbol{v})$ simply as the angular position of v relative to u's cone axis.

Furthermore, the energy function $E(\boldsymbol{u}, \boldsymbol{v})$ is defined as follows:

$$E(\boldsymbol{u}, \boldsymbol{v}) = \max(0, \Xi(\boldsymbol{u}, \boldsymbol{v}) - \psi(\boldsymbol{u})) \tag{5}$$

Intuitively, E measures how far v lies outside u's cone: it is zero whenever v is correctly contained, and grows with the size of the angular violation otherwise.

During model training, the following loss function $\mathcal{L}$ is minimized to reduce the energy for the positive sample set $\mathbb{P}$ while ensuring that the energy of the negative sample set $\mathbb{N}$ is at least a margin γ:

$$\mathcal{L} = \sum_{(\boldsymbol{u},\boldsymbol{v})\in\mathbb{P}} E(\boldsymbol{u},\boldsymbol{v}) + \sum_{(\boldsymbol{u}',\boldsymbol{v}')\in\mathbb{N}} \max(0, \gamma - E(\boldsymbol{u}',\boldsymbol{v}')) \quad (6)$$

In other words, training pulls genuine hyponym pairs inside their hypernym's cone, driving their energy toward zero, while pushing unrelated pairs at least a margin γ outside it.

Through this optimization, a geometrically consistent hierarchical structure is constructed within the hyperbolic space.

3.3 Data Preparation and Preprocessing

To develop a classifier capable of effectively evaluating and reasoning over world knowledge in Japanese, we construct our training dataset using the procedure outlined below, following the methodology of the previous study (Ganea et al., 2018). Our implementation is based on the official repositories provided by the authors.[2][3]

1. **Relation Extraction**: We extract hypernym-hyponym relations between nouns based on SynsetIDs from WordNet (Bond et al., 2012).

2. **Relation Classification**: By computing the transitive closure and transitive reduction of the extracted relations, we classify edges into "Basic edges," which represent direct parent-child relationships, and "Non-Basic edges," which represent other transitive relationships.

3. **Data Splitting**: The entire dataset is divided into training (Train), validation (Valid), and evaluation (Eval) sets. All Basic edges, which are essential for maintaining the hierarchical structure, are included in the training data. We then evaluate and compare model performance by varying the proportion of additional Non-Basic edges.

4. **Cross-lingual Alignment and Filtering**: To apply this learned hierarchical structure to our target Japanese vocabulary, we map the embeddings of the English SynsetIDs to their corresponding Japanese lemmas using the Open Multilingual WordNet (OMW) (Bond and Foster, 2013). By taking the intersection of the trained concepts and the available Japanese translations, we simultaneously assign the learned representations to the Japanese vocabulary and filter out any concepts that lack a Japanese translation. Importantly, this filtering is applied strictly as a post-processing step for the Eval set; the Train set retains all original English concepts to preserve the continuity and density of the underlying graph structure.

[2] https://github.com/dalab/hyperbolic_cones

[3] https://github.com/facebookresearch/poincare-embeddings

3.4 Model Training and Results

We trained models using three distinct methods—Order Embeddings (Vendrov et al., 2016), Poincaré Embeddings (Nickel and Kiela, 2017), and Hyperbolic Entailment Cones (Ganea et al., 2018)—and compared their performance. The parameters configured during training are detailed in Table 1.

Table 1: Parameter settings used during training

Item	OrderEmb	Poincaré	Hyp Cones
Dimension	5 / 10	5 / 10	5 / 10
LR	1×10^{-1}	1×10^{-4}	3×10^{-4}
Epochs	500	300	300
Optimizer	SGD	ExpMap	RSGD
Neg Samples	10	10	10
Margin (γ)	1.0	—	0.01
Init Model	None	Poincaré	Poincaré
Neg Sampling	parent	child	parent

The performance of these models, evaluated by their F1 scores on the binary entailment classification task, is summarized in Table 2 (5 dimensions) and Table 3 (10 dimensions). We assessed the performance by varying the proportion of Non-Basic edges appended to the training data.

Table 2: Experimental results (5 dimensions)

	Embedding Dimension = 5				
Model	0%	10%	25%	50%	90%
Order Emb	**38.0%**	72.7%	**78.3%**	83.3%	87.4%
Poincaré	28.6%	55.2%	73.8%	79.4%	82.4%
Hyp Cones	29.9%	**73.2%**	75.1%	**92.0%**	**93.3%**

Table 3: Experimental results (10 dimensions)

	Embedding Dimension = 10				
Model	0%	10%	25%	50%	90%
Order Emb	**46.0%**	72.4%	81.4%	87.0%	87.0%
Poincaré	30.2%	68.0%	82.3%	84.5%	86.3%
Hyp Cones	30.9%	**76.9%**	**87.9%**	**92.8%**	**93.9%**

From the results shown in Table 2 and Table 3, we observed a trend across all methods: the F1 score improved as the proportion of Non-Basic edges added to the training data increased. In particular, the high accuracy achieved by Hyperbolic Entailment Cones even in low-dimensional spaces (10 dimensions or fewer) indicates that the exponentially growing hierarchical structure of WordNet is successfully embedded with minimal distortion. These results are highly consistent with the findings of the previous study by Ganea et al. (2018), confirming that our training implementation accurately reproduces the geometric characteristics of each embedding method.

3.5 Standalone Oracle Evaluation

Building upon the models trained in the previous section, we implemented the hyperbolic classifier *oracle* in Haskell, the same language as the `wani` theorem prover, allowing it to be integrated natively into the prover for NLI. To evaluate its standalone performance, we exported the embeddings trained with 10 dimensions and 90% of the edges.

The Haskell-based *oracle* yielded an F1 score of 92.85% on the test set. Crucially, this result is directly comparable to the performance achieved during the Python-based training phase, demonstrating that the exported geometric parameters maintain their numerical stability and strict discriminative capability when loaded into a purely functional programming environment. This confirms that our Haskell implementation acts as a reliable and rigorous *oracle* for the formal proof search in DTS.

4 Integration into Natural Language Inference Systems

Next, we propose a framework for integrating the trained hyperbolic classifier into our NLI system.

4.1 Framework for Dynamic Axiom Addition

When the `wani` theorem prover receives the trained classifier as part of its input, it determines dynamically whether to invoke the hyperbolic *oracle* during its backward inference process. In `wani`, backward inference refers to the process of advancing the proof search by applying a subset of DTT rules to generate subgoals from the original premises and hypothesis. During this process, if the conclusion of a generated subgoal takes the form of a predicate (represented as a constant in DTT) applied to one or more terms—for example, $\mathbf{dog}(s, x)$—`wani` invokes the classifier. In this section, we assume that predicates represent properties of the given terms. Specifically, it searches the current premises for an entry that shares the same term but applies a different predicate, such as $\mathbf{puppy}(s, x)$. It then passes the premise's predicate **puppy** (as a hyponym candidate) and the subgoal's predicate **dog** (as a hypernym candidate) to the classifier. If the probability calculated by the classifier exceeds a threshold of 0.5, `wani` treats the semantic entailment $\mathbf{puppy} \sqsubseteq \mathbf{dog}$ as a valid logical axiom, successfully resolving that specific subgoal.

4.2 Entailment Determination by the Oracle Classifier

The *oracle* determines the entailment probability between surface-level words w_1 and w_2 by mapping them to their corresponding synset embeddings. To handle polysemy and the convergence of multiple concepts into a single Japanese lemma, we first define the entailment probability between two vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ as:

$$\delta(\boldsymbol{v}_1, \boldsymbol{v}_2) = \frac{1}{1 + e^{\Xi(\boldsymbol{v}_1, \boldsymbol{v}_2) - \psi(\boldsymbol{v}_1)}} \tag{7}$$

Let $S(w)$ be the set of vectors associated with word w. The *oracle*'s final decision logic is defined as the maximum score among all possible combinations:

$$oracle(w_1, w_2) = \max_{\boldsymbol{v}_1 \in S(w_1), \boldsymbol{v}_2 \in S(w_2)} \delta(\boldsymbol{v}_1, \boldsymbol{v}_2) \tag{8}$$

The system treats the relation as a valid logical axiom if $oracle(w_1, w_2) > 0.5$.

5 System Evaluation

We integrated *oracle* into our NLI system and evaluated its overall impact. It is important to note that the primary objective of this evaluation is not to benchmark quantitative performance on a large-scale dataset, but rather to provide a proof-of-concept (PoC) for the proposed hybrid architecture. Specifically, we aim to demonstrate that dynamic knowledge injection by *oracle* successfully bridges the "knowledge bottleneck" without compromising the logical rigor and formal provability inherent in DTS.

To this end, we first present a case study to illustrate the internal mechanism of the proof

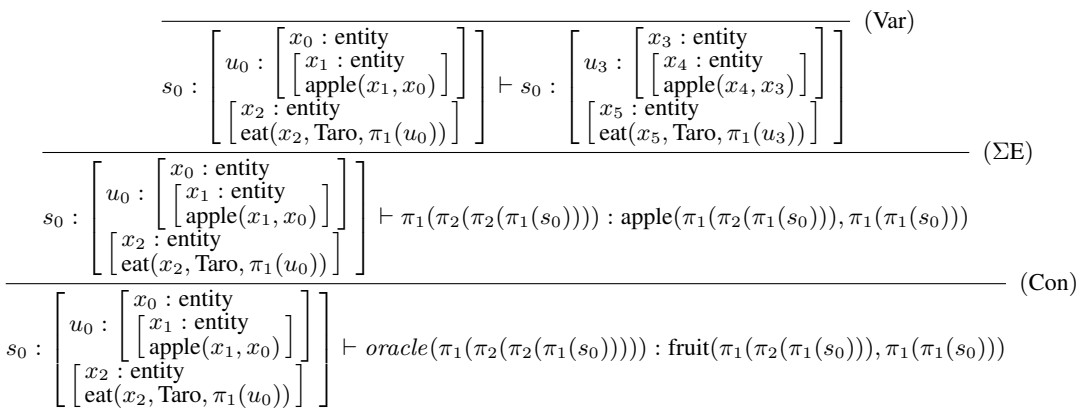


Figure 2: Part of proof search result

search, followed by a rigorous pipeline evaluation using a carefully curated set of monotonicity reasoning tasks.

5.1 Case Study: Lexical Entailment with Oracle

To verify the functional integrity of the integrated *oracle*, we consider a representative upward monotonicity task involving a hyponymy-hypernymy relation:

> **Premise:** Taro eats an apple.
> **Hypothesis:** Taro eats a fruit.

In conventional logic-based NLI systems, this inference would typically result in *Unknown* unless the specific lexical knowledge—*every apple is a fruit*—is manually encoded as a logical axiom. In contrast, our proposed framework successfully yields a *Yes* result without requiring additional explicit premises.

Figure 2 illustrates a fragment of the proof diagram generated for this case. The diagram follows the standard proof-theoretic notation of DTS: a judgment $\Gamma \vdash M : A$ asserts that the proof term M inhabits type A under context Γ; (ΣE) is the elimination rule for Σ-types (dependent pairs); and the projection operators π_1 and π_2 retrieve the first and second components of such a pair, respectively. We omit a full presentation of the DTS inference rules here and refer the reader to the DTS literature (Bekki and Mineshima, 2017; Bekki et al., 2023) for their formal definitions. As shown in the figure, when the entailment relationship between *apple* and *fruit* becomes necessary during the proof search, `wani` internally invokes the *oracle*. The hyperbolic classifier determines the validity of this relation based on its embedding space (e.g., computing an entailment probability that exceeds the 0.5 threshold) and dynamically injects it as a logical axiom into the reasoning process. This confirms that the system can bridge the knowledge gap through a hybrid approach combining formal logic and continuous geometric embeddings.

5.2 Dataset Construction

Based on the successful verification in the case study, we proceeded to a pipeline evaluation. Rather than relying on a large-scale, noisy dataset, we constructed a highly controlled, diagnostic Japanese test suite by manually translating and adapting 20 specific test cases from the Monotonicity Entailment Dataset (MED) (Yanaka et al., 2019).

The rationale for selecting monotonicity reasoning as our benchmark is that it rigorously tests a system's ability to integrate formal logical structure with continuous lexical knowledge. While the direction of entailment is governed by the logical context (e.g., upward or downward monotonicity), the actual proof often requires implicit world knowledge that is absent from the premise.

Specifically, we focused on 20 scenarios of upward monotonicity, where an entailment validly holds from a specific concept to a more general one in an upward-entailing context. Because our *oracle* is designed to dynamically supplement this missing hierarchical knowledge, these curated cases provide an ideal environment to evaluate the precise synergy between DTS-based logical inference and the hyperbolic classifier, strictly isolating the reasoning process from irrelevant

syntactic parsing noise.

5.3 Experimental Results

Tables 4 and 5 present the confusion matrices of the inference results before and after the integration of the *oracle*, respectively, where rows correspond to system predictions and columns to gold labels.

Table 4: Before integration

	Yes	No	Unk	Other
Yes	0	0	0	0
No	0	0	0	0
Unk	10	0	10	0
Other	0	0	0	0

Table 5: After integration

	Yes	No	Unk	Other
Yes	8	0	1	0
No	0	0	0	0
Unk	2	0	9	0
Other	0	0	0	0

Table 6: Detailed performance metrics per class (After integration)

Class	**Precision**	**Recall**	**F1-score**
Yes	0.889	0.800	0.842
Unknown	0.818	0.900	0.857
Macro Average	0.854	0.850	0.850

Table 6 demonstrates the substantial impact of the *oracle* on the NLI system. Most notably, recall for the *Yes* class rises from 0.0% to 80.0%. This indicates that the vast majority of upward entailment cases, which previously caused the system to default to an *Unknown* state, are now resolvable. This confirms that the hyperbolic classifier functions effectively as a dynamic knowledge bridge, enabling the DTS-based inference engine to complete proofs by identifying hierarchical relations in a robust, data-driven manner.

Furthermore, the precision of 88.9% and F1-score of 84.2% for the *Yes* class demonstrate that the *oracle* does not merely provide random guesses but rather offers highly reliable lexical information. The high precision is particularly noteworthy in a logic-based framework like DTS, where even a single incorrect axiom can lead to a logically sound but factually false conclusion. The fact that the system maintains high precision while substantially increasing recall suggests that the geometric properties of the hyperbolic embedding space—specifically its ability to model asymmetric entailment—align well with the requirements of formal monotonicity reasoning.

Additionally, the macro-averaged F1-score of 85.0% confirms the overall robustness of the hybrid system, ensuring that the integration of the neural components does not degrade the performance on neutral cases. In summary, these results provide empirical evidence that the hybrid architecture of Neural DTS effectively mitigates the "knowledge bottleneck" problem. By decoupling logical structure from continuous lexical knowledge, the system achieves a degree of flexibility and coverage that was previously unattainable for purely logic-based NLI systems, all while retaining the formal provability of DTS.

6 Error Analysis

During inference, we observed several cases in which the probability assigned by the *oracle* to a hypernym-hyponym relation fell below the decision threshold of 0.5, resulting in incorrect entailment predictions at the sentence level. A representative example is shown below:

> **Premise:** Hanako gave Taro a rose.
> **Hypothesis:** Hanako gave Taro a flower.

In this case, successful inference requires the *oracle* to capture the taxonomic relation between *rose* and *flower*. However, the model produced a score of $oracle(\text{flower}, \text{rose}) = 0.0844$, which is well below the threshold, leading to a misclassification.

To determine whether such errors arise from the threshold setting or from more fundamental limitations of the embedding model, we conducted additional analyses focusing on word-substitution pairs contained in the MED dataset.

6.1 Methodology

To systematically analyze the probability distributions output by *oracle* for the substituted word pairs, we extracted and preprocessed evaluation pairs using the following procedure:

1. **Data Extraction**: We extracted 1,403 instances containing "lexical knowledge" in the genre field from the MED dataset.

2. **Word Pair Identification and Noise Removal**: We compared the premise and hypothesis sentences to extract only the substituted word pairs. During the process, functional words such as prepositions and personal pronouns, which introduce noise during translation or evaluation, were excluded.

3. **Translation and Filtering**: The extracted English word pairs were manually translated

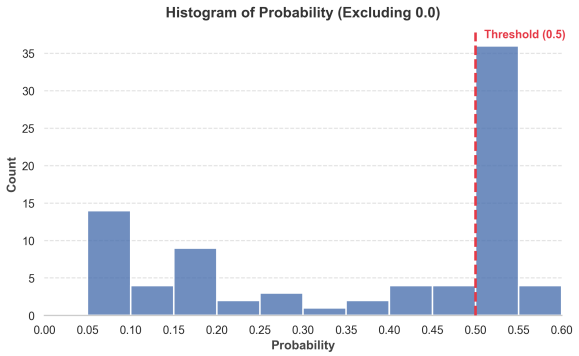


Figure 3: Probabilities for Entailment pairs ($N = 100$).

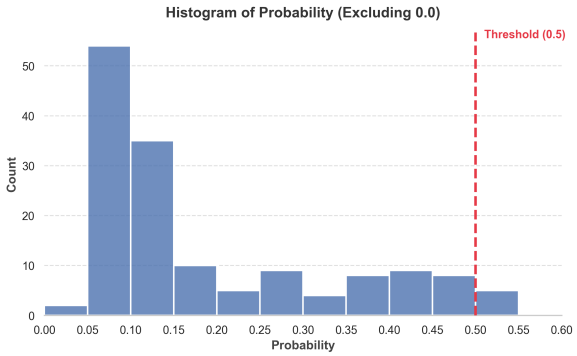


Figure 4: Probabilities for Neutral pairs ($N = 180$).

into Japanese. Pairs lacking an appropriate Japanese translation or those resulting in identical translated words were excluded.

4. **Normalization of Inference Direction**: We aligned the directional relationship between the premise word w_1 and the hypothesis word w_2 based on sentence monotonicity. For **upward-entailing** contexts, we evaluated the pair as (w_2, w_1) to test the hyponym-hypernym relation ($w_1 \sqsubseteq w_2$). For **downward-entailing** contexts, where the entailment direction is reversed, we aligned the pair as (w_1, w_2) to evaluate $w_2 \sqsubseteq w_1$. This ensures the *oracle* consistently measures the taxonomic subordination required for sentence-level entailment.

5. **Selection of Evaluation Pairs**: From the processed data, we constructed a final set of unique evaluation pairs categorized as Entailment and Neutral, ensuring no duplication.

6.2 Evaluation Results

Through the procedure described above, we obtained 100 Entailment pairs and 180 Neutral pairs. We calculated the entailment probabilities for these pairs using the *oracle* and classified them with a threshold of 0.5. Note that pairs containing out-of-vocabulary (OOV) words were excluded from the accuracy calculations.

6.3 Discussion

Mismatch Between Strict Hypernymy and Meronymy: As shown in Figure 3, the relatively low accuracy for Entailment pairs (62.12%) is primarily due to the structural divergence between the strict hypernymy modeled by the *oracle* and the broader semantic relations in the MED dataset. Specifically, many failure cases involve meronymy (part-whole relations). As shown in Table 7, pairs such as "Mt. Fuji–Japan" represent valid contextual entailments; however, their relative positions in the embedding space do not satisfy the geometric inclusion criteria of the hypernymic cones. Since the *oracle* is mathematically optimized to capture hypernymy, it assigns low probabilities to these relations. This result demonstrates that the model is behaving exactly as specified—preserving high specificity by excluding non-hypernymic pairs—even if it leads to lower recall in the presence of semantic diversity.

Table 7: Examples of Entailment pairs in MED misclassified by *oracle* due to non-taxonomic relations.

Hyper	Hypo	Score
Song	Lyrics	0.2617
Europe	Spain	0.1084
Skin	Epidermis	0.1696

High Specificity in Neutral Pairs: Conversely, as shown in Figure 4, the accuracy for the Neutral pairs demonstrated an extremely high value of 96.64%. This indicates that false positives—where the model erroneously outputs a high probability for word pairs inherently lacking a hierarchical relationship—rarely occur. This high specificity confirms that the *oracle* is highly reliable, consistently avoiding the hallucination of invalid logical axioms during the proof search.

The Challenge of Out-of-Vocabulary Words: For word pairs containing OOV words, the *oracle* defaults to a probability of 0.00. An analysis of these excluded pairs revealed that the majority consist of specific geographic names, personal names, or highly specific concepts (e.g., "leafy vegetables" or "hummingbirds"). This highlights an issue with the vocabulary coverage of the

pre-trained embeddings rather than a limitation of the geometric model or the threshold setting. Expanding the training dataset to improve vocabulary coverage remains a future challenge.

Summary and Threshold Justification: The above analysis demonstrates that the observed inference failures are not primarily caused by an inappropriate threshold. Instead, they stem from a fundamental mismatch between the *oracle*'s strict geometric design (optimized for IS-A relations) and the semantic diversity present in the evaluation dataset. Lowering the threshold indiscriminately to capture meronymy would compromise the high specificity (96.64%) observed for Neutral pairs. Therefore, retaining the current threshold of 0.5 is mathematically and practically appropriate for rigorous hypernym-hyponym classification.

7 Conclusion

In this study, we proposed and implemented a novel framework for integrating a hyperbolic classifier, based on Hyperbolic Entailment Cones, into a natural language inference system grounded in DTS. We demonstrated that the proposed hybrid system can automatically derive dynamic logical axioms to correctly resolve lexical entailments, which were previously impossible to prove without manual axiom creation. These results highlight the effectiveness of combining a broad coverage of continuous geometric embeddings with the formal reliability and explainability inherent in linguistically-oriented NLI systems.

Furthermore, our error analysis confirmed the high "geometric rigor" of the proposed *oracle*. Although the evaluation dataset contained positive entailment examples based on non-IS-A relations such as meronymy, our model successfully rejected them. This confirms that the approach safely injects external lexical knowledge while preserving strict geometric classification accuracy for taxonomic relationships.

To address current limitations, specifically misclassifications caused by out-of-vocabulary words, reconsidering and expanding the training dataset is an immediate next step. For future work, we plan to extend the model's inferential scope by mapping a broader set of semantic relationships, such as meronymy, into distinct geometric representations within the hyperbolic space, ultimately aiming to establish the proposed framework as the foundation for designing a highly versatile Neuro-Symbolic system.

Limitations

While our proposed framework successfully bridges the knowledge bottleneck for hypernymy in logic-based NLI, several limitations remain.

First, the current *oracle* is strictly optimized for taxonomic IS-A relations. As discussed in the Error Analysis, it struggles with other contextually valid semantic relations, such as meronymy (part-whole relations). Expanding the hyperbolic embedding space to accommodate multiple distinct relational structures is necessary for broader NLI coverage.

Second, the system's reasoning capacity is strictly bounded by the vocabulary coverage of the pre-trained WordNet embeddings. Out-of-vocabulary (OOV) words, particularly highly specific concepts and proper nouns, currently default to a zero entailment probability, leading to inference failures.

Finally, our pipeline evaluation was conducted on a small, carefully curated diagnostic set of 20 upward-monotonicity cases in Japanese, designed as a proof-of-concept rather than a large-scale benchmark. Scaling this evaluation to a substantially larger and more diverse test suite is a natural next step. We also restricted the present study to upward-entailing contexts. Downward-entailing contexts are logically more involved, primarily because they typically embed negation and other polarity-reversing operators. However, DTS already represents negation and such operators natively within its proof system, and the *oracle* supplies only lexical IS-A axioms whose validity is independent of the surrounding logical polarity. Because the proof search consumes these axioms in the same way regardless of monotonicity direction, we expect the extension to downward-entailing contexts to be conceptually straightforward; the principal remaining work is empirical validation at scale, which lies beyond the scope of this proof-of-concept and is left for future work.

Acknowledgments

This work was supported in part by JST CREST Grant Number JPMJCR2565 and JSPS KAKENHI Grant Number JP23H03452.

References

Daisuke Bekki and Koji Mineshima. 2017. Context-passing and underspecification in dependent type semantics. In *Studies in Linguistics and Philosophy*, pages 11–41.

Daisuke Bekki, Koji Mineshima, and Elin McCready, editors. 2023. *Logic and Engineering of Natural Language Semantics: 19th International Conference, LENLS19, Tokyo, Japan, November 19–21, 2022, Revised Selected Papers*, volume 14213 of *Lecture Notes in Computer Science*. Springer.

Daisuke Bekki, Ribeka Tanaka, and Yuta Takahashi. 2021. Integrating deep neural networks with dependent type semantics. In *Proceedings of the Symposium Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021)*.

Daisuke Bekki, Ribeka Tanaka, and Yuta Takahashi. 2022. Learning knowledge with Neural DTS. In *Proceedings of the 3rd Natural Logic Meets Machine Learning Workshop (NALOMA III)*, pages 17–25.

Francis Bond, Timothy Baldwin, Richard Fothergill, and Kiyotaka Uchimoto. 2012. Japanese SemCor: A sense-tagged corpus of Japanese. In *Proceedings of the 6th International Conference of the Global WordNet Association (GWC-2012)*, pages 56–63.

Francis Bond and Ryan Foster. 2013. Linking and extending an open multilingual Wordnet. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1352–1362, Sofia, Bulgaria. Association for Computational Linguistics.

Hinari Daido. 2022. Development of an automated theorem prover for the fragment of dts. Master thesis, Ochanomizu University.

Hinari Daido and Daisuke Bekki. 2020. Development of an automated theorem prover for the fragment of dts. In *In the 17th International Workshop on Logic and Engineering of Natural Language Semantics (LENLS17)*.

Octavian-Eugen Ganea, Gary Bécigneul, and Thomas Hofmann. 2018. Hyperbolic entailment cones for learning hierarchical embeddings. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1646–1655.

Mizuki Iinuma, Yuta Takahashi, Sora Tagami, and Daisuke Bekki. 2023. Neural DTS: A hybrid NLI system combining two procedural approaches. In *Proceedings of Procedural and computational models of semantic and pragmatic processes, ESSLLI2023 workshop, Ljubljana, Slovenia.*

Per Martin-Löf. 1984. *Intuitionistic Type Theory, Vol. 9*. Bibliopolis Naples.

Maximilian Nickel and Douwe Kiela. 2017. Poincaré embeddings for learning hierarchical representations. In *Advances in Neural Information Processing Systems (NIPS 2017)*, pages 6338–6347.

Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. 2013. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems (NIPS 2013)*, pages 926–934.

Mark Steedman. 1996. *Surface Structure and Interpretation*. The MIT Press, Cambridge.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press.

Asa Tomita, Mai Matsubara, Hinari Daido, and Daisuke Bekki. 2025. Natural language inference with CCG parser and automated theorem prover for DTS. In *Proceedings of the Second Workshop on the Bridges and Gaps between Formal and Computational Linguistics (BriGap-2)*, pages 1–7.

Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2016. Order-embeddings of images and language. In *Proceedings of the 4th International Conference on Learning Representations (ICLR 2016)*.

Hitomi Yanaka, Koji Mineshima, Daisuke Bekki, Kentaro Inui, Satoshi Sekine, Lasha Abzianidze, and Johan Bos. 2019. Can neural networks understand monotonicity reasoning? In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 31–40.

Negation in Reasoning Traces: Interpretable Signals of Correctness and Provenance

Leon Hammerla and Alexander Mehler
{hammerla · mehler}@em.uni-frankfurt.de
Goethe University, Frankfurt am Main, Germany

Abstract

Chain-of-thought (CoT) reasoning is widely used in large language models (LLMs), but the resulting reasoning traces remain underexplored. We study these traces through the lens of discourse-level negation. Specifically, we distinguish between *corrective* negation, which rejects a prior reasoning step, and *refining* negation, which narrows or qualifies it, and introduce metrics to quantify their use in human- and LLM-authored reasoning traces. Across multiple benchmarks, we find that negation occurs much more frequently in intermediate reasoning traces than in final response texts. We then test whether negation-based features provide predictive and descriptive signal for correctness, model identity, and human-vs.-LLM authorship. For correctness prediction, negation-based features consistently outperform simple structural baselines and in several settings add complementary signal to embedding-based representations, although embeddings remain stronger overall. In a controlled comparison on correct human and LLM traces from the same dataset, our strongest results arise in human-vs.-LLM classification, where negation features outperform both structural and embedding baselines. Overall, these findings position discourse-level negation as an interpretable feature for reasoning-trace analysis, with especially strong utility for provenance-related classification and modest but consistent value for correctness prediction.

1 Introduction

Chain-of-thought (CoT) has made intermediate reasoning traces a prominent object of study in large language models (LLMs) (Wei et al., 2022). CoT presents problem solving as a sequence of intermediate steps that move from an initial, possibly flawed hypothesis toward a more accurate conclusion. CoT's stepwise reasoning loosely resembles classical forms of human inquiry. For example, the Socratic method (Benson, 2011) deepens understanding through iterative questioning, while the Hegelian dialectic (Forster, 1993) advances by synthesizing opposing ideas into higher-order insights. These analogies show how structured reasoning sequences refine conclusions (Pei et al., 2025; Abdali et al., 2025), using negation (Schon et al., 2021) to signal correction, opposition, or refinement. In reasoning, negation may appear in at least two operations: (1) as a corrective operation that rejects a prior line of thought entirely, or (2) as a refining operation that narrows and sharpens reasoning without discarding it completely. We hypothesize that these two operations are salient discourse markers of stepwise reasoning and are reflected in both human- and model-generated reasoning traces. Modeling how LLMs employ negation during reasoning can provide insights into the dynamics of stepwise reasoning, as well as into variation in response correctness and trace style. However, while prior work on LLM reasoning traces has examined their structure (Bogdan et al., 2025), interpretability (Bhambri et al., 2025), and effectiveness (Gandhi et al., 2025), the role of negation within such traces remains largely unexplored, with few exceptions (Ye et al., 2023). We address this research gap by making three contributions:

1. We build a dataset of human and LLM reasoning traces, auto-labeled for two negation types and partially validated through human evaluation.

2. We introduce several metrics to quantify negation in human and LLM reasoning traces.

3. We use these metrics to examine how negation patterns in reasoning traces relate to response correctness, model identity, and human authorship.

