Generating Natural Language Proofs with Verifier Guided Search

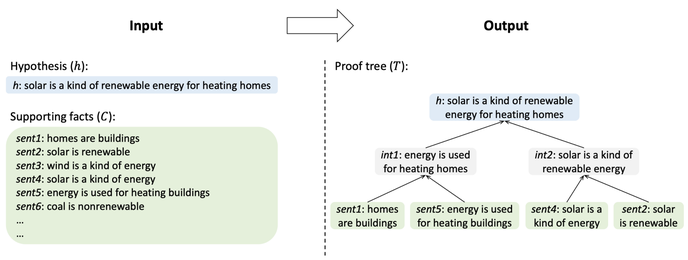
Dwaipayan Saha[†] **Kevin Huang**[†] **Bryan Wang**[†]
[†]Department of Computer Science, Princeton University

PRINCETON UNIVERSITY

How to generate Natural Language Proofs

Input: A hypothesis h and set of supporting facts $C = \{\texttt{sent}_1, \texttt{sent}_2, ..., \texttt{sent}_n\}$, all of which are in natural language.

Output: A proof tree T, where the leaf nodes are $\texttt{sent}_i$, the root is h with intermediate nodes $\texttt{int}_i$ that demonstrate proof steps being tied together to make the conclusion. Consider the image below for completeness:



There are two main ways this has been done in the literature:

1. Single-Shot method, which given the input return the output tree T is one shot.[6]
2. Step-Wise method, which given the input generate each intermediate node step-wise eventually giving the tree. [5]

The work we modify, [3], uses a step-wise method with the caveat of having an trained verifier making significant progress from previous work. Similar to the Yang et al. we also evaluate our results on the EntailmentBank [2] and RuleTaker [4].

Related Work

Generating proof trees is a challenging logical and argumentative task. Natural language processing is an effective way to measure this task due to its ability to parse and infer reasoning structures and patterns in proofs for increasing complexity. The task of proof generation using natural language processing can be approached from two poles: single-shot methods or stepwise methods [5,6]. Previous work has established useful proof trees, but falls short due to various limitations in each technique. Single-shot methods, while generating the entire proof tree in one take (e.g. using linear programming), fail to take into account the relationships for longer, more nuanced proofs. On the other hand, stepwise methods suffer from the issue of generating logically invalid steps. This work expands upon the idea that implementing an additional verifier suite can help a stepwise model miminize invalid steps and