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 19–39

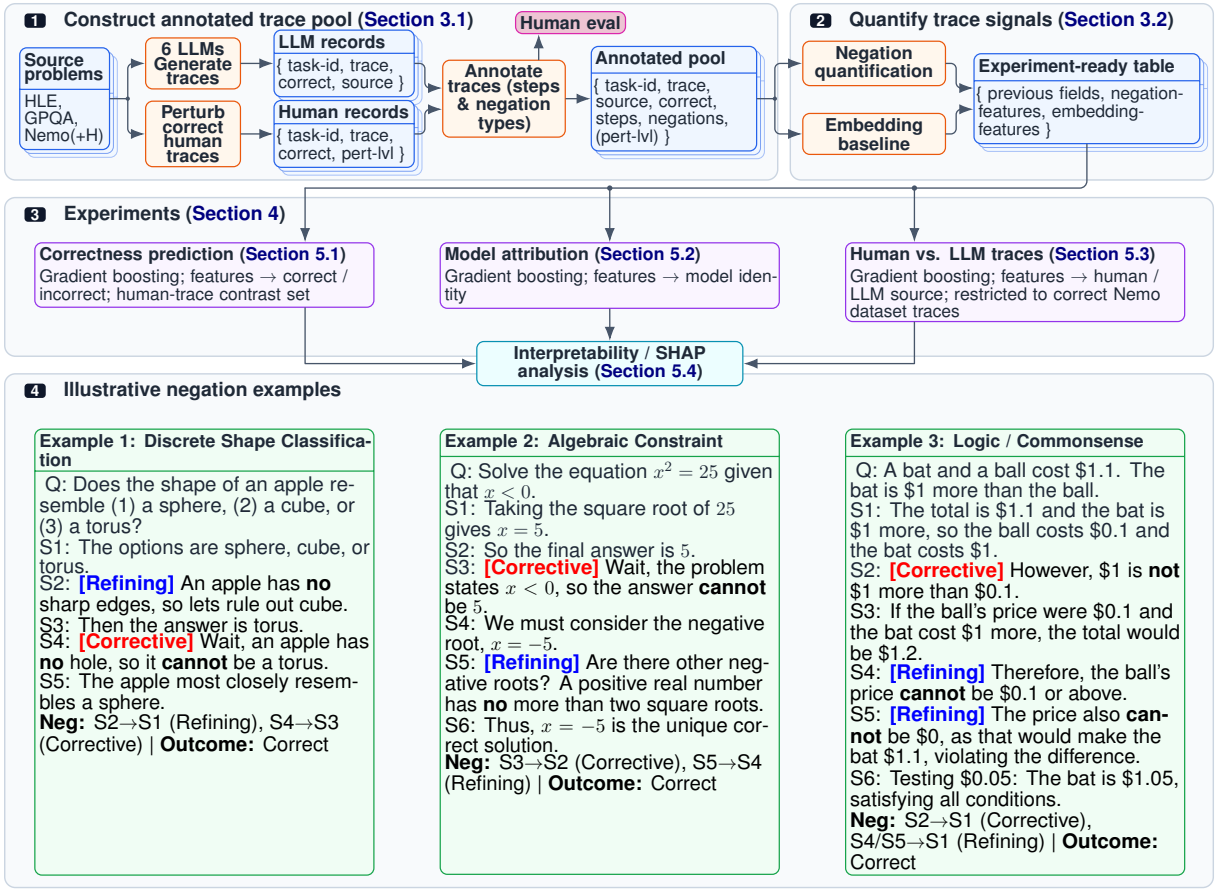


Figure 1: Pipeline for constructing our annotated reasoning-trace dataset, extracting negation-sensitive features, and evaluating the resulting representations across different tasks. Representative annotated traces illustrate our distinct negation types.

2 Related Work

2.1 Reasoning Traces in LLMs

Chain-of-thought (CoT) prompting (Wei et al., 2022) established that large language models can often improve performance on multi-step tasks when prompted to produce intermediate reasoning steps. Subsequent work expanded this general paradigm through methods such as self-consistency (Wang et al., 2023), Tree/Graph of Thoughts (Yao et al., 2023; Besta et al., 2024), and Program of Thoughts (Chen et al., 2023), which vary the structure or execution of intermediate reasoning. More recently, reasoning-oriented open-weight models such as Qwen3 (Yang et al., 2025), DeepSeek-R1 (Guo et al., 2025), and gpt-oss (OpenAI et al., 2025) have made such traces increasingly accessible through explicit "thinking" or reasoning modes.

2.2 Analyzing LLM Reasoning Traces

As reasoning traces have become more available, recent work has shifted attention from final-answer accuracy alone to the analysis of the traces themselves. Prior studies examine, among other things, which intermediate steps are most influential for correct answers (Bogdan et al., 2025; Zhang et al., 2025a), how traces align with structured knowledge sources (Zhang et al., 2025b), how interpretability relates to model performance (Bhambri et al., 2025), and how reasoning length affects controllability and effectiveness (Marjanovic et al., 2026). Other work studies reasoning chains as a route to model improvement (Gandhi et al., 2025), compares LLM reasoning with human cognition (Marjanovic et al., 2026), and questions the extent to which LLM traces reflect logical reasoning (Xu et al., 2025). Overall, this literature has shown that reasoning traces can be studied in terms of usefulness, step importance, faithfulness, interpretability, and error propagation, rather than only through final prediction accuracy.

2.3 Negation, Discourse, and Self-Revision in Reasoning

Our work is most closely related to research that treats reasoning traces as structured discourse rather than merely as sequences of tokens or steps. In linguistic and discourse-theoretic terms, negation plays a central role in rejection, correction, narrowing, and contrast (Horn, 1989; Fălăuş, 2020), and argumentative work has shown that it can function to counter or delimit prior commitments (Apothéloz et al., 1993). These functions are especially relevant to stepwise reasoning, where later steps often revise, restrict, or overturn earlier ones. In this sense, corrective and refining negation can be viewed as discourse-level markers of self-revision, complementing broader frameworks that analyze reasoning traces in terms of cognitive or functional elements (Kargupta et al., 2025). However, despite growing interest in LLM reasoning traces, prior work has rarely operationalized explicit discourse-level linguistic phenomena within them. Most existing studies focus on performance, length, structure, interpretability, or faithfulness, rather than on how particular linguistic devices organize revision across steps. Negation in particular has received little systematic attention as a trace-level feature, despite its natural connection to error correction, hypothesis narrowing, and self-revision during reasoning. We address this gap by modeling corrective and refining negation as interpretable discourse features and testing whether their distribution in reasoning traces carries signal about correctness, model identity, and human-versus-LLM authorship.

3 Methodology

3.1 Dataset

We construct our dataset from three sources[1]: three established benchmarks (GPQA Diamond (gpqa) (Rein et al., 2024), AIME 25 (aime25) (Zhang and Math-AI, 2025), and the Nemotron-Math-HumanReasoning (nemo) dataset (Du et al., 2025), which provides human-authored CoT traces for Olympiad-level mathematics problems from the AoPS forum (comparable in difficulty to AIME 25). In the first stage, we prompt six reasoning-capable LLMs to solve each benchmark task (full model list in Figure 2). The prompt instructs models to generate an explanation and return a JSON-formatted

[1]Our datasets are hosted on Hugging Face at https://huggingface.co/RT-NEG.

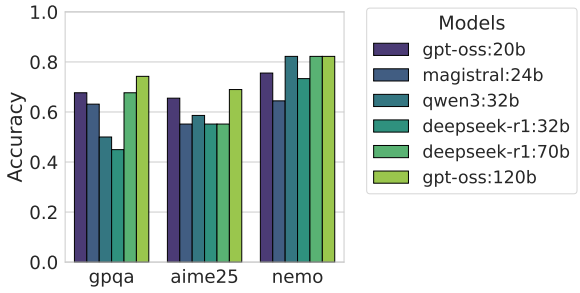


Figure 2: Benchmark results of all models evaluated on the gpqa, aime25, and the nemo dataset.

conclusion for unambiguous parsing (prompt template in Appendix F.1). The intermediate reasoning trace is implicitly generated by each model's native reasoning process. As the human dataset contains only correct solutions, we generate incorrect examples to balance prediction tasks. We use Llama3.3-70B as a meta-annotator to inject human-style reasoning errors at three severity levels, each yielding an incorrect conclusion. These corrupted traces enable us to construct both positive and negative human samples (prompt template in Appendix F.2). In the second stage, we annotate all LLM and human reasoning traces using Llama3.3-70B. Following (Bogdan et al., 2025), the annotator first segments each trace into reasoning steps, typically aligned with individual sentences. It then labels each step with one of two negation types using a dedicated prompt (see Appendix F.3). Building on linguistic theories of negation as a logic-based means to cancel alternatives (Horn, 1989; Fălăuş, 2020) and argumentative models showing its use in countering or delimiting conclusions (Apothéloz et al., 1993), we distinguish two pragmatic functions of explicit negation in reasoning: (1) **refining negation**, which rules out previously considered alternatives to narrow the solution space, and (2) **corrective negation**, which explicitly rejects or revises a prior step, sometimes without offering an alternative. Both types span reasoning steps rather than single sentences, reflecting how negation dynamically manages discourse commitments and alternatives (Büring, 2008; Krifka, 2003).

3.1.1 Preliminary Analysis

Before introducing our negation metrics, we conduct four preliminary checks to verify that reasoning traces are suitable for discourse-level negation analysis. First, across datasets, *gpt-oss* performs strongest overall, with *gpt-oss:120b* leading in most settings, while *DeepSeek-R1:70b* re-

mains competitive on *gpqa* and *nemo* but weaker on *aime25* (Figure 2). Second, using D-Neg (Hammerla et al., 2025), which builds on NegBERT (Khandelwal and Sawant, 2020), we find that reasoning traces contain substantially more negation than final responses in almost all dataset-model combinations, often by a factor of 2× to 30×. This indicates that negation is considerably more salient in intermediate reasoning than in answer-only text (Figure 3). Third, we perform a coarse sanity check on our distinction between refining and corrective negation. Across datasets, refining negations show a positive aggregate association with accuracy, whereas corrective negations show a negative one. We treat this only as supporting evidence that the two categories capture different aspects of reasoning behavior, rather than as a standalone predictive result.[2] Finally, to assess annotation reliability, two linguists independently reviewed 36 reasoning traces[3] sampled across datasets and models. Although this sample is limited at the trace level, it comprises 634 reasoning steps, each of which required a judgment about whether it contains discourse-relevant negation and, when applicable, which negation type it instantiates. Inter-annotator agreement between the two human annotators was moderate ($\kappa = 0.68$), whereas agreement between the LLM annotator and the human annotators was lower ($\kappa = 0.37$ and 0.38), as expected for a challenging discourse-level phenomenon (Mirzakhmedova et al., 2024; Li et al., 2025). We therefore treat the resulting labels as weak but operationally useful proxy annotations for aggregate analysis.

3.2 Metrics

We quantify negation to examine its role in reasoning, where it signals self-monitoring, error detection, and logical refinement (Horn, 1989). Refining negations often reflect hypothesis adjustment; corrective ones mark prior inconsistencies. Quantifying them reveals how reasoning unfolds and links to performance or reasoning specifics of LLMs.

3.2.1 Metric Design

To capture the complexity of negation use, we designed metrics across three key dimensions: **composition**, which captures the frequency, presence pattern, and type balance of negations; **position**, which captures when negations appear and thus the timing of revisions; and **structure**, which captures how negations cluster or persist across steps, indicating reflective consistency.

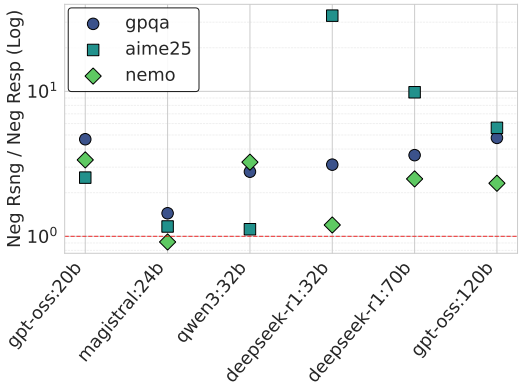


Figure 3: Ratio of average number of negations per 100 tokens in reasoning traces relative to responses across multiple datasets and models. A value greater than one (see red dashed line) indicates that negations occur more frequently in reasoning traces than in responses.

1. Compositional Trace Statistics This category captures the compositional properties of negations within a reasoning trace. Specifically, it indicates whether refining and/or corrective negations are present, and quantifies the frequency of each negation type relative to the total number of reasoning steps. It also includes a score for the balance between corrective and refining negations, providing insight into how evenly the two forms are represented within the trace:

- **Norm Neg:** Number of negations type t normalized by total number of reasoning steps:

$$\tilde{N}^t_{\text{neg}} = \frac{N^t_{\text{neg}}}{N_{\text{rstep}}}$$

  where $N_{\text{rstep}} = |T|$ is the total number of reasoning steps in reasoning trace T and $N^t_{\text{neg}} = |\text{Negs}^t|$ is the total number of corrective and refining negations in the reasoning trace ($t \in \{\text{cor}, \text{ref}\}$ is the negation type and Negs^t the set of all negations in T of type t).

- **RT Label:** Categorical label indicating which types of negation appear in the trace:

$$L = \begin{cases} 1, & N^{\text{cor}}_{\text{neg}} > 0 \text{ and } N^{\text{ref}}_{\text{neg}} = 0 \\ 2, & N^{\text{cor}}_{\text{neg}} = 0 \text{ and } N^{\text{ref}}_{\text{neg}} > 0 \\ 3, & N^{\text{cor}}_{\text{neg}} > 0 \text{ and } N^{\text{ref}}_{\text{neg}} > 0 \\ 0, & \text{otherwise} \end{cases}$$

[2] See Appendix B for full methodology, binning, and smoothed trends.

[3] Using the same guidelines as the LLM; see Appendix F.3.

- **Diversity Score:** Measures the balance of negation types in trace T:

$$D = \sum_{t \in \{\text{cor,ref}\}} p_t^2$$

where $p_t = \frac{N_{\text{neg}}^t}{N_{\text{neg}}^{\text{cor}} + N_{\text{neg}}^{\text{ref}}}$.

2. Positional Characteristics These metrics capture *where* negations occur within reasoning traces (e.g., average normalized position, first/last negation type, and frequency in the first vs. second half). They reveal whether negations tend to appear early, late, or evenly, reflecting the temporal dynamics of self-monitoring.

- **Norm Avg Pos:** Average position of each negation type t in the trace T, normalized by total number of reasoning steps:

$$\tilde{P}^t = \frac{1}{N_{\text{neg}}^t} \sum_{i=1}^{N_{\text{neg}}^t} \frac{\text{step\_idx}(\text{Negs}_i^t)}{N_{\text{rstep}}}$$

- **Edge Neg Type** Type of the first and last negation in the trace:

$$L_{\text{edge}} = \begin{cases} 1, & \text{ntype}(\text{Negs}_{\text{edge}}) = \text{cor} \\ 2, & \text{ntype}(\text{Negs}_{\text{edge}}) = \text{ref} \\ 0, & \text{otherwise} \end{cases}$$

where edge $\in \{\text{first}, \text{last}\}$ and Negs is the set of all negations in trace T.

- **Neg Frequency in first vs. second half:** Ratio of type-t negations in the first vs. second half of trace T:

$$F^t = \frac{|\text{Negs}_{\text{first half}}^t|}{|\text{Negs}_{\text{second half}}^t|}$$

3. Sequential Structure and Dependency This category captures structural patterns of negation across reasoning steps, including gap statistics (maximum span and variance of inter-negation distances), sequence patterns (number of type switches, longest consecutive runs), randomness tests, and autocorrelation. These metrics reveal whether negations are scattered or follow structured, self-regulatory sequences.

- **Max Neg Span:** Max number of steps between *gt*-type negations, $gt \in \{\text{cor}, \text{ref}, \text{any}\}$:

$$G_{\max}^{gt} = \max(\text{Gaps}^{gt})$$

- **Neg Gap Variance:** Variance of gaps between consecutive gt-type negations ($N^{gt} = |\text{Gaps}^{gt}|$):

$$\text{Var}[\text{Gaps}^{gt}] = \frac{1}{N^{gt}} \sum_{i=1}^{N^{gt}} \left(\text{Gap}_i^{gt} - \overline{\text{Gap}^{gt}}\right)^2$$

- **Neg Patterns:** Number of switches between corrective and refining negations, and longest consecutive sequence of each negation type.

$$S = |\text{C}_{\text{cor}\to\text{ref}}| + |\text{C}_{\text{ref}\to\text{cor}}|$$
$$L^t = \max(\hat{C}^t)$$

$\text{C}_{\text{cor}\to\text{ref}}$ and $\text{C}_{\text{ref}\to\text{cor}}$ are the sets of the corresponding type switches; $\hat{C}^t$ is the set of consecutive sequences of negations of type t.

- **Wald-Wolfowitz Test:** Tests randomness of the binary sequence representing the trace ($0 =$ no negation, $1 =$ negation of type t).

$$\mathbb{E}[R^t] = \frac{2n_1^t n_2^t}{n_1^t + n_2^t} + 1$$

$$\text{Var}[R^t] = \frac{2n_1^t n_2^t (2n_1^t n_2^t - n_1^t - n_2^t)}{(n_1^t + n_2^t)^2 (n_1^t + n_2^t - 1)}$$

n_1^t (n_2^t): number of steps with(out) negation of type t. The standardized test statistic is:

$$Z^t = \frac{R^t - \mathbb{E}[R^t]}{\sqrt{\text{Var}[R^t]}}$$

- **Autocor:** Autocorrelation for different lags ($k \in [1, .., 8]$) for negations of type t:

$$\rho_k^t = \frac{\sum_{i=1}^{N-k} (Y_i^t - \overline{Y^t})(Y_{i+k}^t - \overline{Y^t})}{\sum_{i=1}^{N} (Y_i^t - \overline{Y^t})^2}$$

- **Integrated Autocor:** Measures persistence of negations of type t across multiple steps:

$$\tau^t = 1 + 2 \sum_{k=1}^{\text{max\_lag}} \rho_k^t$$

3.2.2 Metric Complementarity

To assess redundancy in the proposed metric set, we compute pairwise correlations between all negation-based metrics. Figure 4 shows that most metrics are weakly correlated or uncorrelated, indicating that they capture distinct aspects of negation behavior. The main exceptions are two larger

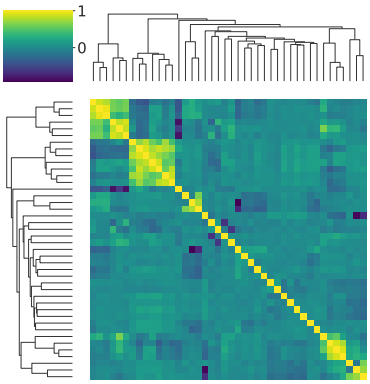


Figure 4: Hierarchically clustered heatmap of correlations between negation-quantifying metrics.

clusters corresponding primarily to autocorrelation features for refining and corrective negation across different lag sizes, where correlation is expected. Overall, the heatmap supports retaining the full metric set, as redundancy is limited and mostly confined to closely related temporal-dependency metrics.

4 Experiments

We investigate whether discourse-level negation patterns in reasoning traces provide information about (i) response correctness, (ii) model identity, and (iii) authorship, i.e. whether a trace was produced by a human or an LLM. Our experiments are organized accordingly: we first study correctness prediction, beginning with single-metric analyses and then moving to multi-feature classification, including an additional evaluation on perturbed human reasoning traces. We then examine model identity prediction and, finally, human-vs.-LLM authorship classification. Unless otherwise noted, all classification experiments use a Gradient Boosting (GB) LightGBM model with 1000 estimators, learning rate 0.01, maximum depth 3, minimum of 2 samples per leaf, subsample ratio 0.8, column subsampling of 0.8, and balanced class weights. All models are evaluated with five-fold cross-validation (CV), and we report fold-averaged results.[4] For each classification task, we compare four feature settings: (1) our negation-based feature set, (2) a structural baseline consisting of reasoning-trace length, total number of reasoning steps, and number of sentences, (3) an embedding baseline based on sentence-transformer representations from `all-MiniLM-L6-v2` (Reimers and Gurevych, 2019), and (4) a combined representation that concatenates the negation features with the embedding features.

Correctness prediction. We begin with single-metric analyses to assess how individual negation-related metrics relate to response correctness. To capture both linear and non-linear effects, we report correlation, mutual information (MI), area under the ROC curve (AUC), and Cohen's d effect size for individual features. We then turn to multi-feature classification, where the goal is to predict whether a reasoning trace leads to a correct or incorrect final answer. These experiments are conducted across our benchmark datasets and model outputs using the four feature settings described above. For human reasoning traces, we additionally evaluate correctness prediction across the three perturbation levels introduced in Section 3.1. Incorrect traces are generated by perturbing correct human traces with increasing severity. This setup allows us to examine how classification performance changes as reasoning errors become more pronounced and correct and incorrect traces become more easily separable. Each perturbation-level dataset is balanced, containing equal numbers of correct and incorrect traces.

Model identity prediction. We next test whether reasoning traces contain model-specific stylistic signal by predicting which LLM produced a given trace. We use the same GB configuration and the same four feature settings as above in order to compare the discriminative value of negation patterns against structural and embedding-based baselines.

Human-vs.-LLM authorship classification. Finally, we evaluate whether reasoning traces authored by humans can be distinguished from those generated by LLMs. To this end, we construct a balanced dataset by pairing all human traces from the *nemo* dataset with an equal number of randomly selected correct LLM-generated traces drawn from the nemo dataset as well. We again compare negation features, the structural baseline, the embedding baseline, and the combined negation+embedding representation under the same evaluation setup.

[4] Experiments primarily use scikit-learn (Pedregosa et al., 2011) and LightGBM (Lamb et al., 2025).

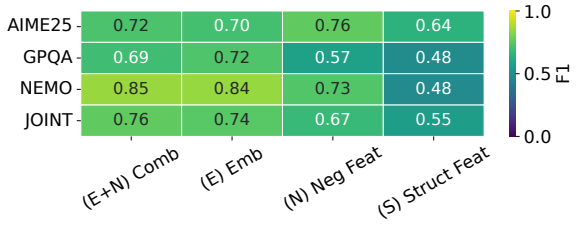


Figure 5: Five-fold CV binary F1 scores for correctness prediction across all datasets and the merged dataset across all different feature settings.

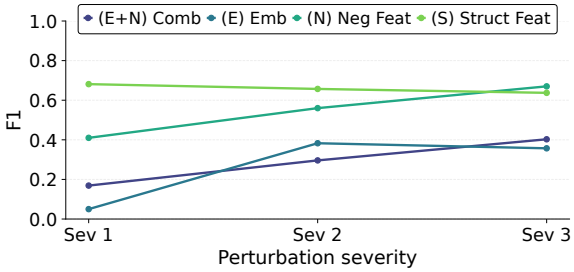


Figure 6: Five-fold CV binary F1 scores for classifier performance on human reasoning traces under varying levels of human-like perturbation severity.

5 Results and Analysis

5.1 Correctness Prediction

5.1.1 Single-Feature Analysis

Across all single-feature analyses, individual metrics exhibit only limited predictive power for distinguishing correct from incorrect reasoning traces. Absolute correlations remain low (max. ≈ 0.1), and mutual information scores peak at roughly 0.02. Cohen's d shows somewhat larger separations between the distributions, with several metrics reaching values around 0.2, corresponding to small to moderate effect sizes. Similarly, AUC values exceed the random baseline only marginally for a small subset of features. A consistent pattern emerges, however: metrics associated with refining negations dominate among the strongest-performing features across all measures (24 refining vs. 2 corrective). This suggests that refining-related behavior is more systematically linked to model correctness than corrective negation, reinforcing our observations from the preliminary analysis (Appendix C).

5.1.2 Multi-Feature Correctness Prediction

We next evaluate multi-feature correctness prediction with Gradient Boosting, reporting five-fold CV F1 scores for the four feature settings introduced in Section 4. Across the benchmark datasets and their joint aggregation, negation features consistently outperform the structural baseline, indicating that discourse-level negation captures signal beyond simple trace statistics such as length, number of steps, and sentence count. Averaged across datasets, this improvement amounts to 27.85%, with a large standardized effect size ($d = 2.03$). At the same time, embedding-based representations remain stronger overall: on *nemo*, *gpqa*, and the joint dataset, embedding features outperform negation features, whereas on *aime25*, negation features achieve a higher F1 score by 0.06. Overall, moving from negation features to embeddings yields an average relative improvement of 10.98% ($d = -0.74$), suggesting that embeddings provide a stronger standalone representation for correctness prediction. However, combining negation features with embeddings yields small additional gains in several settings, suggesting that negation captures some complementary information beyond generic embedding-based representations (Figure 5). Performance also varies across model families and datasets. No single model dominates in every condition: *magistral:24b* performs best on *aime25* (mean F1 0.643), *deepseek-r1:70b* on *gpqa* (0.753), *qwen3:32b* on *nemo* (0.900), and *gpt-oss:120b* on the joint dataset (0.737). Across all conditions, *gpt-oss:120b* achieves the highest average F1 score (0.719). Within both the *gpt-oss* and *deepseek-r1* families, larger models generally outperform smaller ones, although this trend is not uniform across all datasets (see Figure 11).

5.1.3 Human-Trace Perturbation Study

On the human reasoning traces from the *nemo* dataset, we conduct a controlled perturbation study in which incorrect traces are constructed by introducing human-like reasoning errors into originally correct human CoTs. We then repeat the multi-feature correctness experiment using these perturbed traces as negative samples and report fold-averaged binary F1 scores across the three perturbation severity levels (see Section 3.1). Because these negative samples are synthesized rather than naturally occurring, we interpret this setting as a controlled stress test of feature sensitivity to injected reasoning errors rather than as a direct model of authentic human failure. Across all three severity levels, the structural baseline performs unexpectedly strongly, achieving fold-averaged F1 scores above 0.6 throughout. At the same time,

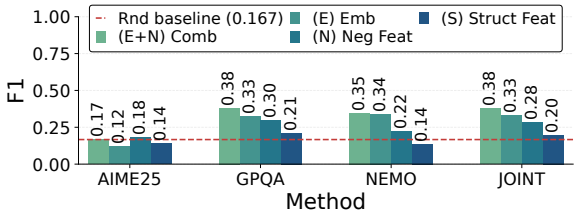


Figure 7: Five-fold CV macro F1 scores for model detection from reasoning traces grouped by datasets and feature settings.

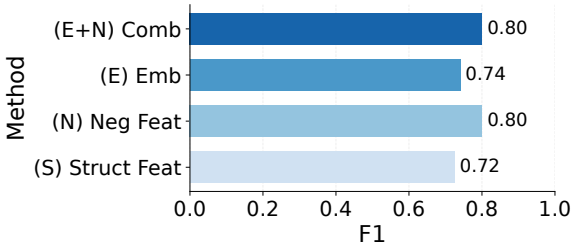


Figure 8: Five-fold CV binary F1 scores for human-vs.-LLM authorship classification with different feature sets.

its performance decreases as perturbation severity increases, whereas the negation-based representation becomes more discriminative as the injected reasoning errors grow more pronounced, reaching its best performance at the highest severity level. Embedding-based features perform substantially worse than both the structural and negation-based representations in this setting. One plausible explanation is that the perturbations preserve much of the lexical content of the original trace while altering its explicit discourse structure, making negation- and structure-based features relatively more informative than generic sentence embeddings. Overall, this controlled result suggests that negation-based features are sensitive to injected reasoning distortions in human-authored traces, although the finding should not be generalized directly to naturally occurring incorrect human reasoning.

5.2 Model Identity Prediction

Next, we test whether reasoning traces contain identifiable model-specific stylistic signatures. Concretely, we ask whether the originating LLM can be predicted from the four feature settings defined in Section 4. Because this is a multiclass classification task, we report five-fold CV macro F1 scores. Overall, model identity prediction proves challenging: with six models, the random baseline is 0.166 macro F1, and even the best results remain modest in absolute terms. Nevertheless, negation features consistently outperform the structural baseline across all four datasets (*aime25*, *nemo*, *gpqa*, and the joint dataset), indicating that they capture stylistic signal beyond simple trace statistics. Embedding-based features remain stronger overall, but combining embeddings with negation features yields further improvements in every setting, reaching a peak macro F1 of 0.38 on the joint dataset. Taken together, these results suggest that reasoning traces contain a limited but non-negligible amount of model-specific signal, and that negation contributes useful complementary information for capturing it (Figure 7).

Feature	Group	Human	LLM	SMD	q_{group}
`cor_neg_longest_run`	Struct.	1.111	0.306	0.561	$< .001$
`cor_negs_normed`	Comp.	0.125	0.051	0.623	.017
`ref_negs_normed`	Comp.	0.031	0.079	-0.476	.021

Table 1: Group-wise FDR-significant differences between human and LLM reasoning traces in the authorship classification dataset. Positive SMD values indicate higher values for human traces; negative values indicate higher values for LLM traces. Only features satisfying $q_{\text{group}} \leq .05$ and $|\text{SMD}| \geq .2$ are shown.

5.3 Human-vs.-LLM Authorship Classification

Finally, we evaluate whether human-authored reasoning traces can be distinguished from LLM-generated ones. As in the other binary classification settings, we report five-fold CV binary F1 scores. Unlike model identity prediction, this task exhibits a much stronger signal across all feature settings. Negation features perform best overall, reaching an average F1 score of 0.80 and outperforming both the embedding baseline (0.74) and the structural baseline (0.72). Combining negation features with embeddings does not yield further improvement, with the combined representation remaining at the same maximum F1 score of 0.80. These results suggest that human-vs.-LLM authorship is substantially easier to detect than model identity in reasoning traces, and that discourse-level negation features are particularly effective for capturing differences between human and model-generated reasoning traces (Figure 8).

5.4 Interpretability

To better understand which aspects of negation drive performance across tasks, we compute SHAP values for correctness prediction on the joint dataset, model identity prediction, and human-vs.-

LLM authorship classification. Detailed SHAP plots are provided in Appendix E. For interpretability, we aggregate features into broader classes and summarize their importance using mean absolute SHAP values normalized to sum to 1 within each task. A first clear pattern is that the relative importance of *refining* and *corrective* negation differs by task. For correctness prediction, refining features contribute more strongly than corrective ones (64% vs. 36%), consistent with both our preliminary analyses and the single-feature results. This suggests that metrics quantifying refining negation provide more information for correctness prediction than metrics quantifying overt corrective rejection of prior steps. Importantly, this does not imply that correct traces necessarily contain more refining negation, nor that refining negation is causally more effective. Rather, it indicates that variation in refining-related features is more useful for distinguishing correct from incorrect model reasoning traces in our classification setting. By contrast, contributions are more balanced for model identity (43% vs. 57%) and human-vs.-LLM authorship classification (58% vs. 42%), indicating that provenance-related tasks depend on a broader mix of negation behavior. A second pattern concerns feature type. Across all three tasks, structural features are the most consistently informative dimension (average 41%), ahead of positional (29%) and compositional features (30%). This suggests that *how* negation is organized across a reasoning trace is at least as important as *how much* negation occurs. Overall, the SHAP analysis supports the usefulness of the corrective/refining distinction and indicates that different downstream tasks rely on different aspects of negation behavior. To complement this model-based interpretation with a direct descriptive comparison, we compare human and LLM traces in the authorship setting across all negation features. For numeric features, we compute group means, standardized mean differences (SMD), and Mann-Whitney U tests; for categorical features, we use contingency-table statistics. Since the metrics were defined a priori as compositional, positional, and structural feature families, we apply Benjamini-Hochberg FDR correction within each family and report features as significant only if they satisfy $q \leq 0.05$ and a small effect-size threshold ($|\mathrm{SMD}| \geq 0.2$ for numeric features). Only three features pass these criteria (Table 1): humans show more corrective negation per reasoning step and longer repeated runs of corrective negations, whereas LLMs show more refining negation per reasoning step. No positional feature passes the group-wise FDR criterion, suggesting that the strongest human-LLM differences concern the function and sequential organization of negation rather than its location in the trace. These descriptive results clarify the SHAP analysis: the authorship signal is concentrated in a small set of interpretable negation behaviors, with human traces characterized more by corrective revision and LLM traces more by refining restriction.

6 Conclusion

We introduced a discourse-level approach to analyzing reasoning traces through negation. Specifically, we distinguished between *refining* and *corrective* negation, assembled a dataset of human- and LLM-authored reasoning traces annotated for these two functions, and proposed metrics that quantify negation in terms of composition, position, and sequential structure. Using these representations, we tested whether negation patterns carry signal for correctness, model identity, and human-vs.-LLM authorship. Across correctness experiments on LLM-generated traces, negation features consistently outperformed simple structural baselines, although embedding-based representations remained stronger overall and negation provided mainly complementary gains in combined settings. In a controlled perturbation study on human traces, negation features proved especially sensitive to injected reasoning distortions, though this result should be interpreted separately from naturally occurring human error. For model identity prediction, performance remained modest overall, but negation still contributed useful signal beyond structural cues. Our strongest results were obtained for human-vs.-LLM authorship classification on correct traces from the same benchmark, where negation features outperformed both structural and embedding baselines. Overall, these findings suggest that discourse-level negation is an interpretable and informative feature for reasoning-trace analysis, especially for provenance-related distinctions and, to a lesser extent, correctness-related variation. More broadly, the results motivate further work on reasoning traces through explicit linguistic phenomena, including revision, uncertainty, and inter-sentential propositional relations.

Limitations

This work is subject to several scope-related limitations:
(1) First, we focus on explicit instances of negation, distinguishing between refining and corrective uses within reasoning traces. While this provides a clear and interpretable operationalization, it does not capture more implicit or indirect forms of negation, such as pragmatic denial or contrastive framing, which may also influence reasoning.
(2) Second, our analysis is centered on CoT–style reasoning traces, which offer an explicit, step-wise view and naturally encode structured dependencies across steps. Other reasoning frameworks or prompting strategies may exhibit different surface realizations of negation, and extending the analysis to such settings remains a promising direction for future work.
(3) Finally, this study covers only a subset of task types that elicit explicit reasoning (restricted to textual, single-modality tasks). Although the datasets include both open-ended and multiple-choice problems, we hypothesize that negation usage patterns vary with task characteristics.
Overall, these considerations reflect design choices aimed at analytical clarity and point to natural extensions rather than fundamental shortcomings of the proposed approach.

Acknowledgements

This work was supported by the German Research Foundation (DFG) as part of Project INF within CRC 1629 "Negation in Language and Beyond" (NegLaB - https://www.neglab.de/).

References

Sara Abdali, Can Goksen, Michael Solodko, Saeed Amizadeh, Julie E. Maybee, and Kazuhito Koishida. 2025. Self-reflecting large language models: A hegelian dialectical approach. *Preprint*, arXiv:2501.14917.

Denis Apothéloz, Pierre-Yves Brandt, and Gustavo Quiroz. 1993. The function of negation in argumentation. *Journal of Pragmatics*, 19(1):23–38.

Hugh H Benson. 2011. Socratic method. *The Cambridge companion to socrates*, pages 179–200.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.

Siddhant Bhambri, Upasana Biswas, and Subbarao Kambhampati. 2025. Do cognitively interpretable reasoning traces improve llm performance? *Preprint*, arXiv:2508.16695.

Paul C. Bogdan, Uzay Macar, Neel Nanda, and Arthur Conmy. 2025. Thought anchors: Which llm reasoning steps matter? *Preprint*, arXiv:2506.19143.

Daniel Büring. 2008. The least at least can do. In *West Coast Conference on Formal Linguistics (WCCFL)*, volume 26, pages 114–120.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Wei Du, Branislav Kisacanin, George Armstrong, Shubham Toshniwal, Ivan Moshkov, Alexan Ayrapetyan, Sadegh Mahdavi, Dan Zhao, Shizhe Diao, Dragan Masulovic, Marius Stanean, Advaith Avadhanam, Max Wang, Ashmit Dutta, Shitij Govil, Sri Yanamandara, Mihir Tandon, Sriram Ananthakrishnan, Vedant Rathi, and 6 others. 2025. The challenge of teaching reasoning to llms without rl or distillation. *Preprint*, arXiv:2507.09850. Accepted at the 2nd AI for Math Workshop at ICML 2025.

Michael N. Forster. 1993. Hegel's dialectical method.

Anamaria Fălăuş. 2020. Negation and alternatives: Interaction with focus constituents. In *The Oxford Handbook of Negation*. Oxford University Press.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D. Goodman. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *Preprint*, arXiv:2503.01307.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Leon Lukas Hammerla, Andy Lücking, Carolin Reinert, and Alexander Mehler. 2025. D-neg: Syntax-aware graph reasoning for negation detection. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 1432–1454, Mumbai, India. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics.

Laurence R. Horn. 1989. *A Natural History of Negation*. University of Chicago Press.

Priyanka Kargupta, Shuyue Stella Li, Haocheng Wang, Jinu Lee, Shan Chen, Orevaoghene Ahia, Dean Light, Thomas L. Griffiths, Max Kleiman-Weiner, Jiawei Han, Asli Celikyilmaz, and Yulia Tsvetkov. 2025. Cognitive foundations for reasoning and their manifestation in llms. *Preprint*, arXiv:2511.16660.

Aditya Khandelwal and Suraj Sawant. 2020. NegBERT: A transfer learning approach for negation detection and scope resolution. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5739–5748, Marseille, France. European Language Resources Association.

Manfred Krifka. 2003. The semantics and pragmatics of polarity items.

James Lamb, Nikita Titov, Guolin Ke, wxchan, Laurae, shiyu1994, Qiwei Ye, José Morales, OMOTO Tsukasa, david-cortes, Belinda Trotta, Zhuyi Xue, Oliver Borchert, Ilya Matiach, xuehui, Alberto Ferreira, Nick Miller, Huan Zhang, Chen Yufei, and 11 others. 2025. microsoft/LightGBM. https://github.com/microsoft/LightGBM.

Qintong Li, Leyang Cui, Lingpeng Kong, and Wei Bi. 2025. Exploring the reliability of large language models as customized evaluators for diverse NLP tasks. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 10325–10344, Abu Dhabi, UAE. Association for Computational Linguistics.

Sara Vera Marjanovic, Arkil Patel, Vaibhav Adlakha, Milad Aghajohari, Parishad BehnamGhader, Mehar Bhatia, Aditi Khandelwal, Austin Kraft, Benno Krojer, Xing Han Lù, Nicholas Meade, Dongchan Shin, Amirhossein Kazemnejad, Gaurav Kamath, Marius Mosbach, Karolina Stanczak, and Siva Reddy. 2026. Deepseek-r1 thoughtology: Let's think about LLM reasoning. *Transactions on Machine Learning Research*.

Nailia Mirzakhmedova, Marcel Gohsen, Chia Hao Chang, and Benno Stein. 2024. Are large language models reliable argument quality annotators? In *Robust Argumentation Machines*, pages 129–146, Cham. Springer Nature Switzerland.

OpenAI, :, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, and 108 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jiangbo Pei, Peiyu Liu, Xin Zhao, Aidong Men, and Yang Liu. 2025. Socratic style chain-of-thoughts help LLMs to be a better reasoner. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 12384–12395, Vienna, Austria. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof Q&A benchmark. In *Proceedings of the First Conference on Language Modeling (COLM)*.

Claudia Schon, Sophie Siebert, and Frieder Stolzenburg. 2021. Negation in cognitive reasoning. In *KI 2021: Advances in Artificial Intelligence*, pages 217–232, Cham. Springer International Publishing.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

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

Fangzhi Xu, Qika Lin, Jiawei Han, Tianzhe Zhao, Jun Liu, and Erik Cambria. 2025. Are large language models really good logical reasoners? a comprehensive evaluation and beyond. *IEEE Transactions on Knowledge and Data Engineering*, 37(4):1620–1634.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822. Curran Associates, Inc.

Mengyu Ye, Tatsuki Kuribayashi, Jun Suzuki, Goro Kobayashi, and Hiroaki Funayama. 2023. Assessing step-by-step reasoning against lexical negation: A case study on syllogism. In *Proceedings of the 2023*

Conference on Empirical Methods in Natural Language Processing, pages 14753–14773, Singapore. Association for Computational Linguistics.

Anqi Zhang, Yulin Chen, Jane Pan, Chen Zhao, Aurojit Panda, Jinyang Li, and He He. 2025a. Reasoning models know when they're right: Probing hidden states for self-verification. In *Proceedings of the Second Conference on Language Modeling (COLM)*.

Mike Zhang, Johannes Bjerva, and Russa Biswas. 2025b. Follow the path: Reasoning over knowledge graph paths to improve llm factuality. *Preprint*, arXiv:2505.11140.

Yifan Zhang and Team Math-AI. 2025. American invitational mathematics examination (aime) 2025.

A Human Evaluation

Across tasks, inter-human agreement is highest on gpqa ($\kappa = 0.77$), followed by aime25 ($\kappa = 0.63$), and lowest on nemo ($\kappa = 0.50$). Human-LLM agreement shows a related but not identical pattern. On aime25, agreement with the LLM ranges from $\kappa = 0.36$ to 0.45; on gpqa, it ranges from $\kappa = 0.35$ to 0.46; and on nemo, both annotators yield $\kappa = 0.25$. Thus, although gpqa has the highest inter-human agreement, its human-LLM agreement is not uniformly higher than that of aime25. By contrast, nemo yields the lowest human-LLM agreement for both annotators (Table 2). A similar pattern appears across models. qwen3:32b shows the strongest human-LLM alignment ($\kappa = 0.65$ for both annotators) and also perfect inter-human agreement ($\kappa = 1.00$). deepseek-r1:32b also exhibits relatively strong human-LLM agreement ($\kappa = 0.53$ for both annotators), paired with high human-human agreement ($\kappa = 0.848$). In contrast, gpt-oss:120b and gpt-oss:20b show comparatively weak human-LLM agreement despite moderate to high inter-human agreement on the same samples (Table 3).

Agreement Metric	aime25	gpqa	nemo
H1-LLM (κ)	0.45	0.35	0.25
H2-LLM (κ)	0.36	0.46	0.25
H-H (κ)	0.63	0.77	0.50

Table 2: Agreement (Cohen's κ) across reasoning traces on different benchmarks.

Model	H1-LLM (κ)	H2-LLM (κ)	H-H (κ)
gpt-oss:120b	0.07	0.28	0.88
deepseek-r1:70b	0.41	0.37	0.54
deepseek-r1:32b	0.53	0.53	0.85
magistral:24b	0.24	0.27	0.31
gpt-oss:20b	0.12	0.09	0.69
qwen3:32b	0.65	0.65	1.00

Table 3: Agreement (Cohen's κ) across reasoning traces from different LLMs.

B Preliminary Analysis (Refining vs. Corrective Negation Frequencies)

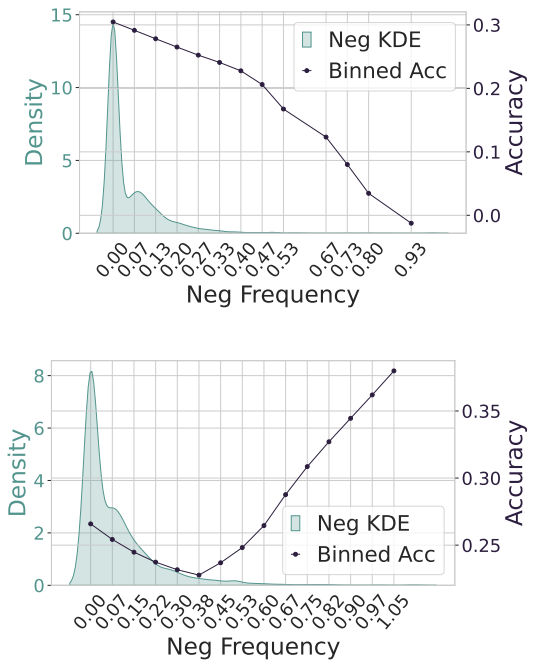
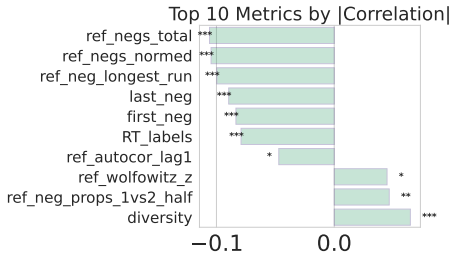


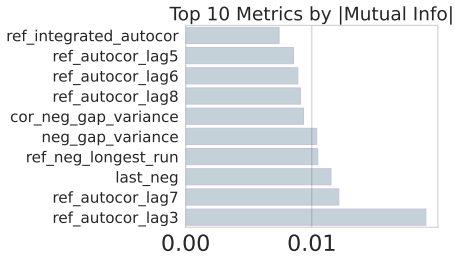
Figure 9: Aggregated results by datasets and models: binned accuracies and kernel density estimates of negation frequencies (negations per reasoning step), smoothed using LOWESS. Corrective negation is shown on the top, and refining negation on the bottom.

To investigate the contrasting behavior of refining and corrective negations, we bin all reasoning traces across datasets according to the frequency of each negation type and compute the mean model accuracy within each bin. We then apply LOWESS smoothing to the resulting trends to obtain robust trajectories. The patterns reveal a clear divergence between the two types: accuracy initially declines for both, but the drop is substantially steeper for corrective negations. At higher frequencies, refining negations exhibit a recovery and eventually show a positive association with model accuracy, whereas corrective negations continue to decline monotonically. This indicates that increased use of refining negation correlates with improved model performance, while corrective negation does not. These effects are further supported by single-metric analyses reported in Appendix C. The full smoothed curves are shown in Figure 9.

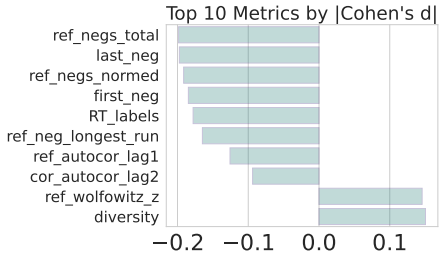
C Single-metrics (negative) results



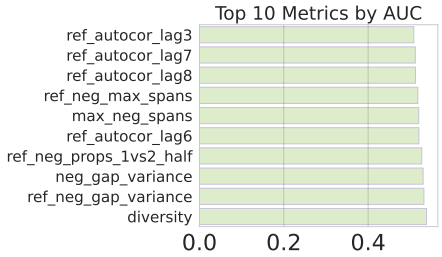
(a) Pairwise Spearman correlations between correctness (binary) and the proposed metrics. Significance levels are marked with asterisks: *** $p < 0.001$, ** $p < 0.01$, * $p < 0.05$.



(b) Mutual information between correctness (binary) and the proposed metrics, with continuous metrics discretized by binning.



(c) Distributional differences of the proposed metrics between correct and incorrect samples, quantified by Cohen's d.



(d) Discriminative ability of metrics between correct and incorrect samples, measured by AUC.

Figure 10: Comparison of different statistical measures assessing the relationship between the proposed metrics and binary correctness across all datasets. Each subplot reports the top 10 metrics ranked by the respective measure.

D Multi-metrics Result

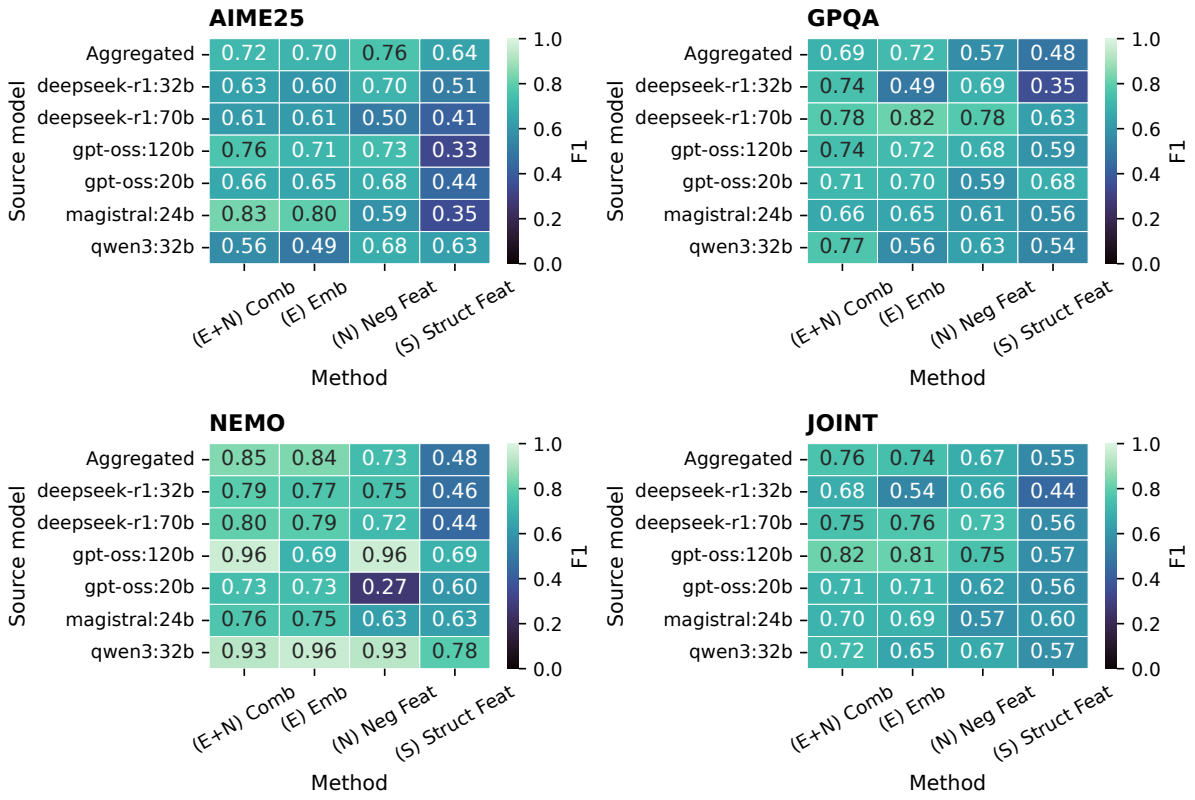


Figure 11: Five-fold CV binary F1 scores for correctness prediction on each dataset and the merged dataset, grouped by source model and feature setting.

E SHAP Analysis

E.1 Predicting Model Correctness (SHAP)

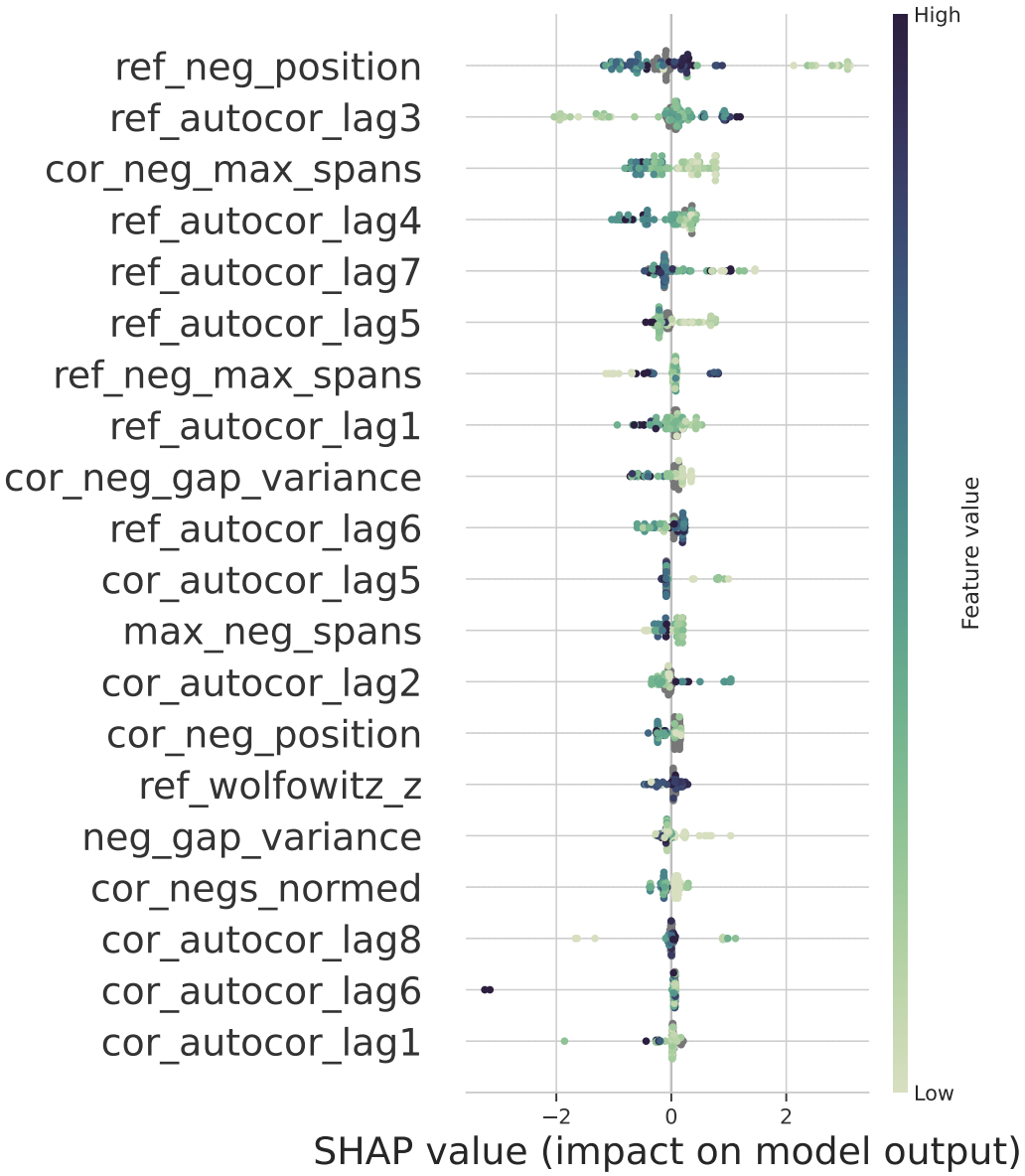


Figure 12: SHAP values showing the impact of negation metrics on predicting response correctness from reasoning traces using gradient boosting. Predictions were made using aggregated data from all three datasets.

E.2 Predicting Identity (LLM)

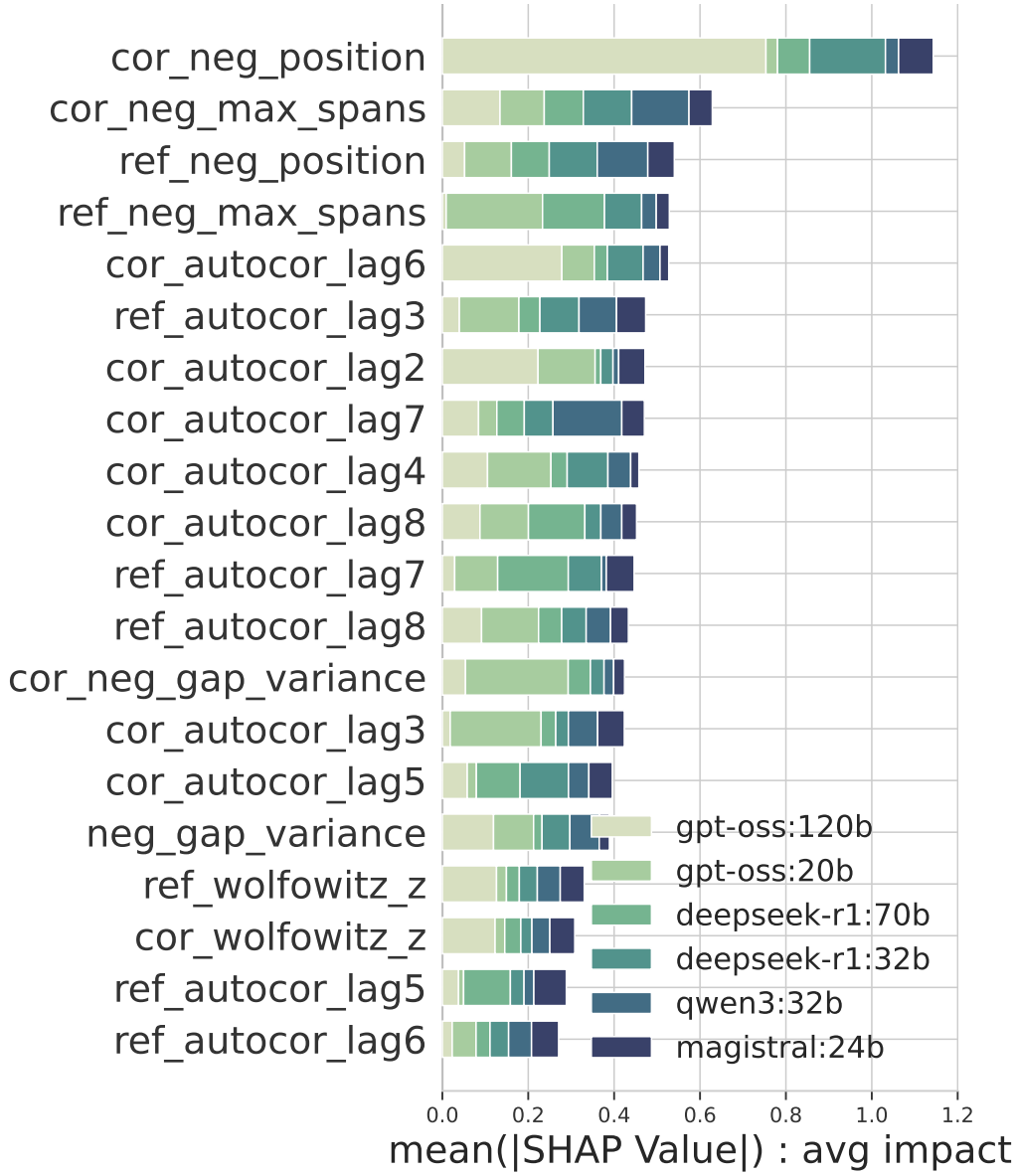


Figure 13: Mean SHAP values showing the contribution of each feature to the model output when predicting which model produced a reasoning trace, based on its negation quantification (top 20 features displayed).

E.3 Predicting Identity (Human-LLM)

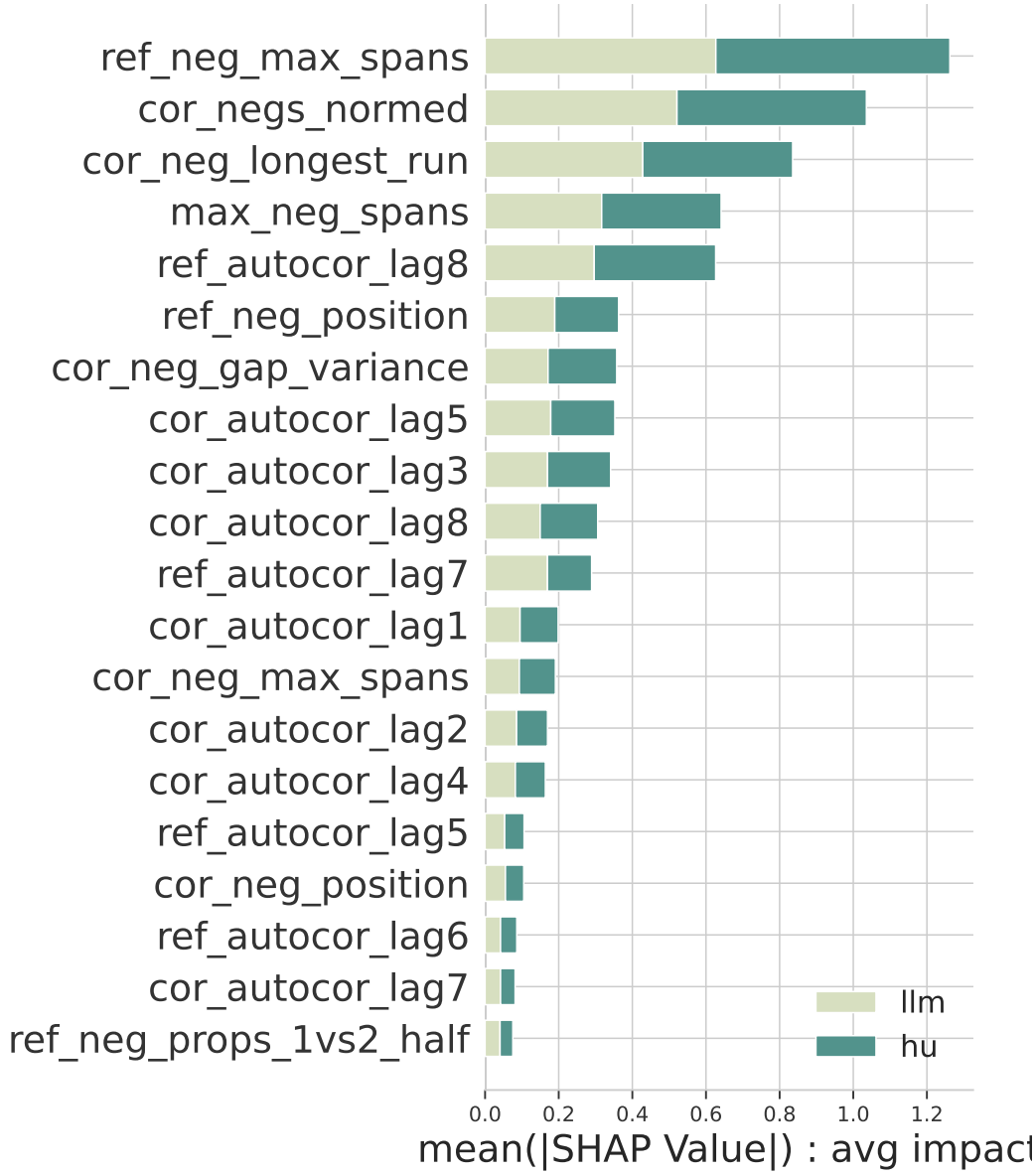


Figure 14: Mean SHAP values showing the contribution of each negation feature to the model output when classifying human and LLM reasoning traces, based on their negation quantification (top 20 features displayed).

F Prompts

F.1 Dataset - Model Benchmark (Step 1)

```
You are given a multiple-choice question with four options. Only one option is correct.

1. Provide a brief explanation for your choice.
2. Additionally, add your answer in JSON format:
   {"answer":ANSWER_ID}
   where ANSWER_ID is the number (0, 1, 2, or 3) corresponding to the correct option.

---

Question:
[question]

---

Options:
0: [option1]
1: [option2]
2: [option3]
3: [option4]
```

Figure 15: Prompt for creating reasoning traces and model responses for the gpqa benchmark.

```
You are tasked with solving a math question where the answer is guaranteed to be an integer.

1. Provide a concise explanation of your solution.
2. Verify that your answer is an integer, as required by the question.
3. Additionally, add your answer in JSON format: {"answer": RESULT}, where RESULT is an integer solution to the provided
↪  question.

---

Question:
[question]
```

Figure 16: Prompt for creating reasoning traces and model responses for the aime25 and nemo benchmark.

F.2 Dataset - Human Error (Step 2)

```
You are given a **correct human reasoning trace**.
Your task is to generate a **perturbed version of that reasoning trace** that:
* stays *close* to the original (similar structure, similar steps, similar flow)
* includes **human-like reasoning slips** (minor misreads, small leaps, overlooked details, mis-prioritizations)
* modifies the trace only slightly --- **do not rewrite it from scratch**
* introduces perturbations according to a severity scale:
  * **1 = very light distortions** (tiny misinterpretations, small skipped details)
  * **2 = moderate distortions** (noticeable but still realistic human mistakes)
  * **3 = strong distortions** (multiple compounded human-like errors, while still keeping the trace coherent)
**Important:**
* The output should **only be the perturbed reasoning trace itself**, do not include any kind of observation or explanation.
* **Do not** produce a final answer or solution --- the task is only to perturb the reasoning trace, not to solve the problem.
* The perturbed trace must remain recognizably similar to the original.
* The perturbation must be based *directly* on the original trace, not generated fresh.
```

Figure 17: Prompt for introducing human reasoning errors.

F.3 Dataset - Annotation (Step 3)

````
**Instruction:**
You are an annotator for reasoning chains generated by language models. Your task is to:
---
#### **1. Chunk the reasoning text**
* Break the reasoning text into **discrete reasoning steps**.
* Each step can contain **one or more sentences** representing a coherent thought, inference, or claim.
* Number the steps sequentially starting from 1.
---
#### **2. Detect corrective or refining negations**
Annotate only negations that:
* **CORRECTIVE_NEGATION**: Directly contradict or correct one or more earlier steps.
* **REFINING_NEGATION**: Narrow, qualify, or improve prior reasoning without fully contradicting it.
**Important:**
* Negations may appear **anywhere within a step**, even if the step contains multiple sentences.
* Only annotate negations that **modify previous reasoning to improve accuracy, precision, or clarity**.
* Ignore general negations that are purely factual, descriptive, or do not impact prior reasoning.
---
#### **3. Annotation requirements**
For each detected negation, provide:
* `type`: `CORRECTIVE_NEGATION` or `REFINING_NEGATION`
* `step`: Step number where the negation occurs
* `text`: Exact negated phrase in that step
* `refines_steps` or `corrects_steps`: List of prior step numbers affected
* If a single step contains **multiple corrective/refining negations**, list each separately.
* A negation can refer to **one or more previous steps**, and this should be reflected in the `refines_steps` or `corrects_steps` lists.
---
### **Output format (JSON-like)**
```json
{
  "steps": [
    "Step 1 text (can be multiple sentences)",
    "Step 2 text",
    ...
  ],
  "negations": [
    {
      "type": "CORRECTIVE_NEGATION",
      "step": 2,
      "text": "",
      "corrects_steps": [1]
    },
    {
      "type": "REFINING_NEGATION",
      "step": 3,
      "text": "",
      "refines_steps": [1]
    }
  ]
}
```
---
### **Example 1: Object Color (Extended)**
**Reasoning chain:**
*"The object appears red from one angle. Under direct light, it seems vibrant and uniform. But this may not be accurate because shadows and reflections can distort perception. Furthermore, it is not necessarily completely red; some areas show orange hints. Considering these observations, the object might actually be a mix of red and orange. Therefore, it is safer to describe the object as mostly orange with red undertones."*
**Annotation:**
```json
{
  "steps": [
    "The object appears red from one angle. Under direct light, it seems vibrant and uniform.",
    "But this may not be accurate because shadows and reflections can distort perception.",
    "Furthermore, it is not necessarily completely red; some areas show orange hints.",
    "Considering these observations, the object might actually be a mix of red and orange.",
    "Therefore, it is safer to describe the object as mostly orange with red undertones."
  ],
  "negations": [
    {
      "type": "CORRECTIVE_NEGATION",
      "step": 2,
      "text": "this may not be accurate",
      "corrects_steps": [1]
    },
    {
      "type": "REFINING_NEGATION",
      "step": 3,
      "text": "not necessarily completely red",
      "refines_steps": [1]
    }
  ]
}
```
...
### **Example n: xxx (Extended)**
...
````

Figure 18: Prompt for reasoning trace segmentation and discourse-level negation type annotation.

Discovering New Theorems via LLMs with In-Context Proof Learning in Lean

Kazumi Kasaura[1,5,3], Naoto Onda[1,2,3], Yuta Oriike[4,3], Masaya Taniguchi[5,3], Akiyoshi Sannai[6,7,8,9,10,3], Sho Sonoda[5,4,3]

[1]OMRON SINIC X Corporation, [2]NexaScience, [3]AutoRes, [4]CyberAgent, [5]RIKEN AIP, [6]Kyoto University, [7]RIKEN AGIS, [8]Shiga University, [9]NII, [10]NISTEP

Correspondence: kazumi.kasaura@sinicx.com

Abstract

Large Language Models (LLMs) have demonstrated significant promise in formal theorem proving. In this study, we investigate the ability of LLMs to discover novel theorems and produce verified proofs. We propose a pipeline called *Conjecturing-Proving Loop* (CPL), which iteratively generates mathematical conjectures and attempts to prove them in Lean 4. A key feature of CPL is that each iteration conditions the LLM on previously generated theorems and their formal proofs, enabling parameter-free improvement of proof strategies via in-context learning. We provide both theoretical and experimental evidence that CPL increases the discovery rate of hard-to-prove theorems compared to frameworks that generate statements and proofs simultaneously. Moreover, our experiments show that reusing the LLM's own formally verified outputs as context consistently improves subsequent proof success, demonstrating the effectiveness of self-generated in-context learning for neural theorem proving. The source code is available at https://github.com/auto-res/ConjecturingProvingLoop.

1 Introduction

Large Language Models (LLMs) have demonstrated significant promise in theorem proving. Since LLMs can hallucinate and it is difficult to detect such hallucinations in natural language, generating formal proofs using an LLM and verifying them using an interactive theorem prover (ITP), such as Lean[1], has been studied. In this paper, we focus on the ability of LLMs to discover novel theorems.

We propose the *Conjecturing-Proving Loop*, a pipeline for automatically generating mathematical conjectures and proving them in Lean 4 format. By separating the conjecturing and proving phases, we avoid the generation of identical theorems and encourage proving more difficult theorems. In other words, CPL employs *stratified sampling* over conjecture/proof candidates to allocate search resources according to proof difficulty, preventing the loop from collapsing to easy, short proofs. This stratification allows CPL to discover and verify longer proofs that are harder to discover in a simpler framework that samples statements and proofs simultaneously. In this paper, we present a more detailed theoretical discussion of this point.

Another feature of our approach is that we generate and prove further theorems using context that includes proven theorems and their proofs, which enables the generation of more difficult proofs by in-context learning of proof strategies without training of LLMs. Since the ability of reasoning and generation of Lean code by closed-source LLMs, such as GPT, has been improved recently, we use them as both conjecturer and prover. While a disadvantage of using closed-source LLMs is that models cannot be trained freely, in our framework, the proving ability of LLMs can be improved by in-context learning from previous verified proofs.

In our experiment, when mathematical notions were given as seeds, we verified whether important properties about them could be rediscovered in our framework. More specifically, we focused on a few topological notions which are defined only by notions in Mathlib[2], Lean's mathematical library, but not included in Mathlib. We generated theorems about these notions using our framework. As a result, we found that our framework rediscovered an important theorem about these notions that had been published in mathematical papers, which was not found in the simpler framework without separating the Conjecturer and Prover. Moreover, we verified that the in-context learning of proof strate-

[1] https://lean-lang.org/

[2] https://github.com/leanprover-community/mathlib4

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 40–49

gies works within our framework: The important theorem, which cannot be proved without context by LLMs even in natural language, was proved with the generated context.

In summary, the contributions of this paper are as follows. First, we propose the Conjecturing-Proving Loop, a pipeline for automatically generating mathematical conjectures and proving them in Lean 4 format. Second, we demonstrated theoretically and experimentally that our framework enables automatic discovery of hard-to-prove theorems. Third, we verified that the proving ability of LLMs can be improved by in-context learning, when the verified proofs generated by LLMs themselves before the statement of a target theorem is provided are given as context.

Our work also suggests the potential for automatic expansion of formal mathematics libraries using AI. Formalized mathematics is only a part of mathematics expressed in natural language, and expanding formal libraries, such as Mathlib, is crucial for verifying and automating mathematics. On the other hand, the set of propositions that should be included in the library may not always be obtainable in natural language. Our framework can generate propositions about given notions while learning them.

2 Related Work

There are several works that use LLMs for mathematical reasoning, both in natural language (DeepSeek-AI et al., 2025) and formal language for ITP (Ren et al., 2025; Lin et al., 2025a). They focused mainly on solving existing problems and used Supervised Fine-Tuning (SFT) and/or Reinforcement Learning with Verified Rewards (RLVR) to improve problem-solving ability of LLMs. To overcome the limited dataset for training, approaches to generate problems to solve by AI are also proposed in some previous works (Dong and Ma, 2025; Huang et al., 2025; Zhan et al., 2025). These works are different from our approach in the following two points: First, our approach is focused on generating and proving meaningful theorems, while these works are focused on training the LLM prover. Second, while these works are based on reinforcement learning, our approach is based on in-context learning, which can be applied to closed-source LLMs.

Several studies have reported that including appropriate contexts in prompts improves the mathematical reasoning ability of LLMs (Wei et al., 2022; Zhou et al., 2022; Drori et al., 2022; Hu et al., 2024; Poiroux et al., 2025). While these studies use handmade examples or data extracted from a database as in-context learning sources, our framework uses outputs from the LLM itself, as in In-Context Reinforcement Learning (Moeini et al., 2025). It has also been proposed to conjecture and prove propositions to be used as lemmas for proving more difficult theorems (Thakur et al., 2023; Wang et al., 2023; Chen et al., 2025; Baba et al., 2025). Unlike these studies, we do not provide the theorem to prove to the LLM; instead, we focus on the LLM's ability to discover the theorem and the libraries used as context during proof generation are produced before the target theorem statement is given.

The technique to leverage feedback from ITP for formal proof generation was proposed (First et al., 2023; Thakur et al., 2023; Lin et al., 2025b) and we also adopted it (See § 3.3). However, what we emphasize here is learning strategies from verified proofs of other propositions.

Minimo (Poesia et al., 2024) shares a similar framework with ours: it jointly trains the conjecturer and prover agents to find theorems. However, while it aims to rediscover mathematics without using existing knowledge, our research has a more practical purpose: attempting to discover theorems by using existing large language models.

A survey (Zhang and Tan, 2026) provides a comprehensive and up-to-date summary of theorem generation, including methods that use LLMs.

3 Method

In this section, we first present an overview of the framework, and then explain the architectures of the conjecturer and the prover.

3.1 Pipeline Overview

Figure 1 illustrates our framework. Conjecturing-Proving Loop (CPL) consists of four main parts: conjecturer (LLM agent), prover (LLM agent), Lean server, and library (Lean code data). First, the library is initialized by the user.

1. The conjecturer generates novel mathematical conjectures in valid Lean 4 format based on the library, while accessing the Lean server.

2. For each generated conjecture, the prover tries to generate valid proofs, while accessing the

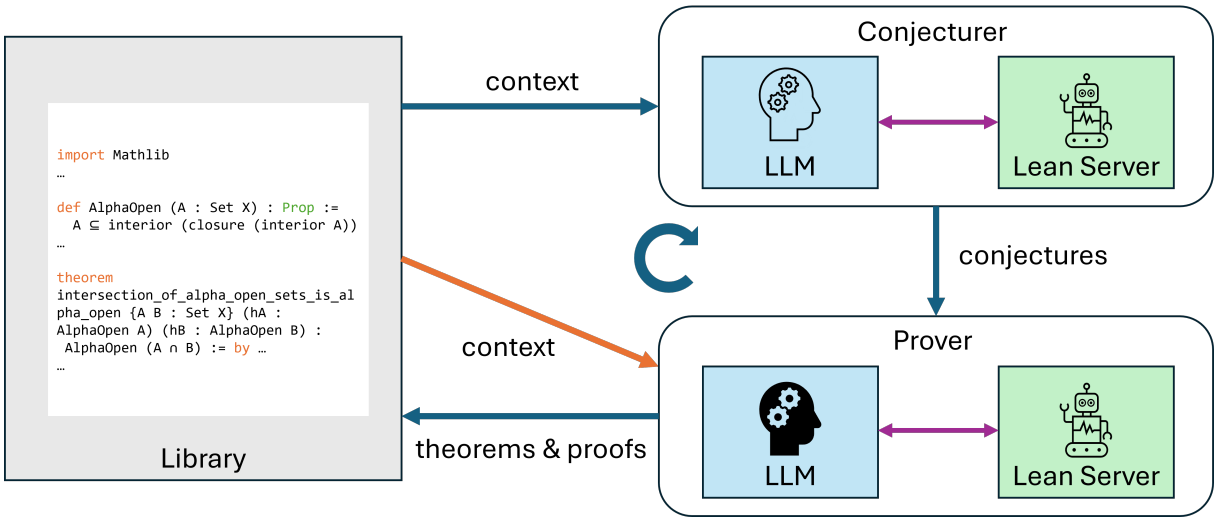


Figure 1: Overview of Conjecturing-Proving Loop. Conjecturer generates conjectures using the library as context, Prover attempts to prove them, and proven conjectures and their proofs are stored in the library as theorems. The library also provides context to Prover. Both Conjecturer and Prover processes consist of interactions between LLMs and Lean Server.

Lean server. The library is used as the context also in this step.

3. The verified pairs of conjectures and their proofs are added to the library. We return to the first step.

For the details of the conjecturing and proving steps, see the following subsections.

By separating the conjecturing and proving phases, we avoid the generation of identical theorems and encourage proving more difficult theorems. For a more detailed discussion, please refer to § 4.

The purpose of feeding the library to the conjecturer as the context is to prevent the generation of duplicate conjectures and to generate conjectures by analogy from already proven theorems.

The purpose of feeding the library to the prover as the context is to make already proven theorems available during proof and to learn proof strategies by in-context learning.

3.2 Conjecture Loop

To generate varied conjectures, for each conjecturing step, we use the following process.

I. The conjecturer LLM generates conjectures following the current library.

II. For each generated conjecture, the Lean server checks whether the conjecture is syntactically valid and novel. Verified conjectures are sent to the prover.

The novelty of conjectures is checked using the `exact?` command in Lean, which checks whether the conjecture can be proved by existing theorems in the context. Note that this command is done with context importing the whole Mathlib4 (Lean4 standard library) and including the library and the verified conjectures. Thus, the checked novelty means that the conjecture is not already in Mathlib4, the already generated library, and the verified conjectures.

The system prompt given to the conjecturer LLM is as follows:

> You are a contributor to the mathlib4 library. Based on a given library, please generate conjectural new theorems in Lean 4 format; they do not need to be true. Do not generate statements that already appear in the list. Do not include proofs, annotations, or imports. Each new statement should begin with 'theorem' (with no annotations) and end with ':= sorry'. Additionally, use standard mathematical symbols (e.g., ∀, ∃, √) rather than Unicode escape sequences (e.g., \u2200).

3.3 Prover Loop

For each generated conjecture, the prover tries to prove it in the following process.

1. The prover LLM produces the proof code of the conjecture. If the LLM judges that the

conjecture is not provable, the prover exits the loop as failure.

2. The Lean server verifies the generated proof. If the proof is verified, the prover exits the loop as success.

3. If the maximum number of trials has been reached, the prover exits the loop with failure. Otherwise, the error message from the Lean server is returned to the LLM and we return to step 1.

The context is given to both the prover and the Lean server. Thus, not only the prover can learn the proof strategies from the context, but also it can use the theorems in the context as lemmas.

The system prompt given to the prover LLM is as follows:

> You are a contributor to the mathlib4 library. Please prove the final theorem in the given content using Lean 4. Write the Lean 4 code that directly follows ':=' in the final theorem. The code should either begin with 'by' or be a term expression. You may use the theorems in the given content as lemmas. Do not use 'sorry' in the proof. If you determine that the theorem is not provable, return an empty string instead of a proof. Do not include any additional text.

In our experiment, the maximum number of trials is set to 16.

3.4 Baseline

For comparison, we also generated theorems for these notions by a simple loop (SL) framework in which an LLM generates theorems and their proofs at once. First, the library is initialized by the user. Unlike CPL, in this simple loop baseline, the conjecturer and the prover are not separated, and the single loop is as follows:

I The LLM generates a statement and its proof in Lean 4 format based on the library, while accessing the Lean server to verify it.

II If the previous step succeeded, the generated pair of statement and proof is stored in the library. We return to step I.

Step I is similar to the prover loop. Its details are as follows:

1. The LLM produces a statement and its proof in Lean.

2. The Lean server checks the generated content. If it is verified, we exit the loop as success.

3. If the maximum number of trials has been reached, we exit the loop with failure. Otherwise, the error message from the Lean server is returned to the LLM and we return to step 1.

The system prompt given to LLM is as follows:

> You are a contributor to the mathlib4 library. Based on a given library, please generate a new theorem together with its proof in Lean 4 format. Do not output anything other than the Lean 4 code. The generated code must follow the given library and contain only the theorem statement and its proof. Do not output declarations other than theorem, such as variable, section, or namespace. Do not generate a theorem that already exists in the library. The new theorem should begin with 'theorem' (with no annotations). You may use the theorems in the given library as lemmas in the proof. Do not use 'sorry' in the proof. Additionally, use standard mathematical symbols (e.g., $\forall$, $\exists$, $\surd$) rather than Unicode escape sequences (e.g., \u2200).

As in the prover loop, the maximum number of trials is set to 16.

4 Theory

For the following reasons, it is expected that the distribution of generated theorems differs between SL and CPL. When a statement and its proof are generated at once, the distribution of generated theorems depends on both the distribution of statements and success rates of proofs. On the other hand, when multiple proofs are attempted after a statement is generated, the distribution of theorems shifts closer to the distribution of provable statements, and the influence of proof success rates diminishes.

More formally, let $s(T)$ be the probability distribution of statements T generated by the LLM, and let $r(T)$ be the probability that the LLM generates a successful proof of T. We model SL as a simplified process that generates a statement and

its proof sequentially, and outputs them if the proof is correct. CPL is also simplified and modeled as a process that attempts to generate a valid proof, after the generation of a statement, until it succeeds or the number of attempts reaches N. For simplicity, we ignore the influence of contexts.

The probability of generating a theorem T is proportional to $s(T)r(T)$ in SL, and is proportional to $s(T)\left(1-(1-r(T))^N\right)$ in CPL. Therefore, as N increases, the distribution of theorems in CPL approaches the distribution of provable statements (T such that $r(T) > 0$), and even theorems that are difficult to prove tend to be generated more frequently.

On the other hand, while the expected number of proof trials required to discover one theorem in SL is $E_{\mathrm{SL}} := (\mathbb{E}_{T\sim s}[r(T)])^{-1}$, it is

$$E_{\mathrm{CPL}} := \frac{\mathbb{E}_{T\sim s}\left[(1-(1-r(T))^N)r(T)^{-1}\right]}{\mathbb{E}_{T\sim s}\left[1-(1-r(T))^N\right]}$$

in CPL, because, when the statement T is generated, the probability of success in proving T is $1-(1-r(T))^N$ and the expected number of trials is $(1-(1-r(T))^N)r(T)^{-1}$.[3]

Since $\left(1-(1-r)^N\right)r^{-1}$ is decreasing for r, from Chebyshev's sum inequality,

$$\begin{aligned}&\mathbb{E}_{T\sim s}\left[(1-(1-r(T))^N)\right]\\ \leq&\mathbb{E}_{T\sim s}\left[(1-(1-r(T))^N)r(T)^{-1}\right]\mathbb{E}_{T\sim s}[r(T)].\end{aligned}$$

Thus, $E_{\mathrm{SL}} \leq E_{\mathrm{CPL}}$, which explains why CPL generates fewer theorems than SL.

The condition under which a theorem T_0 is more likely to be generated in CPL than in SL under the fixed number of proof trials is as follows. In SL, at one generation of a statement, the probability to find T_0 is $s(T_0)r(T_0)$ and the number of proof trials is always 1. In CPL, at one generation of a statement, the probability to find T_0 is $s(T_0)(1-(1-r(T_0))^N)$ and the expected number of proof trials is $\mathbb{E}_{T\sim s}\left[(1-(1-r(T))^N)r(T)^{-1}\right]$. Since $s(T_0) \ll 1$, the desired condition can be approximated as

$$\frac{1-(1-r(T_0))^N}{\mathbb{E}_{T\sim s}\left[(1-(1-r(T))^N)r(T)^{-1}\right]} > r(T_0),$$

which is independent of $s(T_0)$. If $r(T_0) > 0$, this can also be written as

$$\begin{aligned}&(1-(1-r(T_0))^N)r(T_0)^{-1}\\ >&\mathbb{E}_{T\sim s}\left[(1-(1-r(T))^N)r(T)^{-1}\right].\end{aligned}$$

[3] Because $(1-(1-r)^N)r^{-1}$ is actually a polynomial, we consider its value for $r=0$ as N.

Since $\left(1-(1-r)^N\right)r^{-1}$ is decreasing for r, provable theorems with sufficiently low proof success rates are more likely to be generated in CPL.

5 Experiments

We demonstrated that research-level theorems can be rediscovered by our framework and verified that in-context learning worked effectively in our framework.

5.1 Setting

In our experiments, we focus on the minor notions in general topology: semi-openness, α-openness, and preopenness. We used the following file including the definitions of these notions in Lean 4 format as the initial library.

```
import Mathlib
import Aesop

namespace Topology

variable {X : Type*} [TopologicalSpace X]

def P1 (A : Set X) : Prop :=
  A ⊆ closure (interior A)

def P2 (A : Set X) : Prop :=
  A ⊆ interior (closure (interior A))

def P3 (A : Set X) : Prop :=
  A ⊆ interior (closure A)
```

The notions P1, P2, and P3 are 'semi-open', 'α-open', and 'preopen', respectively, and are anonymized to prevent LLMs from using existing knowledge. The reason for choosing these notions is that they can be defined using only notions already present in Mathlib, while they themselves are not yet included in Mathlib, and while their mathematical importance has already been recognized and researched, they are not yet well known enough for LLM to have knowledge of their properties.

We set the following theorem as a target and focused on whether it could be generated or not:

> *The intersection of two P2 (α-open) sets is P2 (α-open)*

This theorem is important because it is the most difficult part of the proof that α-open sets form another topology (Proposition 2 in (Njåstad, 1965)). We have confirmed that this theorem cannot, at least, be naively derived from the knowledge of the LLM used in the experiment. See § 5.3.3. Whether a generated library contains the desired theorem or not is checked as follows: place the statement

of the theorem after the generated library and see if the proof is completed by using `exact?` command. If the library contains a proposition that is trivially equivalent to or stronger than the theorem, the completion succeeds. By doing so, we can accommodate variations in the formulation of the theorem within the range that the Lean server can recognize.

We used GPT-o3[4] both in CPL and in SL. For both CPL and SL, we generated libraries 20 times as follows: we generated theorems until the API usage reached 14000000 tokens.

5.2 Results

In CPL, 106 theorems were generated on average, and **the target theorem was discovered 5 times out of 20**. In SL, 328 theorems were generated on average, but **the target theorem was never generated in any of the 20 runs**. According to Fisher's exact test ($p = 0.024$), CPL is more likely to generate the desired theorem.

An example of generated proofs of this theorem is shown in Appendix A. This proof differs from the original proof by Njåstad, which suggests that the LLM found this proof independently.

The results showing that while SL generates more theorems, CPL is more likely to generate theorems that are difficult to prove are consistent with the discussion in § 4. To further verify this, we measured the proof length of the generated theorems. Figure 2 shows the distribution of proof lengths (the numbers of characters) of the theorems generated by CPL and SL. It can be observed that CPL can generate theorems with longer proofs than SL. It is known that there is a positive relationship between the length and difficulty of proofs [5] (Wu et al., 2025; Sonoda et al., 2026). Thus, this result is consistent with the theoretical analysis.

5.3 Additional Experiments

To verify the aforementioned effect of CPL independently, we conducted an additional experiment (§ 5.3.1).

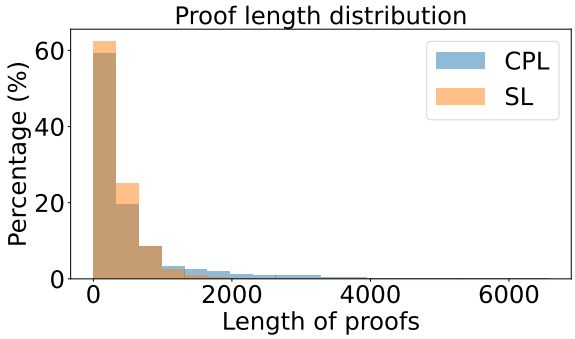


Figure 2: Distribution of proof lengths (the numbers of characters) of theorems generated by our framework and the simple loop framework.

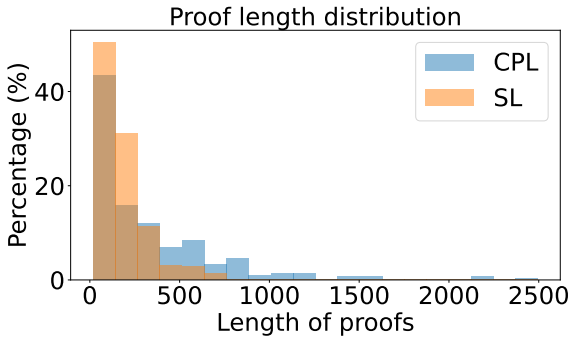


Figure 3: Distribution of proof lengths (the numbers of characters) of theorems generated by our framework and the simple loop framework without in-context learning.

We also verified that feeding the generated library as a context to the prover improves the proof ability (§ 5.3.2 and § 5.3.3).

5.3.1 Generation Without In-Context Learning

To observe the difference between CPL and SL without effects of in-context learning, we generated theorems with only the seed file as a context. In other words, for both CPL and SL, we independently conducted the first single loops several times, until the API usage reached 3000000 tokens.

Including duplicates, 309 theorems were generated in CPL and 941 theorems were generated in SL. The distributions of generated proofs are shown in Figure 3. The shift of distribution is observed. According to the Kolmogorov–Smirnov test, CPL tends to generate longer proofs than SL with a p-value of 1×10^{-13}.

The target theorem was not generated by either CPL or SL. See also the results of the following experiments.

[4] https://platform.openai.com/docs/models/o3. While GPT is currently released up to version 5.2, o3 was the latest version when this research began. Since the notions and theorems used in our experiments were designed assuming o3, GPT-5 was unsuitable because its performance was higher from the start, and thus it was not adopted.

[5] Note that in this context, "difficulty" refers to the difficulty of generating a valid proof in Lean, and "proof length" refers to the length of the Lean code; these are not necessarily the same as the difficulty or length in natural language.

5.3.2 Reproving Generated Theorems

First, we attempted to re-prove all theorems generated in CPL with two settings: one where the context includes the library generated before the theorem to be proved was generated, and one where the context includes only the definitions of the notions. As a result, in the setting with contexts, **99% of the theorems (2106/2123 theorems)** were proved, while in the setting without contexts, only **91% of the theorems (1935/2123 theorems)** were proved. According to the McNemar test, this difference is statistically significant at a p-value of 4×10^{-35}. Thus, the context improves LLMs' proof ability.

5.3.3 Proof Ability for Alpha-Open Intersection

Moreover, we attempted to re-prove the target theorem 16 times for each of the 5 contexts in which the target theorem was generated. (The average number of theorems generated until this theorem was generated is 49.) For comparison, we also attempted to re-prove it 80 times without any generated library. The procedure is the same as the prover loop, except the system prompt was changed to the following:

> You are a contributor to the mathlib4 library. Please prove the final theorem in the given content using Lean 4. Write the Lean 4 code that directly follows ':=' in the final theorem. The code should either begin with 'by' or be a term expression. You may use the theorems in the given content as lemmas. Do not use 'sorry' in the proof. If you determine that the theorem is false, return an empty string instead of a proof. Do not include any additional text.

Note that the condition not to return a proof has changed from "not provable" to "false".

As a result, **in the setup that included the generated library as context, the re-proof succeeded** 7 **times, whereas, in the setup without the library, it failed in all** 80 **attempts.** This suggests that, through in-context learning, the prover acquires the ability to prove theorems that could not be proven without it.

The generated proof of this theorem, shown in Appendix A, does not use other generated theorems as lemmas. Thus, the generated library was used for in-context learning of proof strategies rather than as a collection of lemmas for the proof.

In addition, we also asked LLMs (GPT-4o[6] and GPT-o3) to prove this theorem in natural language (English) with context including the definitions of the notions 16 times and checked the responses by hand. In the natural language experiment, we used the following system prompt:

> Please prove the following theorem. If you judge that the theorem is false, please return "False" instead of the proof.

The statement to prove given to LLMs is as follows:

> In a topological space, a set is alpha-open if it is a subset of the interior of the closure of its interior. The intersection of any two alpha-open sets is alpha-open.

As a result, GPT-4o incorrectly stated that the proposition is false 10 times and generated incorrect proofs 6 times. GPT-o3 never generated an incorrect proof, but it always judged incorrectly that the theorem is false. The fact that the majority of judgments in GPT-4o identify the theorem as false implies that the theorem was not included in GPT's knowledge. An example of a proof with gaps generated by GPT-4o is shown in Appendix B.

6 Conclusion

We presented the Conjecturing-Proving Loop, a pipeline for automatically generating mathematical conjectures and proving them in Lean 4 format. We demonstrated that our framework can rediscover a research-level theorem. We also verified that in-context learning of proof strategies works effectively in our framework.

Limitations

In our experiment, we focused on only a single target theorem and did not investigate the ability to generate broader theorems. In particular, since the target theorem was relatively easy to conjecture, refining the conjecture generation process may be necessary to produce more profound and insightful mathematical statements.

Acknowledgements

This work was supported by JST Moonshot R&D Program JPMJMS2236, JST BOOST JPMJBY24E2, JST CREST JPMJCR2015, JSPS

[6] https://platform.openai.com/docs/models/gpt-4o

KAKENHI 24K21316, 24K16077, and Advanced General Intelligence for Science Program (AGIS), the RIKEN TRIP initiative.

References

Kaito Baba, Chaoran Liu, Shuhei Kurita, and Akiyoshi Sannai. 2025. Prover agent: An agent-based framework for formal mathematical proofs. *arXiv preprint arXiv:2506.19923*.

Luoxin Chen, Jinming Gu, Liankai Huang, Wenhao Huang, Zhicheng Jiang, Allan Jie, Xiaoran Jin, Xing Jin, Chenggang Li, Kaijing Ma, and 1 others. 2025. Seed-prover: Deep and broad reasoning for automated theorem proving. *arXiv preprint arXiv:2507.23726*.

DeepSeek-AI, Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, and 181 others. 2025. DeepSeek-V3 technical report. *Preprint*, arXiv:2412.19437.

Kefan Dong and Tengyu Ma. 2025. STP: Self-play LLM theorem provers with iterative conjecturing and proving. *Preprint*, arXiv:2502.00212.

Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, and 1 others. 2022. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119.

Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1229–1241.

Jiewen Hu, Thomas Zhu, and Sean Welleck. 2024. minictx: Neural theorem proving with (long-) contexts. *arXiv preprint arXiv:2408.03350*.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. 2025. R-zero: Self-evolving reasoning LLM from zero data. *arXiv preprint arXiv:2508.05004*.

Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia LI, Mengzhou Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. 2025a. Goedel-prover: A frontier model for open-source automated theorem proving. In *Second Conference on Language Modeling*.

Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, and 1 others. 2025b. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*.

Amir Moeini, Jiuqi Wang, Jacob Beck, Ethan Blaser, Shimon Whiteson, Rohan Chandra, and Shangtong Zhang. 2025. A survey of in-context reinforcement learning. *arXiv preprint arXiv:2502.07978*.

Olav Njåstad. 1965. On some classes of nearly open sets. *Pacific journal of mathematics*, 15(3):961–970.

Gabriel Poesia, David Broman, Nick Haber, and Noah Goodman. 2024. Learning formal mathematics from intrinsic motivation. *Advances in Neural Information Processing Systems*, 37:43032–43057.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. 2025. Reliable evaluation and benchmarks for statement autoformalization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 17958–17980.

Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. 2025. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *Preprint*, arXiv:2504.21801.

Sho Sonoda, Shunta Akiyama, and Yuya Uezato. 2026. Why agentic theorem prover works: A statistical provability theory of mathematical reasoning models. *arXiv preprint arXiv:2602.10538*.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. 2023. An in-context learning agent for formal theorem-proving. *arXiv preprint arXiv:2310.04353*.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, and 1 others. 2023. Lego-prover: Neural theorem proving with growing libraries. *arXiv preprint arXiv:2310.00656*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Zheng Yuan, Wenwei Zhang, Dahua Lin, and Kai Chen. 2025. Internlm2. 5-stepprover: Advancing automated theorem proving via critic-guided search. In *2nd AI for Math Workshop@ ICML 2025*.

Shaoxiong Zhan, Yanlin Lai, Ziyu Lu, Dahua Lin, Ziqing Yang, and Fei Tang. 2025. Mathsmith: Towards extremely hard mathematical reasoning by forging synthetic problems with a reinforced policy. *arXiv preprint arXiv:2508.05592*.

Jian Zhang and Si-Cheng Tan. 2026. Automated conjecturing and theorem finding: A survey. *Journal of Computer Science and Technology*, pages 1–21.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and 1 others. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A Generated Proof of Focused Theorem

```
theorem P2_inter {X : Type*} [TopologicalSpace
    X] {A B : Set X} : P2 A → P2 B → P2 (A ∩
    B) := by
  intro hP2A hP2B x hx
  rcases hx with ⟨hxA, hxB⟩
  -- Auxiliary open neighbourhoods furnished by
     `P2 A` and `P2 B`.
  set UA : Set X := interior (closure (interior
    A)) with hUA
  set UB : Set X := interior (closure (interior
    B)) with hUB
  have hUA_open : IsOpen UA := by
    simpa [hUA] using (isOpen_interior :
      IsOpen (interior (closure (interior A))))
  have hUB_open : IsOpen UB := by
    simpa [hUB] using (isOpen_interior :
      IsOpen (interior (closure (interior B))))
  have hxUA : x ∈ UA := by
    have : x ∈ interior (closure (interior A))
    := hP2A hxA
    simpa [hUA] using this
  have hxUB : x ∈ UB := by
    have : x ∈ interior (closure (interior B))
    := hP2B hxB
    simpa [hUB] using this
  -- Combine the two neighbourhoods.
  have hO_open : IsOpen (UA ∩ UB : Set X) :=
    hUA_open.inter hUB_open
  have hxO : x ∈ UA ∩ UB := ⟨hxUA, hxUB⟩
  -- Main claim: the intersection lies in the
     relevant closure.
  have hO_sub : (UA ∩ UB : Set X) ⊆ closure
    (interior (A ∩ B)) := by
    intro y hy
    rcases hy with ⟨hyUA, hyUB⟩
    -- `y` is simultaneously in the closures of
     `interior A` and `interior B`.
    have hy_clA : y ∈ closure (interior A) := by
      -- `UA ⊆ closure (interior A)`
      have hsub : (UA : Set X) ⊆ closure
    (interior A) := by
        intro z hz
        have hz' : z ∈ interior (closure
    (interior A)) := by
          simpa [hUA] using hz
        exact interior_subset hz'
      exact hsub hyUA
    have hy_clB : y ∈ closure (interior B) := by
      have hsub : (UB : Set X) ⊆ closure
    (interior B) := by
        intro z hz
        have hz' : z ∈ interior (closure
    (interior B)) := by
          simpa [hUB] using hz
        exact interior_subset hz'
      exact hsub hyUB
    -- Show that every open neighbourhood of
     `y` meets `interior (A ∩ B)`.
    have : y ∈ closure (interior (A ∩ B)) := by
      apply (mem_closure_iff).2
      intro V hVopen hyV
      -- First refinement, intersect with `UB`.
      have hV1_open : IsOpen (V ∩ UB) :=
    hVopen.inter hUB_open
      have hyV1 : y ∈ V ∩ UB := ⟨hyV, hyUB⟩
      -- Obtain a point in `interior A`.
      rcases (mem_closure_iff).1 hy_clA (V ∩
    UB) hV1_open hyV1 with
        ⟨a, ⟨haV, haUB⟩, haIntA⟩
      -- `a` is now in `V`, `UB`, and `interior
    A`.
      have ha_clB : a ∈ closure (interior B) :=
    by
        have hsub : (UB : Set X) ⊆ closure
    (interior B) := by
          intro z hz
          have hz' : z ∈ interior (closure
    (interior B)) := by
            simpa [hUB] using hz
          exact interior_subset hz'
        exact hsub haUB
      -- Second refinement, intersect with
     `interior A`.
      have hW_open : IsOpen (V ∩ interior A) :=
    hVopen.inter isOpen_interior
      have haW : a ∈ V ∩ interior A := ⟨haV,
    haIntA⟩
      -- Obtain a point in `interior B`.
      rcases (mem_closure_iff).1 ha_clB (V ∩
    interior A) hW_open haW with
        ⟨z, ⟨hzV, hzIntA⟩, hzIntB⟩
      -- `z` lies in `V`, `interior A`, and
     `interior B`.
      have hzIntAB : z ∈ interior (A ∩ B) := by
        -- The set `interior A ∩ interior B` is
    open and contained in `A ∩ B`.
        have hS_open : IsOpen (interior A ∩
    interior B) :=
          isOpen_interior.inter isOpen_interior
        have hS_sub : (interior A ∩ interior B
    : Set X) ⊆ A ∩ B := by
          intro w hw
          rcases hw with ⟨h1, h2⟩
          exact ⟨interior_subset h1,
    interior_subset h2⟩
        have hS_to : (interior A ∩ interior B :
    Set X) ⊆
            interior (A ∩ B) :=
          interior_maximal hS_sub hS_open
        have hzS : z ∈ interior A ∩ interior B
    := ⟨hzIntA, hzIntB⟩
        exact hS_to hzS
      exact ⟨z, hzV, hzIntAB⟩
    exact this
  -- Use the neighbourhood just constructed.
  have hNhd : (UA ∩ UB : Set X) ∈ 𝓝 x :=
    hO_open.mem_nhds hxO
  have h_mem :
      x ∈ interior (closure (interior (A ∩
    B))) :=
```

```
    (mem_interior_iff_mem_nhds).2
      (Filter.mem_of_superset hNhd hO_sub)
  simpa using h_mem
```

B Example of Generated Incorrect Proof

The following is an example of a 'proof' of the theorem that the intersection of two alpha-open sets is alpha-open, generated by GPT-4o. We have only changed the line breaks to adjust the width.

> To prove the given theorem, let's first define what it means for a set to be alpha-open in a topological space.
>
> Let (X, τ) be a topological space and let $A \subseteq X$. The set A is called alpha-open if
>
> $$A \subseteq \text{int}(\overline{\text{int}(A)}),$$
>
> where $\text{int}(A)$ denotes the interior of A, and $\overline{\text{int}(A)}$ denotes the closure of $\text{int}(A)$.
>
> We aim to prove that the intersection of any two alpha-open sets is alpha-open. Let $A, B \subseteq X$ be two alpha-open sets; hence, by definition, we have:
>
> 1. $A \subseteq \text{int}(\overline{\text{int}(A)})$ 2. $B \subseteq \text{int}(\overline{\text{int}(B)})$
>
> We need to show that $A \cap B$ is also alpha-open, i.e.,
>
> $$A \cap B \subseteq \text{int}(\overline{\text{int}(A \cap B)}).$$
>
> Firstly, observe that $\text{int}(A \cap B) \subseteq \text{int}(A) \cap \text{int}(B)$.
>
> Both closure and interior operations are monotonic, so:
>
> $$\begin{aligned}\overline{\text{int}(A \cap B)} &\subseteq \overline{\text{int}(A) \cap \text{int}(B)} \\ &\subseteq \overline{\text{int}(A)} \cap \overline{\text{int}(B)}.\end{aligned}$$
>
> Taking interiors again yields:
>
> $$\begin{aligned}\text{int}(\overline{\text{int}(A \cap B)}) &\subseteq \text{int}(\overline{\text{int}(A)} \cap \overline{\text{int}(B)}) \\ &\subseteq \text{int}(\overline{\text{int}(A)}) \cap \text{int}(\overline{\text{int}(B)}).\end{aligned}$$
>
> By the assumptions that $A \subseteq \text{int}(\overline{\text{int}(A)})$ and $B \subseteq \text{int}(\overline{\text{int}(B)})$, it follows that
>
> $$\begin{aligned}A \cap B &\subseteq \text{int}(\overline{\text{int}(A)}) \cap \text{int}(\overline{\text{int}(B)}) \\ &\supset \text{int}(\overline{\text{int}(A \cap B)}).\end{aligned}$$
>
> Therefore,
>
> $$A \cap B \subseteq \text{int}(\overline{\text{int}(A \cap B)}).$$
>
> This shows that $A \cap B$ is indeed alpha-open. Therefore, the intersection of any two alpha-open sets is alpha-open.
>
> Hence, the theorem is true.

This proof clearly has a gap because, when deriving the last relation from the second-to-last relations, the direction of a subset relation is reversed.

Where Does Proof Search Spend Its Effort? A Visualization and Profiling Tool for Formal NLI

Koharu Saeki
Ochanomizu University
saeki.koharu@is.ocha.ac.jp

Daisuke Bekki
Ochanomizu University
bekki@is.ocha.ac.jp

Abstract

Recent work has substantially accelerated proof search in interactive theorem provers by integrating learned models, for both Lean and Coq. The natural language inference (NLI) counterpart lacks an analogous infrastructure: the behavior of dedicated DTT-based provers such as *wani*, inside the Japanese NLI system *lightblue*, is observable today only through verbose textual logs. This opacity blocks ML-acceleration efforts such as *Neural Wani* that need to know where the search spends its time and why it fails. We present a profiling and visualization tool for *wani*, implemented as a web-based component of the *lightblue* development environment, that exposes the proof search through a four-panel dashboard, Search Tree, Flame Graph, Rule Statistics, and Failure Analysis, each making one aspect of prover behavior directly inspectable. The tool provides the observability that ML-acceleration research in NLI currently needs but cannot easily obtain. It is released as open-source software and provided as a Docker image.

1 Introduction

Recent progress in automating proof search in ITPs like Coq (The Coq Development Team, 2024) and Lean (Moura and Ullrich, 2021) leverages machine learning. Approaches like LeanDojo (Yang et al., 2023) and Graph2Tac (Blaauwbroek et al., 2024) integrate learned models into tactic prediction, significantly reducing formalization costs. Extending these results to natural language inference (NLI) is a key challenge. Unlike theorem-proving in mathematics, NLI must cope with linguistic phenomena such as anaphora resolution and presupposition binding, which demand a semantically rich formal framework. Dependent Type Semantics (DTS, Bekki and Mineshima, 2017) provides such a framework, grounded in Dependent Type Theory (DTT, Martin-Löf, 1984). In DTS, establishing an entailment amounts to constructing, from the premises, a proof term that inhabits the type of the hypothesis; the entailment therefore comes with the proof term itself as an explicit witness, not merely a label or score. A representative implementation is *lightblue* (Tomita et al., 2025), with its built-in prover *wani* (Daido and Bekki, 2020). However, NLI proof search involves rapid subgoal proliferation and complex branching, making the search process opaque in practice. Developers currently rely on manual inspection of voluminous text logs, hindering both system improvement and neuro-symbolic efforts like *Neural Wani* (Miyagawa et al., 2026), which aims to accelerate rule selection via ML.

We introduce a profiling tool for DTT-based NLI provers, implemented as a web-based interface for inspecting *lightblue*'s proof-search results. The tool surfaces prover behavior through four coordinated views: *Search Tree*, *Flame Graph*, *Rule Statistics*, and *Failure Analysis*. By making internal processes inspectable, the tool supports both human debugging and ML-driven acceleration. The instrumentation specific to *wani* is a single optional event log; the surrounding design (the structured event schema and the post-hoc reconstruction of the four views) is prover-agnostic and could be applied to other proof-search-based NLI systems, such as the Coq-based ccg2lambda pipeline (Mineshima et al., 2015; Martínez-Gómez et al., 2016), by instrumenting their provers. We release the tool as open-source software and provide it as a Docker image.[1]

Our contributions are as follows:

- We present the first tool for profiling and visualizing proof search in DTT-based NLI provers.
- We demonstrate, on JSeM (Kawazoe et al., 2015), a curated Japanese NLI benchmark,

[1] https://wani-viz-site.fly.dev/

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 50–59

that the tool reveals concrete bottlenecks in rule exploration.

- We provide a structured event logging framework that serves as a training-data generator for ML-based acceleration methods such as *Neural Wani*, and release it as open-source software.

2 Background

2.1 Proof Search in Formal NLI

In the DTS framework, the meaning of a sentence is represented as a type. Based on the Curry–Howard correspondence, *wani* decides entailment by searching for a proof of the hypothesis from the premises. Given a context Γ constructed from the premises $S_1 : M_1, \ldots, S_{n-1} : M_{n-1}$, it attempts to construct a term H such that $\Gamma \vdash H : M_n$, where M_n is the type corresponding to the hypothesis. If such a term is found, the entailment holds. Similarly, contradictions are detected by searching for a proof of the negation of the hypothesis, i.e., a term of type $\neg M_n$ (implemented as $M_n \to \bot$). *wani* performs proof search using both forward and backward chaining. Forward chaining derives consequences from the current context, while backward chaining focuses on the goal and recursively decomposes it into subgoals required for rule application. In terms of implementation, given a goal type, the recursive function `deduce'` applies inference rules and attempts to solve the generated subgoals. Specifically, `deduce'` tries each of fourteen inference rules (ΠI, ΣI, ΠE, MEMBERSHIP, etc.) and recurses on the subgoals each accepted rule produces. A few rules (notably MEMBERSHIP and ΠE) invoke forward reasoning internally to discharge subgoals directly from the context or signature. Search is implemented as iterative-deepening depth-first search, where depth-bounded DFS is repeatedly executed with increasing depth limits, under three constraints: a depth bound, a time bound, and loop avoidance via a list of previously failed goals. A rule application can produce several alternative SubGoalSets, each consisting of an indexed list of subgoals that must all be discharged. Because a subgoal may admit multiple solutions, each subgoal may be explored in several alternative branches. As a running example, take problem 720 from JSeM: the entailment "*Taro was cried on by Hanako*" $\to$ "*Hanako cried*". The *lightblue* pipeline yields the proof search query in Figure 1. *wani* succeeds, finding two terms that inhabit the goal type. The type dependencies in G_h determine that its entity, cry, and tense components must be constructed from the corresponding components of the premise variable s_0. Consequently, the meaning representation of "Hanako cried" is already contained as part of the meaning representation of the premise "Taro was cried on by Hanako". Appendix B details the logical constraints that govern this construction.

2.2 The Need for Profiling

Figure 2 shows a typical excerpt of *wani*'s textual debug log, the sole existing means by which a developer can inspect the prover's behavior. From it, individual goals and rule applications are readable. What is not readable by eye is the aggregate, cross-cutting picture: the overall structure of the search, where wall-clock time is spent, which rules dominate the cost, and the patterns by which the prover repeatedly fails. Recovering this picture amounts to providing the observability whose absence Section 1 identified as the practical barrier between current dedicated NLI provers and ML-driven acceleration.

3 The Profiling Tool

Our tool exposes *wani*'s proof search through four coordinated views, Search Tree, Flame Graph, Rule Statistics, and Failure Analysis, each designed to answer a different question about prover behavior. All four views are reconstructed post-hoc from a structured event log embedded inside *wani*.

The tool is deployed as a live demo[1] where the Docker image and installation instructions are also available. The demo presents the four interactive views for JSeM 720, the running example of this paper. A usage guide is provided on the demo site.

3.1 Log Structuring

wani's textual debug output is not produced by a unified logging system. Instead, it consists of ad-hoc `Debug.Trace` `print` calls that developers inserted to inspect specific code paths. The profiler runs alongside these and provides a structured alternative: it adds one field to *wani*'s settings record, an optional reference to a log buffer, and, when present, records timestamped, structured events into an in-memory sequence at each step of `deduce'`. When absent, every logging call short-circuits to a no-op, leaving regular runs unaffected.

The fourteen event kinds fall into four groups (Table 1): goal lifecycle, rule trial, terminal out-

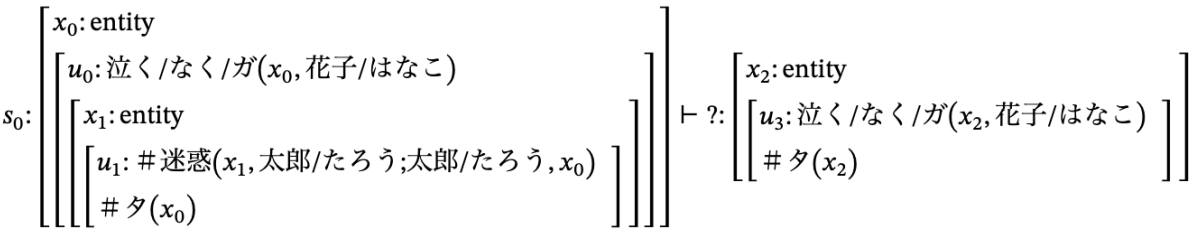


Figure 1: Proof search query for JSeM 720. The premise s_0 corresponds to *Taro was cried on by Hanako*; the goal, to *Hanako cried*.

```
0 current goal : G |- ? : (S entity)(S cry(hanako,0))(S tense(1))T
0 (S entity)(S cry(hanako,0))(S tense(1))T is not acceptable type in PiF
0 (S entity)(S cry(hanako,0))(S tense(1))T is not acceptable type in SigmaF
0 (S entity)(S cry(hanako,0))(S tense(1))T is not acceptable type in IqF
0-acceptable [SubGoalSet SigmaI ...]
0-acceptable [SubGoalSet EFQ ...]
  with SigmaI, want to prove [SubGoal entity, SubGoal S cry. S tense]
  1 current goal : G |- ? : entity
  1 depth @ deduce : entity
0 deduce failed
```

Figure 2: Excerpt of *wani*'s textual debug log on JSeM 720 (depth-1 iteration, failed). *S* denotes Σ; predicate and entity names transliterated from Japanese for readability.

Group	Event kinds
Goal lifecycle	GOALSTART, GOALEND
Rule trial	RULEATTEMPT, RULEACCEPT, RULEREJECT, RULEEXIT
Outcome	DEDUCED, DEDUCEFAILED, DEPTHEXCEEDED, AVOIDLOOP, TIMELIMIT, SPECIALCASE
Search control	ITERDEEPEN, GOALUPDATE

Table 1: The fourteen event kinds emitted by the instrumented prover, grouped by role.

come, and search control. Each event carries content fields (recursion depth, timestamp, goal string, optional rule name, and a message) together with bookkeeping fields used for post-hoc tree reconstruction (event id, kind, goal id, and optional subgoal index within the parent's SubGoalSet).

Aggregation is entirely post-hoc: pairing GOALSTART and GOALEND by goal id, inferring parent–child structure from recursion depth, and grouping events by rule name are performed outside *wani* by the SearchLog module. This separation lets the prover emit events linearly while each view derives the shape it needs (tree, flat list, histogram) on demand.

3.2 Proof Search Tree

The Search Tree visualizes the recursion structure of *wani*'s `deduce'` calls directly. Each node corresponds to one `deduce'` invocation (one goal), and parent-to-child edges encode the subgoals produced by an applied rule. Sibling nodes exhibit three distinct relations. (i) Nodes with different subgoal indices within the same SubGoalSet form an AND relation, where all must succeed. (ii) Nodes corresponding to alternative branches for the same subgoal index form an OR relation, where any one branch suffices. (iii) Nodes from different SubGoalSets, produced by different rules or context bindings, also form an OR relation. The node label `Rule (i/m#k) [children] duration` encodes multiple pieces of information in a compact form. It specifies the applied rule (ΣI, VAR, etc.), the AND position `i/m` (the i-th subgoal out of m produced by the parent rule), and an optional OR index `#k`, which appears when multiple branches exist. It also shows the total number of direct children of this node (including any that are currently collapsed) and the subtree's wall-clock duration in ms. For instance, ΣI (2/4 #1) [10] 0.7 ms denotes the 2nd of four ΣI subgoals, branch 1 of multiple alternatives, with 10 direct children and 0.7 ms duration. Nodes are colored by outcome (green for success, red for failure, orange for depth-bound exhaustion, yellow for loop avoidance, gray for time limit). For scalability on large traces, nodes at depth four or deeper are collapsed by default and can be expanded interactively.

Figure 3 shows the Search Tree for our running example, JSeM 720. The root corresponds to the

Rule	Att.	Suc.	Fail	Dep.	Loop	Time	Pend.	Rate	Tot. ms
CON	12	12	0	0	0	0	0	100%	0.01
EFQ	40	0	2	38	0	0	0	0%	0.9
IQE	48	48	0	0	0	0	0	100%	0.1
ΠE	48	0	0	48	0	0	0	0%	0.02
ΣE	68	68	0	0	0	0	0	100%	0.1
ΣF	32	0	0	32	0	0	0	0%	0.02
ΣI	18	8	8	2	0	0	0	44%	9.2
VAR	68	68	0	0	0	0	0	100%	0.04

Table 2: Rule Statistics for JSeM 720. The six outcome columns (Suc., Fail, Dep., Loop, Time, Pend.) are mutually exclusive and sum to Att.

outermost `deduce'` call; below it, ΣI and EFQ are tried as alternative root rules. Green subtrees reach leaves by forward reasoning (ΣE followed by VAR); the red subtrees mostly hit the depth bound.

3.3 Flame Graph

The Flame Graph adapts a now-standard profiling visualization (Gregg, 2016) to the proof search call stack: each cell's width encodes the total wall-clock time spent in the subtree rooted at that node, and the vertical axis encodes recursion depth. Wide cells are temporal hotspots. The view condenses the same structure shown in Figure 3 into a time-weighted form, so a developer can identify which subtree dominated wall-clock time at a single glance.

3.4 Rule Statistics

The Rule Statistics view groups every search-tree node by the rule that produced it and reports, per rule, the number of attempts, the breakdown of outcomes (success, failure, depth-bound exhaustion, loop avoidance, time limit, pending), the success rate, and the total time spent. Columns are sortable, so a developer can quickly see which rule dominates cost and which rule fails for which reason. Table 2 reports the aggregation for JSeM 720: forward-reasoning rules (VAR, CON, ΣE, IQE) succeed uniformly and contribute negligible time; ΠE, EFQ, and ΣF are attempted many times but never succeed, mostly terminating at the depth bound; backward ΣI accounts for the dominant share of total wall-clock time despite a 44% success rate.

3.5 Failure Analysis

The Failure Analysis panel groups failure events (DEDUCEFAILED, DEPTHEXCEEDED, AVOIDLOOP, TIMELIMIT) by category and, within each category, counts repeated failures on the same goal. Figure 7 (Appendix C) shows the aggregation for JSeM 720: failures are overwhelmingly DEPTHEXCEEDED (over 90% of all failure events), with the remainder DEDUCEFAILED, and the same goal recurs many times within DEPTHEXCEEDED. This indicates that the search repeatedly hits the depth bound on closely related sub-problems—a pattern that is hard to see from raw textual logs but immediately apparent here.

4 Evaluation

We evaluate the tool from two angles: §4.1 demonstrates what is revealed about the JSeM 720 running example, and §4.2 verifies that what is visible faithfully reflects *wani*'s actual search log.

4.1 Diagnostic Case Study: JSeM 720

The four panels together expose the following facts about *wani*'s behavior on JSeM 720, none of which are directly visible in the textual debug log of Figure 2:

Where time concentrates. Table 2 and Figure 4 together show that ΣI dominates total wall-clock time (9.2 ms out of approximately 10.4 ms, about 89%); all other rules together account for about 1.2 ms.

Direct guidance for ML acceleration. *wani* makes a substantial number of attempts at ΠE, ΣF, and EFQ (48, 32, and 40 respectively in our run), even though these attempts almost always terminate by exhausting the depth bound and do not contribute to the final proof. Our tool identifies these three rules as the primary targets for *Neural Wani*'s learned ranker to prune, directly guiding the development of more efficient neuro-symbolic inference for NLI.

Where failures cluster. Figure 7 (Appendix C) shows that failure events cluster on a small set of subgoals, mostly concerning the `entity` type, with the same goal recurring across many `depth_exceeded` terminations. This pattern points to a specific subgoal type where prioritization by a learned ranker could yield the largest reduction in wasted exploration.

4.2 Structural Faithfulness

wani's proof search exhibits a highly branching structure: it explores many alternative branches per goal, fails in heterogeneous ways (depth bound, loop avoidance, time limit, exhaustion of all rules), and may revisit closely related subgoals across

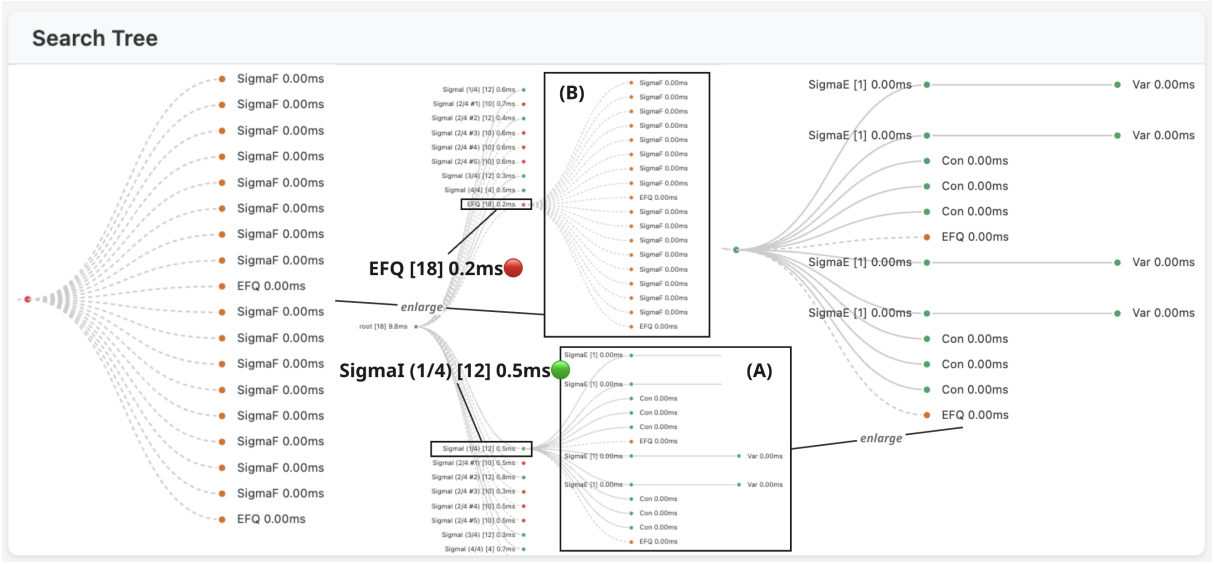


Figure 3: Search Tree for JSeM 720. The central tree shows the direct children of the root, with two enlarged callouts. **(A)** A successful ΣI sub-problem (parent node ΣI (1/4) [12] 0.5 ms in the central tree), expanded to show its forward-reasoning leaves: ΣE followed by VAR, or CON. **(B)** The stalled EFQ sub-problem (parent node EFQ [18] 0.2 ms in the central tree), where its ΣF and EFQ subgoals all exhaust the depth bound. The contrast between (A) and (B) is exactly the kind of signal a learned ranker could exploit. Each node is one deduce' invocation; colors indicate outcome (green: successful discharge; orange: depth-bound exhaustion; red: failure after all applicable rules).

iterative-deepening rounds. Because the Search Tree is reconstructed post-hoc from a flat event log rather than from explicit parent pointers, a reconstruction error would silently distort every downstream view; it must therefore recover this structure faithfully. We verify that it does by checking three structural invariants of the underlying event sequence: (i) *goal pairing*, where every GOALSTART has exactly one matching GOALEND, ensuring no dangling nodes; (ii) *outcome uniqueness*, meaning each goal carries at most one terminal outcome event, ruling out double-counting in the Rule Statistics aggregation; and (iii) *depth invariant*, whereby every GOALSTART at depth $d > 1$ has an active predecessor at depth $d - 1$, formalizing the strict depth-first property required for inferring parent–child structure from recursion depth. We apply the checks to two contrasting problems, JSeM 720 (indirect passive; 336 goals, 1414 events) and JSeM 699 (distributive predication; 1680 goals, 6120 events; query in Appendix A), and find that all three invariants hold without violation in both cases (Table 3). The structural invariants verify that the post-hoc reconstruction logic is sound at problem scales differing by nearly a factor of five and across distinct linguistic phenomena. The check is implemented as a Python script and will be included in the public release, so the same procedure can be applied to any other proof search query.

Check	JSeM 720	JSeM 699
goal pairing	336 / 336 pass	1680 / 1680 pass
outcome uniqueness	336 / 336 pass	1680 / 1680 pass
depth invariant	336 / 336 pass	1680 / 1680 pass

Table 3: Structural faithfulness checks on two JSeM problems of contrasting scale and linguistic phenomena: JSeM 720 (indirect passive; 1414 events / 336 goals) and JSeM 699 (distributive predication; 6120 events / 1680 goals). All three invariants hold without violation in both cases.

5 Related Work

LLM-assisted theorem proving. The line of work introduced in Section 1, LeanDojo, Lean Copilot (Song et al., 2025), Baldur (First et al., 2023), Proverbot9001 (Sanchez-Stern et al., 2020), COPRA (Thakur et al., 2023), Graph2Tac, and, in the NLI context, *Neural Wani*, all aim at making proof search faster by integrating learned components. Our work is complementary: rather than proposing a learned strategy, we provide observability into proof search, measuring where time is spent and exposing failure patterns, thereby enabling and evaluating such interventions.

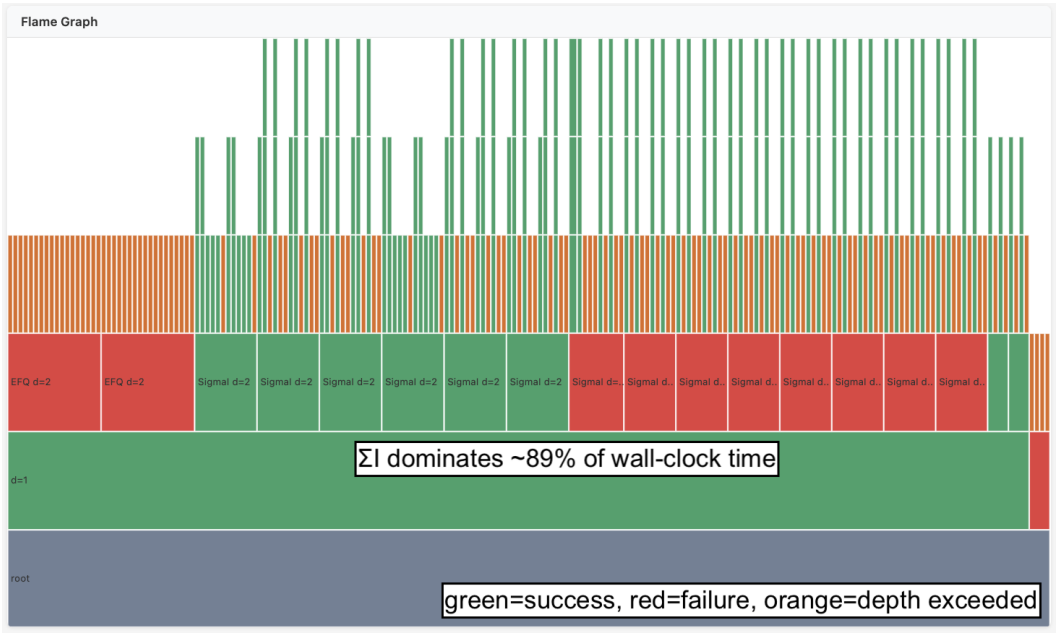


Figure 4: Flame Graph panel for JSeM 720. Cell widths are proportional to subtree wall-clock duration. Colors: green = success, red = failure, orange = depth-bound exhaustion.

Logic-based NLI. Beyond DTS, NLI has also been pursued with higher-order logic and off-the-shelf provers, notably ccg2lambda, which uses Coq to prove entailments between compositionally derived representations. Such systems share the observability gap our tool addresses and are natural portability targets.

Proof-state visualization in interactive provers. Tools such as CoqIDE, vscoq, and Lean's Infoview render the constructed proof and the current tactic state for the developer. They focus on the proof being built, not on the search that produced it: the structure of explored alternatives, the time distribution across rules, and recurring failure patterns, which are precisely the aspects our tool makes observable.

Program profiling. The Flame Graph has become a standard visualization in general software performance analysis. Our work adapts this idea to proof search, where the call structure is not explicit and must be reconstructed from logical inference traces.

6 Conclusion

We presented a profiling and visualization tool that provides observability into proof search in formal natural language inference. The tool exposes *wani*'s search behavior through four coordinated views—Search Tree, Flame Graph, Rule Statistics, and Failure Analysis—reconstructed post-hoc from a structured event log, and is released as open-source software with a Docker image.

The case study on JSeM demonstrates that the tool can surface actionable signals invisible in raw textual logs: which rules dominate wall-clock time, which rules are repeatedly attempted without contributing to the proof, and which subgoal types concentrate failure events. These are precisely the signals that a learned rule ranker needs in order to decide what to prune.

Beyond diagnostics, the structured event log doubles as a training-data generator for ML-acceleration efforts such as *Neural Wani*. Prior to this tool, search paths that led to failure were lost; the profiler now makes them fully recoverable, enabling extraction of both positive and negative training examples for a learned ranker.

Future work proceeds along three directions: (i) applying the tool to corpus-level analysis across JSeM to identify systematic bottleneck patterns beyond individual problems; (ii) porting the profiling framework to other proof-search-based NLI systems by emitting the same event schema, beginning with a Coq-based pipeline such as ccg2lambda and extending toward Lean-based provers; and (iii) directly using the collected positive and negative signals to train rule-selection heuristics within *Neural Wani*.

Limitations

Scope. The current implementation instruments *wani* specifically, but by design the prover-specific part is confined to a single component. *wani*'s only modification is one optional field in its settings record that emits timestamped, structured events. Everything downstream is independent of *wani* and operates on the event sequence alone: the fourteen-kind event schema (Table 1), the post-hoc reconstruction of the four views, and the Python invariant checker of §4.2. Porting the tool to another proof-search-based NLI system therefore reduces to emitting the same schema from that system's search procedure; the reconstruction and visualization layers are reused unchanged. We have not yet performed such a port, and exposing the schema as a shared, prover-independent log format that other systems can target is left as future work.

Rendering scalability. The Search Tree auto-collapses nodes at depth four or deeper to remain interactive at the trace sizes we have measured (up to a few thousand events). For substantially larger proofs, scaling the rendering or providing aggregate-only views is left as future work.

Timing precision. Per-goal durations are computed from the timestamps of structured event records and therefore include the (small) overhead of the logging calls themselves. Reported durations are appropriate for identifying hotspots and relative comparisons, but not for microsecond-level benchmarking.

Iterative-deepening semantics. The event log accumulates events across all iterative-deepening rounds. Users analyzing successful proofs might find the inclusion of events from previous iterative-deepening rounds counterintuitive; however, round boundaries are recoverable from ITERDEEPEN events.

Evaluation scope and ML integration. Our evaluation focuses on a small number of case studies. Corpus-level aggregate evaluation across many JSeM problems is left for future work. We have not conducted a formal user study, as the target user group (developers of DTT-based NLI provers) is highly specialized and currently limited in size, making large-scale quantitative evaluation infeasible at present; qualitative evaluation is left for future work. Finally, this paper presents the observability infrastructure required for ML-acceleration efforts; we do not ourselves implement or evaluate a learned heuristic.

Acknowledgments

This work was supported in part by JST CREST Grant Number JPMJCR2565 and JSPS KAKENHI Grant Number JP23H03452, Japan.

References

Daisuke Bekki and Koji Mineshima. 2017. *Context-Passing and Underspecification in Dependent Type Semantics*, pages 11–41. Springer International Publishing, Cham.

Lasse Blaauwbroek, Miroslav Olšák, Jason Rute, Fidel Ivan Schaposnik Massolo, Jelle Piepenbrock, and Vasily Pestun. 2024. Graph2tac: Online representation learning of formal math concepts. In *International Conference on Machine Learning*.

Hinari Daido and Daisuke Bekki. 2020. Development of an automated theorem prover for the fragment of DTS. In *Proceedings of the 17th International Workshop on Logic and Engineering of Natural Language Semantics (LENLS17)*.

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 1229–1241, New York, NY, USA. Association for Computing Machinery.

Brendan Gregg. 2016. The flame graph. *Commun. ACM*, 59(6):48–57.

Ai Kawazoe, Ribeka Tanaka, Koji Mineshima, and Daisuke Bekki. 2015. An inference problem set for evaluating semantic theories and semantic processing systems for japanese. In *Proceedings of the Twelfth International Workshop on Logic and Engineering of Natural Language Semantics (LENLS12)*, pages 67–73, Tokyo-Yokohama, Japan.

Per Martin-Löf. 1984. *Intuitionistic Type Theory Vol. 1*. Bibliopolis.

Pascual Martínez-Gómez, Koji Mineshima, Yusuke Miyao, and Daisuke Bekki. 2016. ccg2lambda: A compositional semantics system. In *Proceedings of ACL-2016 System Demonstrations*, pages 85–90, Berlin, Germany. Association for Computational Linguistics.

Koji Mineshima, Pascual Martínez-Gómez, Yusuke Miyao, and Daisuke Bekki. 2015. Higher-order logical inference with compositional semantics. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2055–2061, Lisbon, Portugal. Association for Computational Linguistics.

Nanako Miyagawa, Hinari Daido, and Daisuke Bekki. 2026. Neural Wani: Izon gata riron no tame no jidou teiri shoumeiki wani no kousokuka ni mukete (trans. neural Wani: Toward speeding up the automated theorem prover Wani for dependent type theory). In *Proceedings of the 32nd Annual Meeting of the Association for Natural Language Processing*, pages 2930–2934. In Japanese.

Leonardo de Moura and Sebastian Ullrich. 2021. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, page 625–635, Berlin, Heidelberg. Springer-Verlag.

Alex Sanchez-Stern, Yousef Alhessi, Lawrence Saul, and Sorin Lerner. 2020. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, MAPL 2020, page 1–10, New York, NY, USA. Association for Computing Machinery.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2025. Lean copilot: Large language models as copilots for theorem proving in lean. *Preprint*, arXiv:2404.12534.

Amitayush Thakur, George D. Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. 2023. An in-context learning agent for formal theorem-proving.

The Coq Development Team. 2024. *The Coq Proof Assistant Reference Manual*.

Asa Tomita, Mai Matsubara, Hinari Daido, and Daisuke Bekki. 2025. Natural language inference with CCG parser and automated theorem prover for DTS. In *Proceedings of the Second Workshop on the Bridges and Gaps between Formal and Computational Linguistics (BriGap-2)*, pages 1–7, Düsseldorf, Germany. Association for Computational Linguistics.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. 2023. Leandojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 21573–21612. Curran Associates, Inc.

A Proof Search Query for JSeM 699

Figure 5 shows the proof search query for JSeM 699, used as a contrasting example in Section 4.2. Compared with the running example JSeM 720 (Figure 1), it differs in both linguistic phenomenon (distributive predication versus indirect passive) and search scale (1680 versus 336 goals).

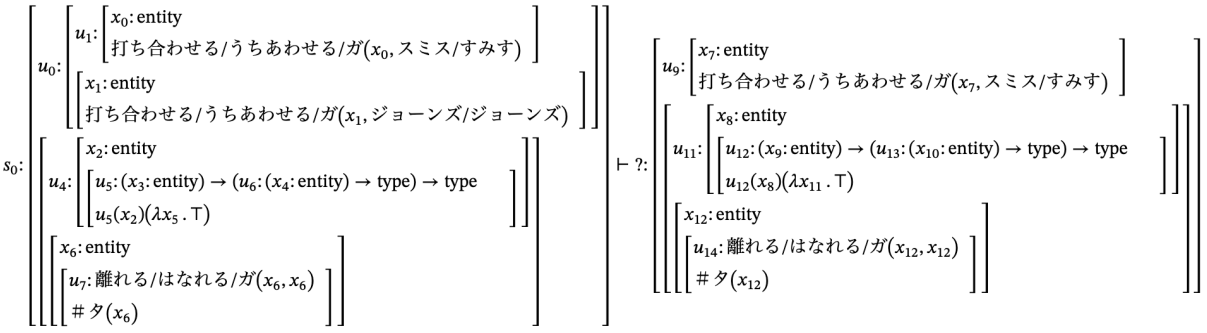


Figure 5: Proof search query for JSeM 699.

B Proof-Structure Dependency Pattern for JSeM 720

For JSeM 720, the search log records that *wani* returns two proof terms inhabiting the goal G_h (`deduced 2 trees` at depth 1). At the root, ΣI is accepted. At depth 2, the RULE_ACCEPT events break down as ΠE 24, EFQ 18, VAR 16, TOPI 2, and ΣF 2. The pattern itself is determined by the type dependencies in the goal. $G_h = \Sigma\, x_2{:}\mathsf{entity}.\, \Sigma\, u_3{:}\mathsf{cry}(x_2, \mathsf{hanako}).\, \mathsf{tense}(x_2)$ is inhabited by a triple $\langle a, \langle b, c\rangle\rangle$ with $a : \mathsf{entity}$, $b : \mathsf{cry}(a, \mathsf{hanako})$, and $c : \mathsf{tense}(a)$. Inside s_0, the only choice of a for which both the dependent cry and tense components are well-typed is x_0: the only proof of $\mathsf{cry}(\cdot, \mathsf{hanako})$ available in s_0 has type $\mathsf{cry}(x_0, \mathsf{hanako})$, and the only tense term in s_0 has type $\mathsf{tense}(x_0)$. Once $a = x_0$ is fixed, u_0 and $\mathsf{tense}(x_0)$ are forced as the components for b and c. The components specific to the adversative-passive marking (x_1, u_1) cannot participate in this well-typed assignment.

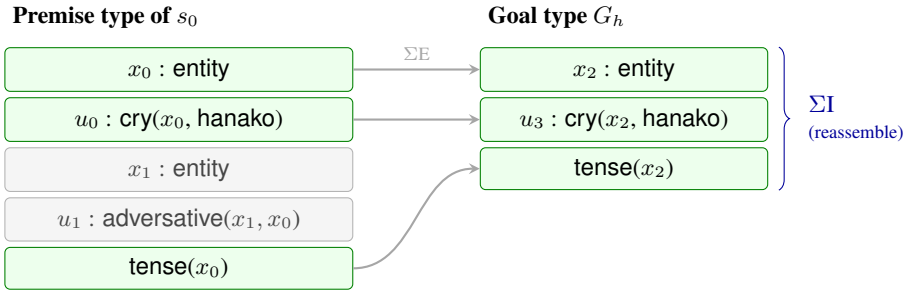


Figure 6: Dependency pattern for any inhabitant of G_h: ΣI at the root combines components projected from the premise variable s_0 via ΣE (each arrow stands for a chain of projections walking down s_0's nested Σ structure). The components specific to the adversative-passive marking (x_1, u_1, in gray) cannot contribute on type-coherence grounds. Predicate names are romanized from the original DTS output of Figure 1.

C Failure Analysis Panel for JSeM 720

Figure 7: Failure Analysis panel for JSeM 720.

Visual Question Answering and the relation between language, perception and the world

Staffan Larsson **Bill Noble** **Robin Cooper**
Centre for Linguistic Theory and Studies in Probability (CLASP)
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg, Sweden
staffan.larsson@ling.gu.se bill.noble@gu.se cooper@ling.gu.se

Abstract

We outline a general account of VQA using a perception-oriented formal semantic framework. We believe that it is instructive to describe VQA not only in terms of an engineering challenge, but also as a linguistically and philosophically relevant task that can help us better understand the relation between language, perception and the world. Specifically, we will argue that linguistic meaning helps us structure our takes on visual scenes, enabling us to classify situations so that we e.g. can answer questions and determine whether a sentence correctly describes a scene.

1 Introduction

When studying abstract matters like the relation between language, perception and the world, it is often helpful to look at specific problems. Visual Question Answering (VQA) is a machine learning task where the model is provided with an image and a natural language question and must provide an answer to the question based on the image. The task is focused on the natural language and visual understanding capabilities of the model. Many VQA datasets, such as the eponymous dataset from Antol et al. (2015), are susceptible to relatively superficial correlations between the inputs and the answer (Zhang et al., 2016; Agrawal et al., 2016; Goyal et al., 2019), which allow machine learning models to avoid the complexity involved in associating language and world more generally.

While Visual LLMs generalize better on VQA and related tasks when compared to older vision-and-language models, LLMs are essentially "black boxes" which can be trained and deployed but whose inner workings are opaque. Hence, LLMs do not necessarily help us derive insights into the relation between language, perception and the world.

In formal semantics in the Montague tradition and Possible World Semantics, the meaning of a word such as "dog" is taken to be a function from possible worlds (and times) to its extension, i.e. the set of all dogs in that world. Such logical representations, however, have no natural interpretation in terms of neural implementation (be it artificial or biological), and offer no biologically plausible explanation of how language is related to sensory perception and the physical world. In modern computational linguistics the meaning of a word such as "dog" might be related to a statistically-based perceptual classifier that is able to recognize dogs or give a probability estimation that a given individual is a dog. This corresponds to our ability to recognize a dog when we see one and is also much more hopeful in terms of neural modelling.

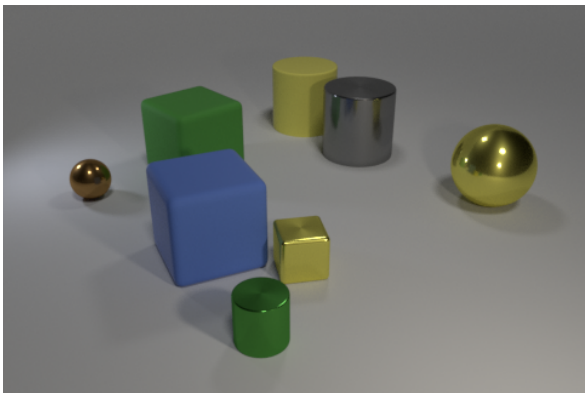


Figure 1: Consider this example from the CLEVR dataset (Johnson et al., 2016, image ID: 11568; question ID 115673). What is involved in answering the question about the scene? Q: *What color is the cylinder in front of the yellow cube?*

We adopt a view of linguistic meaning that relates to the notion of classifiers while at the same time preserving important techniques from formal semantics, such as the treatment of compositionality.

In this paper, we outline a general account of VQA, taking the CLEVR data set (see Figure 1) as an example and using a perception-oriented for-

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 60–70

mal semantic framework.[1] We believe that it is instructive to describe VQA not only in terms of an engineering challenge, but also as a linguistically and philosophically relevant task that can help us better understand the relation between language, perception and the world. Specifically, we will argue that linguistic meaning helps us structure our *take* on a visual scenes, enabling us to classify situations so that we can e.g. answer questions and determine whether a sentence correctly describes a scene.

2 TTR

TTR is a theory of types with records (Cooper and Ginzburg, 2015; Cooper, 2023). It is a broad semantic framework intended to account for how agents dynamically understand and interact with each other and the world. Thus, TTR accounts have been proposed for a multitude of linguistic and language related phenomena, including perception (Larsson, 2015, 2018), spatial and pragmatic reasoning (Dobnik and Cooper, 2017; Breitholtz, 2020), learning (Larsson and Cooper, 2009; Larsson, 2020b), quantification (Cooper, 2013; Lücking and Ginzburg, 2022; Cooper, 2023) and humour (Maraev et al., 2021).

We can only give a brief and partial introduction to TTR here; see also Cooper (2005, 2012); Cooper and Ginzburg (2015); Cooper (2023). To begin with, $s : T$ is a judgment that some s is of type T. To make explicit who is making this judgment, the of-type relation may be subscripted with an agent A, as in $:_A T$. One *basic type* in TTR is Ind, the type of an individual; another basic type is $\mathbb{R}$, the type of real numbers. Given that T_1 and T_2 are types, $T_1 \rightarrow T_2$ is a *functional type* whose domain is objects of type T_1 and whose range is objects of type T_2.

Next, we introduce *records* and *record types*. If $a_1 : T_1, a_2 : T_2(a_1), \ldots, a_n : T_n(a_1, a_2, \ldots, a_{n-1})$, where $T(a_1, \ldots, a_n)$ represents a type T which depends on the objects $a_1, \ldots, a_n$, the record to the left in Figure 2 is of the record type to the right.

In Figure 2, $\ell_1, \ldots \ell_n$ are *labels* which can be used elsewhere to refer to the values associated with them. A sample record and record type is shown in Figure 3.

Types constructed with predicates may be *dependent*. This is represented by the fact that arguments to the predicate may be represented by labels used on the left of the ':' elsewhere in the record type. In Figure 3, the type of c_{man} is dependent on ref (as is c_{run}).

$$\begin{bmatrix} \ell_1 = a_1 \\ \ell_2 = a_2 \\ \ldots \\ \ell_n = a_n \\ \ldots \end{bmatrix} : \begin{bmatrix} \ell_1 : T_1 \\ \ell_2 : T_2(l_1) \\ \ldots \\ \ell_n : T_n(\ell_1, l_2, \ldots, l_{n-1}) \end{bmatrix}$$

Figure 2: Schema of record and record type

$$\begin{bmatrix} \text{ref} = \text{obj}_{123} \\ c_{man} = \text{prf}(\text{man}(\text{obj}_{123})) \\ c_{run} = \text{prf}(\text{run}(\text{obj}_{123})) \end{bmatrix} : \begin{bmatrix} \text{ref} : \text{Ind} \\ c_{man} : \text{man}(\text{ref}) \\ c_{run} : \text{run}(\text{ref}) \end{bmatrix}$$

Figure 3: Sample record and record type

If r is a record and ℓ is a label in r, we can use a *path* $r.\ell$ to refer to the value of ℓ in r. Similarly, if T is a record type and ℓ is a label in T, $T.\ell$ refers to the type of ℓ in T. Records (and record types) can be nested, so that the value of a label is itself a record (or record type). As can be seen in Figure 3, types can be constructed from predicates, e.g., "run" or "man". Such types are called *ptypes* and correspond roughly to propositions in first order logic.

A fundamental type-theoretical intuition is that something of a ptype T is whatever it is that counts as a proof of T. One way of putting this is that "propositions are types of proofs". In Figure 3, we simply use prf(T) as a placeholder for proofs of T; below, we will show how low-level perceptual input can be included in proofs.

Judgements (and other type actions, see Cooper (2023)) are regulated by action rules of the form

$$(1) \quad \frac{\varphi_0, \varphi_1, \ldots, \varphi_n}{\psi}$$

Each φ_i is either an action or some other condition that must hold true and ψ must be a type action such as a judgement. The wavy line as opposed to a straight line indicates that this is not a rule of inference but that the premises above the line holding *license* or *afford* the action below the line. Thus there is no guarantee that the agent will carry out the action even if all the premises hold.

[1] https://github.com/GU-CLASP/types-vectors-spiking-neurons/

3 Related work

VQA is a rich test bed for investigating the compositional nature of– and relationships between language, perception, and reasoning. Because of its structure, the CLEVR dataset has been used extensively to test systems that explicitly combine these components. The Memory, Attention, and Composition (MAC) network (Hudson and Manning, 2018), is a modular neural network with a recurrent unit that sequentially decomposes the question into operations that retrieve information from the image representation. While the MAC architecture is designed to accommodate compositional reasoning, it does not create explicit symbolic representations of the image or question. One such approach is that of Johnson et al. (2017), who trains a sequence-to-sequence *program generator* that transforms the natural language question into a sequence of functions. A neural module network, with modules for each function, is then used to sequentially execute the program and derive an answer. Improving on the already high accuracy of these previous two methods, Yi et al. (2018) achieves near-perfect results on CLEVR by introducing a module that generates symbolic sense representations. This allows for the use of a deterministic program executor that answers the parsed question. A key insight of these approaches is already present in the ground-truth question semantics of CLEVR itself: the meaning of a question can be represented as a procedure to determine its answer. This insight can also be found in the treatment of questions in Formal Semantics; by introducing a neuro-symbolic system with TTR at its core, we hope that insights from VQA can be generalized to other domains such as perceptually-grounded linguistic interaction.

In line with this proposal, there have been recent efforts to combine neural network representations with linguistic formal semantics. For example, Liu and Emerson (2022) use a Convolutional Neural Network (CNN) to extract entity representations from images. Treating a collection of these representations as a world model, they learn probabilistic lexical and compositional functions that take the entity representations in input and use image captions as the learning signal. The framework used in that work is inspired by linguistic theory, but only attempts to model very simple predicate-argument structures. Likewise, Bekki et al. (2021) use neural network-based entity representations embedded in a dense vector space, but use these representations as the basis for interpreting types in a dependent type system.

Looking more specifically to work on VQA involving TTR, Dobnik and Cooper (2017) assume data in the form of a point map representing a visual scene. The authors propose two functions which (if taken together) take a point map and produces a set of record types such that records of that type would specify an individual, a witness of a ptype predicating that the individual occupies a certain region, and a witness of a ptype predicating the property of the individual. However, only the types of such records are provided, not the records themselves.

Matsson et al. (2019) can be regarded as an implemented variant of the account in Dobnik and Cooper (2017). Matsson's implementation uses YOLO classifiers and is limited to polar questions. You Only Look Once (YOLO) (Redmon et al., 2016) is a neural network model for image recognition that detects and classifies objects in an image. So here, the starting point is an image rather than a point map as in Dobnik and Cooper (2017). In YOLO, each detected object is represented as bounding box coordinates, a class label and a confidence score.

In both Dobnik and Cooper (2017) and Matsson et al. (2019), representations stay on the type level and witnesses takes are not represented as such. This leaves open the question of what constitutes a witness for a ptype, whereas we want to suggest an answer this question, namely that witnesses of ptypes consists of tuples of high-level perceptual data (H-data) associated with the individuals involved in the ptype (typically, the arguments of a predicate).

The account in Utescher (2019) is more similar in spirit to the approach suggested in this paper, both in representing witness takes explicitly and in doing lazy classification conditioned by the need to answer questions. However, Utescher does not address the problem of structuring perceptual data to match the semantics of linguistic expressions as we do here.

We believe that the account presented below is an improvement both technically and philosophically by showing how, when relating linguistic expressions to visual scenes, we *use linguistic meaning to structure our take on the visual scene*, which includes finding the referents of linguistic expressions referring to objects in the scene. This also allows us to handle predicates of arity 2 or more in an elegant way, whereas the examples in Utescher

(2019) are limited to 1-place predicates. Furthermore, the account presented here is meant to generalise over specific kinds of visual data and classifiers.

4 A general TTR account of VQA

Larsson (2011, 2013, 2020a); Larsson et al. (2021); Larsson and Cooper (2021); Larsson (2021) present work towards a general formal perception-oriented semantics. Here, we revise and extend this account to fit with the VQA task. In Section 5 we will show how the general account can be adapted to the CLEVR dataset, where H-data is regarded as CLEVR scene graphs.

4.1 Low-level and high-level perceptual data

When talking about an agent's perception of a sitation, we can call the "raw" input "low level perceptual data" (L-data). This can be e.g. the retinal image in a biological system, a digital image from a camera connected to a computer (as in Matsson et al. (2019) and Utescher (2019)), or a point map (as in Dobnik and Cooper (2017)). We will use *LScene* for the type of L-data from a scene (a visually perceived situation).

When processing perceptual data, L-data is turned into "high level perceptual data" (H-data). H-data is the result of processing L-data, and this processing may include e.g. object detection, individuation, feature detection and compression. Individuation is needed to pick out the part of L-data that concerns a single individual. A biological example of H-data is visual cortical information (simplifying somewhat, the output of the visual cortex when fed the retinal image as input). Computational examples include a set of pairs of point maps and associated features (Dobnik and Cooper, 2017), a set of image and a list of tuples, each consisting of a bounding box, a label and a confidence value (as in YOLO), or an image and a set of bounding boxes corresponding to detected objects.

We can say that H-data is the result of applying a function f_{pp} ('pp' for 'perceptual processing') to L-data. If an agent A is visually perceiving a real-world situation s, the L-data on A's retina is s^A_L : *LScene* and the H-data in the output layer of A's visual cortex is

(2) $s^A_H = f_{pp}(s^A_L)$: *HScene*

where

(3) f_{pp} : *LScene* $\rightarrow$ *HScene*

is the function of A's visual cortex.

We will assume that H-data is individuated, which in TTR terms means that sections of H-data is associated with different objects of type *Ind*[2]. We define a type *HScene* as a list of H-data items corresponding to individuals in a perceived scene.

(4) *HScene=List*($\begin{bmatrix} \text{id} & : & \textit{Ind} \\ \text{data} & : & \textit{HData} \end{bmatrix}$)

We can now define a scene-specific judgement of being of a type *Ind* in a scene s:

(5) For h:*HScene*, $a :_h$ *Ind* iff for some n, $a = h.n.\text{id}$ (and for all i, $a = h.i.\text{id}, i = h$)

We use a hash table to access H-data associated with an individual (i.e., an object of type *Ind*)[3].

(6) If h : $HScene$ and $a :_s$*Ind* then $\#_h a = h.n.data$ for n such that $a = h.n.id$

The scope of the H-data associated with an individual a is basically "any available information that can be relevant to understanding an utterance referring to a". If information about other things than a, for example information about other individuals, is needed to understand an utterance referring to a, then such information can be included in #a.

4.2 Perceptual data as evidence for ptypes

In general, we will be using H-data as evidence/proofs for ptypes, echoing the dictum "propositions are types of proofs". We will here limit our analysis to ptypes whose arguments are individuals (objects of type *Ind*). Ptypes constructed from an n-place predicates with arguments $a_1, \ldots, a_n$ have an n-tuple $\langle \#a_1, \ldots, \#a_n \rangle$ as evidence.

(7) $\langle$#a$\rangle$: yellow(a)
$\langle$#a$\rangle$: cube(a)
$\langle$#b, #a$\rangle$: in-front(b,a)
$\langle$#b$\rangle$: cylinder(b)

Judging whether some H-data evidence is of a ptype can be done using classifiers,[4] for example

[2]For most datasets, making individuation explicit in the way we do here is trivial.

[3]The hash table is thus relative to an HScene h. However, we will be surpressing this subscript for the remainder of this paper.

[4]The examples (8) and (9) may appear confusing to a type-theorist. The left-hand side is a judgement in type theory whereas the right-hand side belongs not to type theory but to the world of classifiers. The classifiers are used to place constraints on what judgements are available in the type theory.

(assuming a colour classifier κ_{col} whose domain is *Hdata* and whose range are colour labels RED, BLUE etc, and assuming that a and b are variables over objects of type *Ind*):

(8) $\langle \#a \rangle$: red(a) if $\kappa_{\text{col}}(\#a)$=RED
$\langle \#a \rangle$: blue(a) if $\kappa_{\text{col}}(\#a)$=BLUE
...

...and similarly for 2-place ptypes assuming a left-or-right classifier κ_{lor}:

(9) $\langle \#a, \#b \rangle$: right(a, b) if $\kappa_{\text{lor}}(\#a, \#b)$=RIGHT;
$\langle \#a, \#b \rangle$: left(a, b) if $\kappa_{\text{lor}}(\#a, \#b)$=LEFT

4.3 Natural language semantics

In Larsson (2020a), meanings of linguistic expressions are defined as objects of type *Mng*[5]:

(10) $\mathit{Mng}=\begin{bmatrix} \text{bg} & : & \mathit{RecType} \\ \text{intrp} & : & \text{bg}\rightarrow\mathit{Type} \\ \text{clfr} & : & \text{bg}\rightarrow\mathit{Type} \end{bmatrix}$

Intuitively, for a declarative utterance with meaning p:*Mng*, p.bg is the type of the context in p is appropriately uttered, p.intrp is the context-independent meaning of the utterance (a function from a situation to a type) and p.clfr is a function that evaluates whether the utterance of p holds of the situation.

In general, the bg field can be understood as capturing the presupposition of a linguistic expression. When interpreting linguistic expressions referring to visual scenes, an agent needs to structure their take on the scene to reflect these presuppositions. For example, "the red cube" needs to be mapped to a specific object (which is red and a cube) in the agent's take on the scene. This means that the agent, starting from low-level perceptual data, needs to individuate and object which is classified as both red and as cube (and also to make sure no other object can be individuated which can be classified as red and a cube).

For an utterance u of a declarative sentence with meaning p:*Mng* and a situation s:p.bg, we can provide an action rule specifying when an agent A may judge (A's take on the situation) s to be of the type specified by the (situated interpretation of) u in s.

(11) $\dfrac{p :_A \mathit{Mng} \qquad s :_A p.\text{bg} \qquad p.\text{clrf}(s) = p.\text{intrp}(s)}{s :_A p.\text{intrp}(s)}$

[5]We are here disregarding the 'params' field.

So an agent is licensed to judge s to be of the type T specified by the (situated interpretation of the) u in s provided that s fulfills the presuppositions of p, and A perceives the situation to be of type T. For example, the meaning of the sentence p="The cylinder in front of the yellow cube is green" could be:[6]

(15) p=$\begin{bmatrix} \text{bg} = \begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{colour}_{\text{x}} & : & \text{yellow(x)} \\ \text{shape}_{\text{x}} & : & \text{cube(x)} \\ \text{y} & : & \mathit{Ind} \\ \text{place}_{\text{yx}} & : & \text{in-front(y,x)} \\ \text{shape}_{\text{y}} & : & \text{cylinder(y)} \end{bmatrix} \\ \text{intrp}=\lambda r :\text{bg}.\left[\text{colour}_{\text{y}}:\text{green}(r.\text{y})\right] \\ \text{clfr} =\lambda r :\text{bg}.\ \left[\text{colour}_{\text{y}}:T_{\text{col}}(r.\text{y})\right] \end{bmatrix}$

where T_{col} is a dependent type that takes a tuple of relevant H-data, calls a classifier function κ_{col}, and returns a ptype, e.g. red(r.y) or green(r.y) depending on the output of the classifier:

(16) $T_{\text{col}} : \mathit{Ind} \rightarrow \mathit{Type}$

(17) $T_{\text{col}}(x) =$
$\begin{cases} \text{red}(x) \text{ if } \kappa_{\text{col}}(\langle \#x \rangle)=\text{RED} \\ \text{blue}(x) \text{ if } \kappa_{\text{col}}(\langle \#x \rangle)=\text{BLUE} \\ \ldots \end{cases}$

For a two-place relation[7]:

[6]We are here and for the remainder of this paper for simplicity of exposition ignoring the uniqueness presupposition of the definite article. Nevertheless, for completeness we indicate in this footnote how to remedy this.

(12) If T : *RecType* and $\ell \in \text{labels}(T)$, then $T_{\text{uniq}(\ell)}$ is a type.

(13) $o : T_{\text{uniq}(\ell)}$ iff $o : T$ and there is an a such that for all $r :_H T$, $r.\ell.\text{x} = a$

Here, we use '$:_H T$' for a type judgement relative to an H-scene H, to establish whether T could be instantiated with respect to H, that is, whether a record of this type could be created from H with respect to the basic types and ptypes in T.

Given the above, the meaning of the sentence is now

(14) $\begin{bmatrix} \text{bg} = \begin{bmatrix} \text{o}_1 : \begin{bmatrix} \text{x:}\mathit{Ind} \\ \text{colour:yellow(x)} \\ \text{shape:cube(x)} \end{bmatrix}_{\text{uniq(x)}} \\ \text{o}_2 : \begin{bmatrix} \text{x:}\mathit{Ind} \\ \text{place:in-front(x,}\Uparrow\text{o}_1\text{.x)} \\ \text{shape:cylinder(x)} \end{bmatrix}_{\text{uniq(x)}} \end{bmatrix} \\ \text{intrp}=\lambda r\,\text{bg}\,.\ \left[\text{c:green}(r.\text{o2.x})\right] \\ \text{clfr} =\lambda r\,\text{bg}\,.\ \left[\text{c:}T_{\text{col}}(r.\text{o2.x})\right] \end{bmatrix}$

Other definitions in this paper would need to be adapted accordingly.

[7]We are here assuming that κ_{lor} always returns LEFT or RIGHT.

(18) $T_{\text{place}}(x, y)$ =
$\begin{cases} \text{left}(x, y) \text{ if } \kappa_{\text{lor}}(\langle \#x, \#y \rangle) = \text{LEFT} \\ \text{right}(x, y) \text{ if } \kappa_{\text{lor}}(\langle \#x, \#y \rangle) = \text{RIGHT} \end{cases}$

Assuming that s provides information that can be used by a classifier to decide whether the cylinder in front of the yellow cube is green or not, an agent A may judge $s :_A \text{green}(s.\text{y})$ provided $T_{\text{col}}(s.\text{y}) = \text{green}(s.\text{y})$.

As mentioned above, H-data tuples are regarded as objects of ptypes that witness (i.e. provide evidence or proof of) the ptype. Consequently, a record type containing ptypes (such as the bg field above) can be instantiated by a record where values of ptype fields are H-data tuples. We use such records to represent an agent's take on a scene, a so-called 'witness take' . A situated interpretation of a linguistic expression e is attained by applying $[e]$.bg to a witness take s.

Assume that the declarative sentence above is stated about a focus situation s_{123} to an agent A whose take on this situation is

(19) $\text{s}^A_{123} = \begin{bmatrix} \text{x} & = & \text{a} \\ \text{colour}_\text{x} & = & \langle \#\text{a} \rangle \\ \text{shape}_\text{x} & = & \langle \#\text{a} \rangle \\ \text{y} & = & \text{b} \\ \text{place}_\text{yx} & = & \langle \#\text{b}, \#\text{a} \rangle \\ \text{shape}_\text{y} & = & \langle \#\text{b} \rangle \\ \text{colour}_\text{y} & = & \langle \#\text{b} \rangle \end{bmatrix}$

To evaluate whether p is true for s^A_{123}, A computes p.intrp(s^A_{123}) and p.clfr(s^A_{123}) and checks for equality.

(20) p.intrp(s^A_{123})=
$\lambda r{:}\begin{bmatrix} \text{x} & :\textit{Ind} \\ \text{colour}_\text{x} & :\text{yellow(x)} \\ \text{shape}_\text{x} & :\text{cube(x)} \\ \text{y} & :\textit{Ind} \\ \text{place}_\text{yx} & :\text{in-front(y,x)} \\ \text{shape}_\text{y} & :\text{cylinder(y)} \end{bmatrix}.\left[\text{colour}_\text{y}{:}\text{green}(r.\text{y})\right](\text{s}^A_{123})=$
$\left[\text{colour}_\text{y}{:}\text{green(b)}\right]$

(21) p.clfr(s^A_{123}) = $\left[\text{colour}_\text{y}{:}\text{red(b)}\right]$
assuming that $\kappa_{\text{col}}(\langle \#\text{b} \rangle)$=RED and hence T_{col}(b)=red(b).

In this case, the equality fails and hence p is regarded as not true of s_{123}.

4.4 Questions

Now, what about questions? In this case, we suggest that the situated interpretation is not a Ptype but instead a function from an answer (in this case, a predicate function) to a Ptype. A question q="What color is the cylinder in front of the yellow cube?" could then have the meaning:

(22) q=
$\begin{bmatrix} \text{bg} = \begin{bmatrix} \text{x} & :\text{Ind} \\ \text{colour}_\text{x} & :\text{yellow(x)} \\ \text{shape}_\text{x} & :\text{cube(x)} \\ \text{y} & :\text{Ind} \\ \text{place}_\text{yx} & :\text{in-front(y,x)} \\ \text{shape}_\text{y} & :\text{cylinder(y)} \end{bmatrix} \\ \text{intrp}=\lambda r :\text{bg}.\lambda P{:}(\textit{Ind}\rightarrow\textit{Type}).\left[\ \text{colour}_\text{y} : P(r.\text{y})\ \right] \\ \text{clfr} =\lambda r :\text{bg}.\left[\text{colour}_\text{y}{:}T_{\text{col}}(r.y)\right] \end{bmatrix}$

Whereas for declaratives, we compared the interpretation output with the classifer output, for questions these functions have different roles. Specifically, the interpretation function allows an agent to compute a Ptype from the question and an (elliptical) answer, whereas the classifier function can be used to compute the answer to the question based on the agent's take on the situation referred to. The former is useful in integrating an agent's verbal answer to the question[8]. The latter is useful for figuring out the answer (which could be used when providing a verbal answer).

Assume that q="What color is the cylinder in front of the yellow cube?" is asked about a focus situation s_{123}. This question presupposes that there is a (unique) yellow cube and a (unique) cylinder in front of it. An agent who is asked this question might use their take on the scene to find an answer. The idea here is that the answer to a question is attained by applying a function q.clfr derived from the question to the take on the scene that the question is about. This requires the scene to be of the type specified by q.bg.

(23) q.clfr =
$\lambda r{:}\begin{bmatrix} \text{x} & :\textit{Ind} \\ \text{c}_\text{col-x} & :\text{yellow(x)} \\ \text{c}_\text{shape-x} & :\text{cube(x)} \\ \text{y} & :\textit{Ind} \\ \text{c}_\text{fob-yx} & :\text{in-front(y,x)} \\ \text{c}_\text{shape-y} & :\text{cylinder(y)} \end{bmatrix}.\ \left[\text{colour}_\text{y}{:}T_{\text{col}}(r.\text{y})\ \right]$

4.5 Finding an answer to a question

We can imagine the question above being asked about a focus situation s_{123} to an agent A whose take on this situation is s^A_{123} as above.

Finding the answer to a question about the focus situation involves intepreting the question in light of one's take on the focus situation. Formally, the question function is applied to the take on the focus situation to yield an answer to the question.

[8] Assuming that a response r="green" has the meaning [r].intrp $= \lambda x : \textit{Ind}.\text{green}(x)$ so that q.intrp(s)([r].intrp) $= \left[\text{colour}_\text{y}{:}\text{green(a)}\right]$.

(24) q.clfr(s^A_{123}) = $\left[\begin{array}{ccc} \text{colour}_y & : & \text{red(b)} \end{array}\right]$

again assuming that $\kappa_{\text{col}}(\langle \#b \rangle)$=RED.

Given the above, for a question q we want to take an H-scene H and construct a take on the focussed witness situation s:q.bg that q.clfr can be applied to to find the answer to q. For ease of exposition, we will below use T_{bg} for q.bg. The approach we will take is this: From H and T_{bg}, create s_H such that $s_H : T_{\text{bg}}$. We want this process to work for any kind of H-data, as long as we have access to classifiers that can take that data as input and produce the required judgements.

4.6 Definition of witness takes

To build $s_H : T_{\text{bg}}$ from H, the idea is to label the individual objects in H so that the ptype classifier results (ptypes) come out as specified in T_{bg}. To do this, we start from all possible labellings of (H-data concerning) individuals in the H-data and then filter them out by applying classifiers when typechecking against the ptypes in T_{bg}.

Assuming an scene H : *HScene*, we create records from H, enumerating all possible labellings (records) $\mathcal{S}$ of H that match T_{bg}. For a record type T and an H-scene H, there is a function $\# : \mathit{Ind} \rightarrow \mathit{Hdata}$ and a set of records $\mathcal{S}$ (the set of witness takes) such that:

- For each $h \in H$, $\#a = h$.data iff h.ind=a
- For each $s \in \mathcal{S}$:
  - for all ℓ such that $T.\ell$: *Ind*, there is an $a :_H \mathit{Ind}$ such that $s.\ell = a$
  - for all ℓ such that $T.\ell = \langle f, L \rangle$ where $L = \langle \ell_1, \ldots, \ell_n \rangle$, $s.\ell = \langle \#(s.\ell_1), \ldots \#(s.\ell_n) \rangle$
  - $s : T$

Given the above, we can now tentatively provide an action rule licensing an agent A answering a question q : *Question* based on perceptual data H : *HScene*.[9]:

(25) $\underbrace{q :_A \mathit{Mng} \quad H :_A \mathit{HScene} \quad s_H :_A q.\text{bg}}_{:_A \textsc{Answer}(A, q.\text{clfr}(s_H))!}$

[9]The notation $:_A T!$ means that A creates an object of type T, see Cooper (2023). We are here making several simplifications for reasons of space, including allowing non-exhaustive answers, not requiring that q is conversationally relevant, and not defining the type QUESTION.

4.7 Building witness takes from H-data

Given a type T with labels $labels(T)$ and a H-scene H, we want to find all of the records s that are witness takes on H, i.e., for which $s : T_{\text{bg}}$ holds. Recall that each $h \in H$ has a field h.id such that h'.id $\neq h$.id for any $h' \neq h$.

We define a function BUILDTAKES that uses tree search to build up list of situation takes respecting T. As input, takes a situation type, T , an instance of H-data, H, and a partial label map $\mathcal{L} : \{\ell \in labels(T) \mid T.\ell : \mathit{Ind}\} \rightharpoonup \{h.id \mid h \in H\}$. Running BUILDTAKES with an empty inital label map produces the set of all takes on H (records) that instantiate the given type.

The algorithm itself (see Algorithm 1) is not particularly important. Indeed, naive tree search is probably not cognitively plausible since the complexity scales exponentially with the number of objects in the scene, and there are many ways to make the search more efficient. What is of note here is the relationship between the H-data and the candidate situation takes, which are being built-up and structured according to the interpretation of the question. To satisfy the pre-conditions of the question (i.e., to find a witness for q.bg and candidate argument to q.clfr), it is not enough to use generic perceptual input; instead we must build up (or structure) a *take* on the situation based on high-level perceptual data and the question itself. Only then can the situation take be used to answer the question.

5 Adapting the general account to the CLEVR dataset

Everything we have said above is independent of the type of H-data. In this section, we will show a concrete instance of this general framework, where H-data is a set of CLEVR scene graphs. This is intended as an illustration and proof-of-concept only, showing an example of how the general TTR account applies to a specific type of H-data and classification method.

5.1 The CLEVR dataset

CLEVR (Johnson et al., 2016) is a synthetic dataset of programatically rendered 3d scenes of geometric objects and template-generated questions about them. Designed as a diagnostic dataset for visual and linguistic compositional reasoning, it is drawn from an extremely constrained visual and linguistic domain domain. Objects are drawn from limited

Algorithm 1 Recursive tree search for finding situation takes that satisfy a type from given H-data. The function EXTRACTTAKE produces a record that represents all objects in H, labeled according to $\mathcal{L}$. Objects from H that don't appear in the image of $\mathcal{L}$ are labeled with auxiliary labels that don't appear in T.

Require: $\mathcal{L} : \{\ell \in labels(T) \mid T.\ell : Ind\} \rightharpoonup \{h.id \mid h \in H\}$

function BUILDTAKES($T, H, \mathcal{L}$)
  $U \leftarrow labels(T) \setminus Dom(\mathcal{L})$
  $s \leftarrow$ EXTRACTTAKE($\mathcal{L}, H$)
  if $U = \varnothing$ **then**
    if $s : T$ **then**
      return $\{s\}$
    else
      return $\varnothing$
  else
    if $s : T \restriction Dom(\mathcal{L})$ **then**
      $l \leftarrow Pop(U)$
      $\mathcal{L}'_h \leftarrow \mathcal{L} \cup \{\langle l, h.\mathrm{id}\rangle\}$
      return $\cup\{$BUILDTAKES$(T, H, \mathcal{L}'_h) \mid h \in H\}$
    else
      return $\varnothing$

number of geometric solids and have a finite range of colors, sizes, and textures. Because the images are generated from code, it is possible to use image metadata to design train/test splits that probe for compositional representation by testing how well a model generalizes to novel situations.

Since the dataset is programatically generated, it is possible to produce "ground truth" semantics for the questions and schematic descriptions of the image that are sufficient for answering the questions. In this work, we directly use these schematic question and scene descriptions as input to validate that the symbolic component of our system has the inferential and representational power required for the task. This ensures that the performance of an eventual hypbrid system is reflective of the distributed representations and their interaction with the symbolic components, and not theoretical limitations.

Ground truth scene representations are presented as a *scene graph*, whose nodes are objects, annotated with attributes like color and shape (one-place predicates), and whose edges represent binary spatial relations between pairs of objects. Questions are represented as a functional programs that operate on those graphs.

5.2 Using CLEVR for H-data

In general, classification of visual scenes may involve e.g. processing of visual information in deep neural networks. However, in CLEVR the visual scenes are already annotated with formal structures that we will exploit when defining the classifiers required. In effect, we will treat the annotations as H-data, despite not actually being the result of processing of L-data. The classifiers themselves will be rather trivial, and they are meant only as an illustration of how the general account can be adapted to a specific dataset.

Assume we have as a scene graph g, consisting of a list of object descriptions that in turn consist of feature-value pairs. The original CLEVR data relies on the order of the items. To avoid this, we massage the data so that it includes a field `'id'` to avoid reliance on order:

```
[{'id': 'h0',
  '3d_coords': [2.408, -2.868, 0.350],
  'color': 'green',
  'material': 'metal',
  'pixel_coords': [213, 257, 7.728],
  'rotation': 233.806,
  'shape': 'cylinder',
  'size': 'small'
  'behind': ['h1', 'h2', 'h3',...],
  'front': [ ],
  'left': ['h2', 'h3', ...],
  'right': ['h5', ...]},
 {'id:'h1',
```

```
  '3d_coords': [1.649, -1.450, 0.350],
  'color': 'yellow',
  'material': 'metal',
  'pixel_coords': [246, 201, 9.081],
  'rotation': 140.802,
  'shape': 'cube',
  'size': 'small'
  'behind': ['h2', 'h3', ...],
  'front': ['h0'],
  'left': ['h2', 'h3', ...],
  'right': ['h5', ...]},
]
```

We can transform g into H-data H_s:

(26) H_s=[[id =a, data= {'id': 'h0, '3d_coords':...}], [id =b, data={'id': 'h1', '3d_coords':...}]]

From H_s and a background type T_{bg}, we can construct a witness take as described above in Sections 4.6 and 4.7.

As mentioned above, we are using H-data as evidence/proofs of types, and judging if some H-data is of a ptype is done using classifiers. In the case where H-data are scene graphs, the classification becomes rather trivial as the required information is already available in the H-data. For example, a colour classifier might be specified as follows:

(27) $\kappa_{\text{col}} = \lambda h : \langle Hdata \rangle$.
$\begin{cases} \text{RED if } h(1)\texttt{['color'] == 'red'} \\ \text{BLUE if } h(1)\texttt{['color'] == 'blue'} \\ \ldots \end{cases}$

For a two-place predicate:

(28) $\kappa_{\text{col}} = \lambda h : \langle Hdata, Hdata \rangle$.
$\begin{cases} \text{LEFT if } h(2)\texttt{.['id'] in } h(1)\texttt{['left']} \\ \text{RIGHT if } h(2)\texttt{.['id'] in } h(1)\texttt{['right']} \end{cases}$

Note that we are here mixing TTR notation with `Python code`. This is specific to the type of H-data we are using and a particular way of accessing it. For other types of H-data, the classifier definition may look quite different. However, we have set things up so that the dependence on H-data type has been isolated to the classifier definitions.

6 Summary and future work

We outlined a formal account of VQA where an agent perceives a real world situation by receiving low-level perceptual data, processes that data into high-level individuated perceptual data, structures this data in light of the semantics of a linguistic expression (a question), and then classifies the structured data to arrive at an answer to the question. Importantly (although not directly in scope of this paper), word meanings in general, including perceptual meaning aspects modeled as classifiers, are dynamic and subject to learning and coordination between dialogue participants (Larsson, 2015; Larsson et al., 2021; Larsson, 2021).

In future work, we aim to apply and extend the account to also encompass semantic pointers (Eliasmith, 2015; Larsson et al., 2025).

7 Limitations

There are many ways in which the work presented here could be extended. We only provide examples for the CLEVR dataset, but to concretely show the general applicability of the proposed account, we want to provide examples a large and diverse range of datasets. Furthermore, whereas the current examples focus on individual object classification, we would like to offer a more detailed explanation of how classification of images in terms of complex spatial relations. Another limitation is that the account does not yet cover classification of image sequences in terms of events that are extended in time. Finally, classification is in general probabilistic, and we would like to employ probabilistic TTR (Larsson, 2020a)to extend the current account accordingly.

Acknowledgments

This work was supported by grants from the Swedish Research Council VR project 2025-04592, Semantic Pointers in Natural Language Semantics (SPiNLS) and VR project 2014-39 for the establishment of the Centre for Linguistic Theory and Studies in Probability (CLASP) at the University of Gothenburg.

References

Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. 2016. Analyzing the Behavior of Visual Question Answering Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1955–1960, Austin, Texas. Association for Computational Linguistics.

Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. 2015. VQA: Visual Question Answering. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2425–2433, Santiago, Chile. IEEE.

Daisuke Bekki, Ribeka Tanaka, and Yuta Takahashi. 2021. Integrating Deep Neural Network with Dependent Type Semantics. In *Proceedings of the Symposium Logic and Algorithms in Computational Linguistics 2021 (LACompLing2021) 2021 (LACompLing2021)*, page 37, Stockholm University.

Ellen Breitholtz. 2020. *Enthymemes and Topoi in Dialogue: The Use of Common Sense Reasoning in Conversation*, volume 41 of *Current Research in the Semantics/Pragmatics Interface*. Brill, Leiden/Boston.

Robin Cooper. 2005. Austinian truth, attitudes and type theory. *Research on Language and Computation*, 3:333–362.

Robin Cooper. 2012. Type theory and semantics in flux. In Ruth Kempson, Nicholas Asher, and Tim Fernando, editors, *Handbook of the Philosophy of Science*, volume 14: Philosophy of Linguistics. Elsevier BV. General editors: Dov M. Gabbay, Paul Thagard and John Woods.

Robin Cooper. 2013. Clarification and Generalized Quantifiers. *Dialogue and Discourse*, 4(1):1–25.

Robin Cooper. 2023. *From Perception to Communication: a Theory of Types for Action and Meaning*. Oxford University Press.

Robin Cooper and Jonathan Ginzburg. 2015. Type theory with records for natural language semantics. *The handbook of contemporary semantic theory*, pages 375–407.

Simon Dobnik and Robin Cooper. 2017. Interfacing language, spatial perception and cognition in type theory with records. *Journal of Language Modelling*, 5(2):273–301.

Chris Eliasmith. 2015. *How to Build a Brain: A Neural Architecture for Biological Cognition*. Oxford Series on Cognitive Models and Architectures. Oxford Univ. Press.

Yash Goyal, Tejas Khot, Aishwarya Agrawal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2019. Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering. *Int. J. Comput. Vision*, 127(4):398–414.

Drew A. Hudson and Christopher D. Manning. 2018. Compositional Attention Networks for Machine Reasoning. In *ICLR*. arXiv. ArXiv:1803.03067 [cs.AI].

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. 2016. CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. *arXiv preprint*. ArXiv:1612.06890 [cs].

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Judy Hoffman, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. 2017. Inferring and Executing Programs for Visual Reasoning. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 3008–3017. ISSN: 2380-7504.

Staffan Larsson. 2011. The ttr perceptron: Dynamic perceptual meanings and semantic coordination. In *Proceedings of the 15th Workshop on the Semantics and Pragmatics of Dialogue (SemDial 2011)*, Los Angeles (USA). Institute for Creative Technologies.

Staffan Larsson. 2013. Formal semantics for perceptual classification. *Journal of Logic and Computation*.

Staffan Larsson. 2015. Formal semantics for perceptual classification. *Journal of Logic and Computation*, 25(2):335–369. Published online 2013-12-18.

Staffan Larsson. 2018. Perceptual semantics and dialogue processing. In *Proceedings of the Workshop on Dialogue and Perception*.

Staffan Larsson. 2020a. Discrete and probabilistic classifier-based semantics. In *Proceedings of the Probability and Meaning Conference (PaM 2020)*, pages 62–68, Gothenburg. Association for Computational Linguistics.

Staffan Larsson. 2020b. Extensions are indeterminate if intensions are classifiers. In *Proceedings of the 24th Workshop on the Semantics and Pragmatics of Dialogue–Full Papers, Virtually at Brandeis, Waltham, New Jersey. SEMDIAL.*

Staffan Larsson. 2021. The role of definitions in coordinating on perceptual meanings. In *Proceedings of the Workshop on the Semantics and Pragmatics of Dialogue (SemDial 2021)*.

Staffan Larsson, Jean-Philippe Bernardy, and Robin Cooper. 2021. Semantic learning in a probabilistic type theory with records. In *Proceedings of Workshop on Computing Semantics with Types, Frames and Related Structures 2021*.

Staffan Larsson and Robin Cooper. 2009. Towards a formal view of corrective feedback. In *Proceedings of the EACL 2009 Workshop on Cognitive Aspects of Computational Language Acquisition*.

Staffan Larsson and Robin Cooper. 2021. Bayesian classification and inference in a probabilistic type theory with records. In *Proceedings of NALOMA 2021*.

Staffan Larsson, Jonathan Ginzburg, Robin Cooper, and Andy Lücking. 2025. Finding answers to questions: Bridging between type-based and computational neuroscience approaches. In *Proceedings of the 16th international conference on computational semantics*, pages 118–126.

Yinhong Liu and Guy Emerson. 2022. Learning Functional Distributional Semantics with Visual Data. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3976–3988, Dublin, Ireland. Association for Computational Linguistics.

Andy Lücking and Jonathan Ginzburg. 2022. Referential transparency as the proper treatment for quantification. *Semantics and Pragmatics*, 15(4).

Vladislav Maraev, Ellen Breitholtz, Christine Howes, Staffan Larsson, and Robin Cooper. 2021. Something Old, Something New, Something Borrowed, Something Taboo: Interaction and Creativity in Humour. *Frontiers in Psychology*, 12.

Arild Matsson, Simon Dobnik, and Staffan Larsson. 2019. Imagettr: Grounding type theory with records in image classification for visual question answering. In *Proceedings of the IWCS 2019 Workshop on Computing Semantics with Types, Frames and Related Structures*, pages 55–64.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

Ronja Utescher. 2019. Visual ttr-modelling visual question answering in type theory with records. In *Proceedings of the 13th International Conference on Computational Semantics-Student Papers*, pages 9–14.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Josh Tenenbaum. 2018. Neural-Symbolic VQA: Disentangling Reasoning from Vision and Language Understanding. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Peng Zhang, Yash Goyal, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2016. Yin and Yang: Balancing and Answering Binary Visual Questions. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5014–5022. ISSN: 1063-6919.

Semantic Distance for Presburger Formula Generation

Masaya Taniguchi
RIKEN
masaya.taniguchi@riken.jp

Hitomi Yanaka
The University of Tokyo
RIKEN
Tohoku University
hyanaka@is.s.u-tokyo.ac.jp

Abstract

We study how to evaluate and train natural-language-to-formula generation when surface similarity is a poor proxy for semantic correctness. Focusing on translation into Presburger arithmetic, we introduce a geometric distance between formulas by viewing each formula as a set of integer lattice points and assigning an exponentially decaying weight to that set. The resulting distance yields finite comparisons even for infinite definable sets, can be computed through quantifier elimination, polyhedral decomposition, and lattice-point generating functions, and inherits the standard metric properties. We then use this distance in experiments on supervised fine-tuning (SFT) and Group Relative Policy Optimization (GRPO) for translating arithmetic statements into formal formulas. The results show that a distance-aware reward substantially improves parsability and adjusted semantic quality compared with a string-similarity-only reward, while also revealing the remaining challenge of preserving quantifier structure.

1 Introduction

Natural language models are increasingly asked to map descriptions into formal logical representations, from semantic parsing (Zettlemoyer and Collins, 2005; Liang et al., 2011; Dong and Lapata, 2016; Herzig et al., 2021) to program synthesis (Yin and Neubig, 2017). In these tasks, evaluating predictions against reference formulas is fundamentally difficult: string-based metrics such as BLEU, ROUGE, or exact match fail to capture semantic correctness. Two formulas can differ by a single character yet define entirely different solution sets, while a near-identical string can differ only in punctuation and preserve meaning. This gap motivates metrics that operate on the meaning of formulas rather than their surface form. Such metrics are especially important when small syntactic changes alter the space of valid numerical assignments.

Example 1 (Motivating example)**. Statement**: Alice has at least 6 apples, which is at least as many as Bob has oranges.
Reference: $x \geq 6 \wedge x \geq y$ (Alice has x apples, Bob has y oranges)
Prediction: $x \geq 6 \wedge x > y$
String similarity: ≈ 1 (single character difference)
Geometric distance: $d_\beta > 0$ (strict subset)
The prediction excludes valid boundary points $x = y$. Both surface similarity and semantic distance are needed to evaluate models faithfully.

While this paper situates itself within the broader context of semantic parsing, we explicitly restrict our focus to Presburger arithmetic; *i.e.*, linear integer arithmetic with addition and order. This domain serves as a well-defined and rigorous case study for evaluating semantics-aware rewards. We emphasize that Presburger arithmetic excludes multiplication between variables, polynomial equations, integrals, or higher-order logic commonly found in open-domain scientific texts. Generalizing our geometric distance metric to such richer formal languages remains highly non-trivial due to the lack of efficient quantifier elimination or tractable lattice-point counting methods; thus, this work represents a foundational first step rather than a universal solution.

The choice of Presburger arithmetic is deliberate. It is expressive enough to represent many elementary linear arithmetic statements, while remaining sufficiently well behaved for quantifier elimination and lattice-point counting. This makes it a suitable controlled testbed for studying semantics-aware rewards. At the same time, the restriction is essential: our method should be understood as a first step for linear integer arithmetic rather than as a general-purpose semantic distance for arbitrary mathematical language.

We address this by defining a **weighted geomet-**

Proceedings of the 6th Workshop on Natural Language Meets Logic and Machine Learning (NALOMA), pages 71–79

ric distance on Presburger-definable sets, interpreting each formula as the set of integer lattice points it entails. For instance, $x > 2$ and $x > 0$ share many lattice points, while $x > 1$ shares more with $x > 0$ than $0 > x$ does. The distance quantifies exactly this degree of overlap using an exponentially decaying weight. This yields a finite, semantics-aware comparison even for infinite sets, bridging the gap between strict symbolic equivalence (distance zero) and unconstrained string similarity (distance near one).

We show this distance is computable by combining *quantifier elimination* (Theorem 1), *polyhedral decomposition* (Section 3), and *lattice-point generating functions* (Section 3; Loera et al., 2004), classical tools from logic and discrete geometry. In experiments on natural-language-to-formula translation with supervised fine-tuning (SFT) and Group Relative Policy Optimization (GRPO), a distance-aware reward improves parsability from 76.5% to 98.6% and reduces semantic distance from 0.295 to 0.167 compared to a string-similarity-only baseline.

In summary, this paper makes three contributions:

1. We define a weighted geometric distance on Presburger-definable sets and show that it inherits the standard metric properties of weighted Jaccard distances.

2. We present a practical computation pipeline using quantifier elimination and polyhedral decomposition, establishing tractable cases.

3. We demonstrate through SFT and GRPO experiments that our distance reward substantially improves parsability and semantic quality, while critically discussing its limitations regarding synthetic datasets, quantifier preservation, and domain generalizability.

2 Preliminaries

Definition 1 (Presburger arithmetic)**.** *Presburger arithmetic* is the first-order theory of integers equipped with addition, order, constants, logical connectives, and quantifiers (Cooper, 1972; **?**). Multiplication between variables is excluded. Formulas are constructed from atomic constraints of the form

$$a_1x_1+\cdots+a_dx_d \le b, \qquad a_1x_1+\cdots+a_dx_d = b,$$

where the coefficients a_i and the constant b are integers. Composite formulas are formed using the Boolean connectives $\wedge$, $\vee$, $\neg$, and the quantifiers $\forall$, $\exists$.

One of the key facts underlying our method is that Presburger arithmetic is *decidable* and, moreover, admits *quantifier elimination* (Cooper, 1972; **?**). Therefore, any logical formula can be transformed into a quantifier-free description. Geometrically, such definable sets can be understood as finite unions of regions cut out by linear constraints on the integer lattice. This observation allows one to pass from logical formulas to polyhedral pieces amenable to geometric analysis.

Theorem 1 (Quantifier elimination for Presburger arithmetic)**.** *For any Presburger formula $\varphi(\mathbf{x})$, there exists a quantifier-free formula $\psi(\mathbf{x})$ that defines the same subset of $\mathbf{Z}^d$ as φ. Moreover, ψ can be written as a finite Boolean combination of linear inequalities, linear equalities, and congruence conditions of the form $a_1x_1 + \cdots + a_dx_d \equiv c \pmod m$.*

This theorem is the bridge connecting logic and geometry in our study. Once all quantifiers are eliminated, the resulting formula can be decomposed into finitely many linear regions and periodic conditions.

Example 2 (Quantifier elimination)**.** Consider the formula

$$\exists z\,(x = 2z + 1 \wedge 0 \le z \wedge z \le 2).$$

The conjuncts $0 \le z \wedge z \le 2$ bound the auxiliary variable z, while $x = 2z + 1$ links x to z. Eliminating z removes the dependence on the internal variable and yields an equivalent quantifier-free formula:

$$x = 1 \vee x = 3 \vee x = 5.$$

The same solution set can additionally be expressed compactly as

$$1 \le x \le 5 \wedge x \equiv 1 \pmod 2.$$

This illustrates the general theorem: bounding constraints on quantified variables become explicit threshold conditions, while linear dependencies decompose into congruence conditions on the remaining free variables.

We identify each formula $\varphi(\mathbf{x})$ with its model set $S_\varphi = \{\mathbf{x} \in \mathbf{Z}^d \mid \varphi(\mathbf{x})\}$, so logical connectives correspond to set operations: $\wedge$ to $\cap$, $\vee$ to $\cup$, and $\neg$ to set complementation.

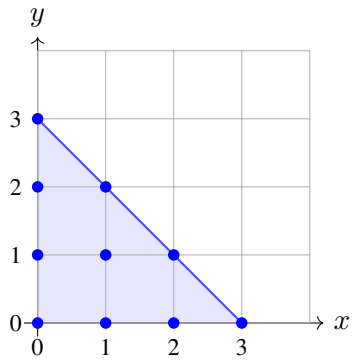


Figure 1: Integer solutions in the two-dimensional plane satisfying the Presburger formula $x + y \leq 3 \wedge x \geq 0 \wedge y \geq 0$.

Example 3 (Basic Presburger formulas)**.** The formula $x + y \leq 3 \wedge x \geq 0 \wedge y \geq 0$ defines the lattice points in a two-dimensional simplex (Figure 1). The formula $\exists z\,(x = 2z)$ defines the set of all even numbers.

On top of this representation, the distance in the next section compares two formulas through the difference in weighted mass of their model sets.

3 Geometric Distance and Metricity

In this section, we define a **weighted Jaccard distance** on Presburger-definable sets, viewing each formula as a set of integer lattice points with exponentially decaying weights. The construction generalizes the standard Jaccard index (union and intersection) to a weighted measure, yielding a finite-size-aware comparison even for infinite sets.

Definition 2 (Weighted Presburger measure)**.** Fix $\beta > 0$, and let $\varphi(\mathbf{x})$ be a Presburger formula over $\mathbf{x} = (x_1, \ldots, x_d) \in \mathbf{Z}^d$. Then the weighted measure of φ is defined by

$$\mu_\beta(\varphi) = \frac{1}{Z_{\beta,d}} \sum_{\mathbf{x} \in S_\varphi} e^{-\beta \|\mathbf{x}\|_1}$$

where the normalization constant is

$$Z_{\beta,d} = \left(\frac{1 + e^{-\beta}}{1 - e^{-\beta}} \right)^d.$$

These definitions yield a probability mass function on $\mathbf{Z}^d$, and the following theorem confirms it is well-defined.

Theorem 2 (Well-definedness and boundedness)**.** *For any Presburger formula φ and any $\beta > 0$, $\mu_\beta(\varphi)$ is well-defined and satisfies $0 \leq \mu_\beta(\varphi) \leq 1$.*

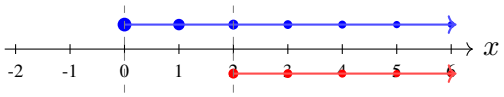


Figure 2: Threshold formulas $p(x)$ and $q(x)$. Points 0 and 1 dominate the distance.

Proof. Every weight $e^{-\beta\|\mathbf{x}\|_1}$ is nonnegative, so $\mu_\beta(\varphi) \geq 0$. The set S_φ is a subset of the full lattice $\mathbf{Z}^d$, so its weighted sum cannot exceed the total mass $Z_{\beta,d}$. Dividing by $Z_{\beta,d}$ therefore yields a value in $[0, 1]$. □

The main computational challenge is to evaluate this weighted sum for an arbitrary Presburger formula. Quantifier elimination transforms the formula into a quantifier-free description, which decomposes into a finite family of rational polyhedra:

$$S_\varphi = \bigsqcup_{k=1}^{m} (P_k \cap \mathbf{Z}^d).$$

Each P_k is described by linear equalities and (strict or non-strict) inequalities. Since $\|\mathbf{x}\|_1$ is piecewise linear, we partition space into 2^d orthants indexed by sign vectors $\mathbf{s} \in \{\pm 1\}^d$ and apply a diagonal transformation $\mathbf{y} = \mathrm{diag}(\mathbf{s})\mathbf{x}$ to map each orthant into the nonnegative quadrant, where $\|\mathbf{x}\|_1$ becomes a linear sum. This yields

$$\mu_\beta(\varphi) = \frac{1}{Z_{\beta,d}} \sum_{k=1}^{m} \sum_{\mathbf{x} \in P_k \cap \mathbf{Z}^d} e^{-\beta\|\mathbf{x}\|_1}.$$

The exponential orthant partition 2^d is tractable for small d but may become prohibitive as dimension grows (see Section 6).

Example 4 (One-dimensional threshold formulas)**.** Let $d = 1$, and let $p(x)$ be the formula $x \geq 0$ and $q(x)$ the formula $x \geq 2$. Then $S_p = \{0, 1, 2, \ldots\}$ and $S_q = \{2, 3, 4, \ldots\}$. In one dimension, $Z_{\beta,1} = \frac{1+e^{-\beta}}{1-e^{-\beta}}$, so

$$\mu_\beta(p) = \frac{\sum_{n \geq 0} e^{-\beta n}}{Z_{\beta,1}}, \quad \mu_\beta(q) = \frac{\sum_{n \geq 2} e^{-\beta n}}{Z_{\beta,1}}.$$

Since $p \wedge q$ is equivalent to q and $p \vee q$ is equivalent to p, the distance is $d_\beta(p, q) = 1 - e^{-2\beta}$. In our experiments Section 4 we use $\beta = 1$.

After mapping into the nonnegative orthant, the sum can be computed with lattice-point generating functions (Ehrhart theory) (Woods, 2014). Let $Q \subseteq \mathbf{R}^d_{\geq 0}$ be a polyhedron and define $G_Q(\mathbf{t}) = \sum_{\mathbf{y} \in Q \cap \mathbf{Z}^d} t_1^{y_1} \cdots t_d^{y_d}$. Barvinok methods represent

G_Q as a rational function, evaluated at $t_i = e^{-\beta}$ to yield the required weighted count. The Barvinok representation is a compressed "weighted counting table": each monomial encodes a lattice point, and the rational-function form allows efficient comparison of $G_{p\wedge q}$ and $G_{p\vee q}$, making the distance a Jaccard-type overlap measure with emphasis near the origin.

We also require that basic order relations and triangle-inequality-type estimates hold in this weighted setting. Since every lattice-point weight is strictly positive, $A \subseteq B$ implies $\mu_\beta(A) \leq \mu_\beta(B)$, and $A \subseteq B \cup C$ implies $\mu_\beta(A) \leq \mu_\beta(B) + \mu_\beta(C)$. These monotonicity and subadditivity properties underpin the triangle inequality for $\delta_\beta(p, q) = \mu_\beta(p\triangle q)$.

Finally, the distance between two formulas p and q is defined as the Jaccard-type ratio with respect to the normalized weighted measure:

$$d_\beta(p, q) = 1 - \frac{\mu_\beta(p \wedge q)}{\mu_\beta(p \vee q)} = \frac{\mu_\beta(p\triangle q)}{\mu_\beta(p \vee q)},$$

where $p\triangle q = (p \wedge \neg q) \vee (q \wedge \neg p)$. When $\mu_\beta(p \vee q) = 0$ (both formulas define the empty set), we set $d_\beta(p, q) = 0$. This is a discrete weighted version of the classical Jaccard distance (Jaccard, 1901): larger shared probability mass implies closer formulas. The parameter β controls locality: large β concentrates mass near the origin, while small β gives more weight to distant points.

To verify that d_β defines a metric, we use symmetric difference.

Lemma 1 (Basic identities for μ_β)**.** *For Presburger formulas p and q, the measure μ_β satisfies:*

1. $\mu_\beta(p) \geq 0$.
2. $\mu_\beta(p \vee q) = \mu_\beta(p) + \mu_\beta(q) - \mu_\beta(p \wedge q)$.
3. $\mu_\beta(p\triangle q) = \mu_\beta(p \vee q) - \mu_\beta(p \wedge q)$.

Proof. Point (1) holds because all weights are nonnegative. For (2), adding the weights of p and q counts the overlap $p \wedge q$ twice, so subtracting it once gives the correct weight of $p \vee q$. For (3), the symmetric difference contains exactly those points that belong to p or q but not both, i.e., the union with the overlap removed, so its weight equals $\mu_\beta(p \vee q) - \mu_\beta(p \wedge q)$. □

Lemma 2 (Positivity and symmetric-difference representation)**.** *For Presburger formulas p and q, if $\mu_\beta(p \vee q) > 0$, then*

$$d_\beta(p, q) = \frac{\mu_\beta(p\triangle q)}{\mu_\beta(p \vee q)}$$

holds. Moreover, if p and q define different subsets of $\mathbf{Z}^d$, then $d_\beta(p, q) > 0$.

Proof. The first claim follows directly from the previous lemma: $\mu_\beta(p \vee q) - \mu_\beta(p \wedge q) = \mu_\beta(p\triangle q)$, so dividing by $\mu_\beta(p \vee q)$ gives the claimed representation.

For positivity: if p and q define different sets, then their symmetric difference contains at least one lattice point $\mathbf{a}$. Since every lattice point has strictly positive weight, a single such contribution already implies $\mu_\beta(p\triangle q) > 0$, and hence $d_\beta(p, q) > 0$. □

Theorem 3 (Metricity of the Presburger distance)**.** *Fix $\beta > 0$. Then the function d_β defines a metric on the family of Presburger-definable subsets of $\mathbf{Z}^d$.*

Proof. The distance d_β is the weighted Jaccard distance induced by the finite positive measure μ_β on $\mathbb{Z}^d$. Hence non-negativity, symmetry, and the triangle inequality follow from the standard measure-theoretic Jaccard metric argument.[1] The identity axiom also holds because every lattice point has strictly positive weight: $\mu_\beta(A\triangle B) = 0$ implies $A = B$. Therefore d_β is a metric on Presburger-definable subsets of $\mathbb{Z}^d$. □

4 Presburger Formula Translation Experiments

This section evaluates GRPO fine-tuning for translating natural-language Presburger arithmetic problems into formal formulas. We follow the GRPO-style reinforcement-learning setup introduced in recent mathematical reasoning work on open language models (Shao et al., 2024). All runs use `Qwen/Qwen3-4B` (Yang et al., 2025) and a fixed split: `data/train.jsonl` for training, `data/dev.jsonl` for validation during SFT, and `data/test.jsonl` for final evaluation. The test set contains 2,000 examples. Code for the experiments is available at https://github.com/tani/presburger-distance-naloma2026.

[1] The standard argument uses $A\triangle C \subseteq (A\triangle B) \cup (B\triangle C)$ together with the usual normalization by union mass; equivalently, this is the classical Jaccard metric proof for finite measures.

Parameter	Value
base model	`Qwen/Qwen3-4B`
SFT epochs	1
SFT max steps	120
SFT batch size	32
SFT gradient accumulation	2
SFT learning rate	`1.5e-5`
max sequence length	512
SFT save steps	20
SFT logging steps	10

Table 1: Shared SFT configuration.

Run	Reward	Steps	Batch	Gens
`string_similarity`	sim-only	100	32	16
`presburger_distance`	dist-pen.	100	32	16

Table 2: GRPO configurations. Both runs use learning rate `1.0e-5`.

4.1 Experimental Setup

All runs first perform supervised fine-tuning (SFT), following the now-standard instruction-tuning stage used before preference optimization and reinforcement-learning-based alignment, and then GRPO. The shared SFT configuration is shown in Table 1. We compare two GRPO reward functions under the same training budget: `string_similarity` with `similarity_only` and `presburger_distance` with `distance_penalized`. Both GRPO runs use 100 steps, batch size 32, 16 generations per prompt, learning rate `1.0e-5`, `temperature = 0.8`, `max_completion_length = 128`, `distance_beta = 1.0`, and vLLM inference.

The full pipeline is straightforward: we first train an SFT model, merge the resulting LoRA adapter into the base model, run GRPO from that SFT-initialized checkpoint, and then evaluate the base, SFT, and GRPO models on the same test set. For each output, we compute parsability, exact match, equivalence when available, string similarity, Presburger distance, and adjusted distance.

4.2 Dataset Format and Pipeline

Each dataset item contains a natural-language arithmetic problem and a target Presburger formula. The model is trained to map the `problem` field to the `formula` field. A typical JSONL entry has the form shown below.

```
{"uuid7": "...",
 "problem": "...",
 "formula": "..."}
```

The training data are synthetic. We first generate logical formulas and propositions by a random-walk-like procedure over tree-structured formula templates, including atomic linear constraints, Boolean connectives, and quantifier prefixes. We then ask a generative language model to produce the corresponding natural-language mathematical problem for each target formula. The prompt used for this problem-generation step is included in `src/dataset.py` in the anonymous repository: https://anonymous.4open.science/r/presburger-distance-1348/src/dataset.py.

Starting from the shared train/dev/test split, we first perform SFT, merge the adapter into the base model, run GRPO with one of the two reward definitions, and then evaluate the base, SFT, and GRPO models on the same held-out test set. This pipeline separates format learning, reward-based optimization, and semantic evaluation.

Illustrative training examples. The task itself ranges from direct linear inequalities to quantified formulas. For example, a simple atomic case maps the sentence "the sum $8x + 6z$ is less than -2" to the target formula `8 x + 6 z < -2`. A logical-equivalence case maps "true exactly when" to `<==>`, for example `[-5 x + 8 z = 8 <==> -6 x + 9 y + 8 z < 6]`. A quantified case requires preserving a prefix such as `(A x)(E y)` in addition to the bracketed matrix formula. These examples already suggest why exact string accuracy, syntactic validity, and semantic distance capture different aspects of model behavior.

4.3 Reward and Evaluation Metrics

The `similarity_only` reward is the normalized string similarity between the generated formula g and the reference formula r:

$$R_{\text{sim}}(g, r) = \text{SeqMatch}(g, r).$$

The `distance_penalized` reward gives invalid outputs a reward of -1.0 and otherwise combines semantic distance with string similarity:

$$R_{\text{dist}}(g, r) = 0.5 \cdot (1 - D(g, r)) + 0.5 \cdot \text{SeqMatch}(g, r).$$

If distance computation times out, the distance-derived component is set to 0.0.

We use a mixed reward rather than a distance-only reward because the semantic distance component is informative only after the output is parsable and the distance computation succeeds. In early

GRPO stages, a pure-distance reward would assign little useful signal to invalid strings. The string-similarity term therefore acts as a syntactic shaping signal, while the distance term adds semantic pressure once formal outputs become well formed. A pure-distance ablation remains an important direction for future work.

The Presburger distance is

$$D(p, q) = 1 - \frac{\mu(p \wedge q)}{\mu(p \vee q)}.$$

Lower values are better. Because raw mean distance is available only for parsable outputs whose distance can be computed, we also report the adjusted distance

$$D_{\text{adj}}(g, r) = \begin{cases} D(g, r) & \text{if computable,} \\ 1.0 & \text{otherwise.} \end{cases}$$

This metric prevents models with many invalid outputs from appearing artificially strong.

4.4 Main Results

Table 3 shows that semantic feedback changes GRPO behavior: the strongest overall result is obtained by the distance-based GRPO run `presburger_distance`, which achieves the best adjusted distance (0.1667) and the highest parsability (98.55%). Its raw distance is worse than that of `string_similarity`, but this is misleading because `string_similarity` leaves many outputs unparsable and therefore excludes them from the raw-distance average.

Relative to SFT, GRPO with `similarity_only` substantially degrades performance: parsability drops from 87.70% to 76.50%, exact/equivalent accuracy drops from 68.45% to 47.90%, and adjusted distance worsens from 0.1778 to 0.2953. In contrast, GRPO with `distance_penalized` improves parsability from 87.45% to 98.55% and improves adjusted distance from 0.1821 to 0.1667, although it still reduces exact/equivalent accuracy from 67.30% to 54.05%. This shows that the distance-based reward mainly improves formal validity rather than exact reproduction of the reference string.

4.5 Direct Comparison and Error Analysis

A direct comparison over the same 2,000 test cases shows where this feedback takes effect. The `presburger_distance` model produces 191 exact outputs that `string_similarity` misses, whereas `string_similarity` uniquely gets only 68 exact. Likewise, `presburger_distance` is uniquely parsable on 448 examples, while `string_similarity` is uniquely parsable on only 7. Under adjusted distance, `presburger_distance` is better on 474 examples, `string_similarity` is better on 62, and 1,464 examples are tied. These results are summarized in Table 4.

The dominant failure mode is quantifier handling, but the two rewards fail differently. Among the 611 quantified references, `string_similarity` often preserves quantifier tokens in invalid bracket syntax: 336 outputs contain fragments such as `[A x]` or `[E y]`, despite high mean string similarity (0.902). By contrast, `presburger_distance` almost eliminates this syntax error, but usually by deleting the prefix: it drops all quantifiers on 597 quantified-reference examples. Thus the distance reward improves parsability but still loses exact formula structure. Even when SFT is exactly correct, `string_similarity` creates 187 invalid bracketed-quantifier outputs, whereas `presburger_distance` creates 309 parsable but quantifier-free outputs.

4.6 Representative Examples

Individual errors make the semantic-feedback effect visible. In one representative case, the SFT model outputs `(3 x - 4 y - 5 z >= 9 \/ 9 x + 5 y - 5 z /= -7)`, which is close to the reference string but unparsable because the grammar expects brackets rather than parentheses. The GRPO model with `distance_penalized` instead produces the exact reference `[3 x - 4 y - 5 z >= 9 \/ 9 x + 5 y - 5 z /= -7]`, so its adjusted distance becomes 0.0 while the SFT output receives the worst-case adjusted distance.

However, the opposite phenomenon also appears. For `(A x)(E y)[-x + 5 z <= 7 <==> 2 y - 3 z = -6]`, `similarity_only` produces the unparsable `[A x][E y][5 z - x <= 7 <==> 2 y - 3 z = -6]`, while `distance_penalized` produces the parsable but quantifier-free `[5 z - x <= 7 <==> 2 y - 3 z = -6]`. These examples show why both parsability and semantic distance are necessary for evaluation.

Presburger arithmetic admits quantifier elimination, so some formulas with different surface quantifier structure may nevertheless induce closely related or even equivalent arithmetic constraints after

Run	Model	Parsable	Exact / Equivalent	String Similarity	Raw Distance	Adjusted Distance
string_similarity	Base	9.90%	0.50%	0.3814	0.2115	0.9223
string_similarity	SFT	87.70%	68.45%	0.9561	0.0533	0.1778
string_similarity	GRPO	76.50%	47.90%	0.9410	0.0740	0.2953
presburger_distance	Base	9.90%	0.50%	0.3814	0.2111	0.9223
presburger_distance	SFT	87.45%	67.30%	0.9544	0.0555	0.1821
presburger_distance	GRPO	98.55%	54.05%	0.9382	0.1462	0.1667

Table 3: Main evaluation results.

Comparison	Count
both exact	890
only string_similarity exact	68
only presburger_distance exact	191
only string_similarity parsable	7
only presburger_distance parsable	448
string_similarity better by adjusted distance	62
presburger_distance better by adjusted distance	474
tied by adjusted distance	1464

Table 4: Head-to-head comparison between the two GRPO rewards.

Pattern	sim	dist
invalid bracketed quantifier after quantified reference	336	0
all quantifiers dropped after quantified reference	158	597
other invalid output after quantified reference	117	14
invalid GRPO output	470	29

Table 5: Most important error patterns for the two GRPO rewards.

elimination. Accordingly, the present metric prioritizes overlap of induced solution sets over exact preservation of the original logical surface form.

4.7 Additional Case Studies

It is also useful to contrast cases in which semantic correctness and syntactic correctness diverge. In one example, the reference is `-9 x + 2 z <= -4`. The `similarity_only` GRPO model outputs `[-9 x + 2 z <= -4]`, which remains parsable and semantically equivalent under the distance computation, but fails exact match because of the added outer brackets. The `distance_penalized` model instead reproduces the exact target string. This illustrates that the two rewards can produce semantically similar outputs while differing in how closely they match the annotation convention.

A second case shows the opposite failure mode. For the reference `[-10 x - 5 y = -5 <==> -9 x - 9 y >= -5]`, the `distance_penalized` model may produce `[-10 x + (-5 y) = -5 <==> -9 x + (-9 y) >= -5]`. Although this looks close to the target as a string, the inserted parenthesized negative terms are rejected by the grammar in the current evaluation pipeline. The example highlights that a reward which strongly discourages invalid outputs can still generate local syntactic forms that are outside the accepted grammar.

Taken together, these examples reinforce a central point of the experiments: the best reward is not simply the one that maximizes local string similarity, but the one that most reliably preserves formal well-formedness while staying semantically close to the target formula.

5 Discussion

5.1 Scope in the NLP Landscape

This work should be read as a controlled case study in semantics-aware formula generation, not as a general solution to semantic similarity for formal mathematical language. Presburger arithmetic is restricted to linear integer arithmetic with addition, order, Boolean connectives, and first-order quantifiers. It excludes multiplication between variables, polynomial equations, integrals, gradients, differential operators, and higher-order or dependent type-theoretic constructions.

This restriction is also what makes the proposed distance computationally meaningful in the present setting. Quantifier elimination, polyhedral decomposition, and lattice-point generating functions provide a principled route from formulas to weighted model sets. For richer formal languages, comparable semantic distances would require different computational tools, and in many cases no similarly tractable pipeline is available. Thus, the present metric is best understood as a foundational step toward semantics-aware rewards, rather than as a universal metric for mathematical NLP.

5.2 Synthetic Data and Pre-training Dependencies

A primary criticism of our evaluation might focus on its reliance on a synthetically generated dataset rather than noisy, real-world human text. However, this choice reflects a deliberate method-

ological decoupling rather than an oversight. The core objective of this study is not to benchmark the well-documented linguistic robustness of LLMs against typos, colloquialisms, or stylistic variants—confounding factors that are better evaluated in orthogonal studies. Instead, our goal is to rigorously isolate and quantify a single, precise causal relationship: whether mapping discrete arithmetic propositions into a continuous space via a geometric distance reward inherently guides policy optimization more effectively.

Introducing open-domain human noise at this foundational stage would introduce unnecessary variance, blurring the semantic signal of the reward and obscuring whether our geometric metric is genuinely driving the observed performance gains. Furthermore, given that Presburger arithmetic is restricted to linear transformations, solving open-domain offline natural language tasks would not immediately yield an end-to-end industrial application. Therefore, we maintain a cautious and principled stance, restricting our benchmark to controlled environments to ensure the precision and clarity of our theoretical contributions.

We also do not claim that the observed performance is independent of pre-training. The experiments start from a pretrained language model that already has substantial knowledge of mathematical notation and instruction-following conventions. Our claim is therefore narrower: given such a pretrained model, a semantic distance component provides a more effective optimization signal than string similarity alone. Disentangling the contribution of pre-training would require additional ablations, such as randomly initialized models, in-domain language-model pre-training, held-out vocabulary splits, or held-out lattice-region splits. We leave these evaluations to future work.

5.3 Computational Complexity and Scalability

Overall, GRPO responds to the semantic signal in the reward. A string-similarity-only reward does not reliably preserve syntactic validity and performs poorly under adjusted distance. The distance-based reward penalizes invalid formulas and shifts outputs toward the formal grammar, yielding much higher parsability and the best adjusted semantic distance.

At the same time, neither GRPO reward surpasses the SFT baseline on exact string accuracy. The remaining bottleneck is structural fidelity, especially quantifier preservation. Because the current distance computation is most useful for quantifier-free formulas, the reward does not strongly penalize missing prefixes such as `(A ...)` and `(E ...)`.

There is also an important computational trade-off behind the semantic evaluation itself. Computing Presburger distances through QEPCAD and counting lattice points with LattE is computationally expensive from a computer-science perspective, and the running time can grow substantially as the number of variables increases or the coefficients become larger, consistent with the broader difficulty of integer-programming-style reasoning and related optimization problems (Papadimitriou, 1981). In that sense, the present benchmark benefits from the fact that the test formulas are comparatively simple. This makes the distance-based pipeline practical in our experiments, but it also means that scalability to harder instances remains an important limitation and a topic for future work.

6 Limitations

This study has several limitations typical of semantics-aware evaluation in NLP. First, the benchmark formulas are comparatively simple, so the current results do not establish that the proposed distance remains practical for larger formulas with more variables, larger coefficients, or more deeply nested quantifier structure.

Second, our semantic pipeline depends on quantifier elimination and lattice-point computation, which are substantially more expensive than string-based metrics and may limit scalability during training.

Third, although the distance-aware reward improves parsability and adjusted semantic quality, it still does not adequately preserve quantifier prefixes. Thus, semantic similarity alone is not sufficient to guarantee full structural fidelity.

Fourth, the experiments focus on a single formal translation setting, and it remains to be tested how well the same reward design transfers to other symbolic generation tasks in NLP.

Fifth, the experiments do not isolate the contribution of the pretrained base model. The results therefore show the effect of reward design under a pretrained-model regime, not the intrinsic learnability of the task from scratch.

Finally, we did not include a pure-distance GRPO reward. The present comparison evaluates a mixed syntactic-semantic reward against a syntac-

tic baseline, leaving a pure semantic-reward ablation for future work.

7 Conclusion

Among the two GRPO reward functions, `distance_penalized` is preferable: its errors suggest that the semantic reward is being optimized, moving outputs away from invalid string-like forms toward parsable formulas. It substantially improves parsability, improves exact match relative to `similarity_only`, and achieves the best adjusted semantic distance. Nevertheless, SFT remains stronger for exact formula reproduction, so the next step is to design rewards that preserve quantifier structure as well as formal validity.

Acknowledgments

This work was supported by JSPS KAKENHI Grant-in-Aid for JSPS Fellows (Grant Number JP26KJ0414), JST CREST (Grant Number JPMJCR2565), and the JST BOOST Program (Grant Number JPMJBY24H5), Japan.

We used ChatGPT for limited coding assistance, LaTeX snippet drafting, and language-editing support. We also used it to help check the wording of mathematical definitions, which were verified by the authors against standard formulations. The authors take full responsibility for all code, mathematical definitions, experimental design, analyses, scientific claims, and conclusions.

References

D. C. Cooper. 1972. Theorem proving in arithmetic without multiplication. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence 7*, pages 91–99. Edinburgh University Press.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43.

Jonathan Herzig, Peter Shaw, Ming-Wei Chang, Kelvin Guu, Panupong Pasupat, and Yuan Zhang. 2021. Span-based semantic parsing for compositional generalization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 908–921.

Paul Jaccard. 1901. Étude comparative de la distribution florale dans une portion des alpes et du jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*, 37(142):547–579.

Percy Liang, Michael I. Jordan, and Dan Klein. 2011. Learning dependency-based compositional semantics. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (ACL-HLT)*, pages 590–599.

Jesús A. De Loera, Raymond Hemmecke, Jörg Tauzer, and Ruriko Yoshida. 2004. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302.

Christos H. Papadimitriou. 1981. On the complexity of integer programming. *Journal of the ACM*, 28(4):765–768.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Kevin Woods. 2014. The unreasonable ubiquitousness of quasi-polynomials. *The Electronic Journal of Combinatorics*, 21(1):Paper 1.44.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.

Luke S. Zettlemoyer and Michael Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence (UAI)*, pages 658–666.

Author Index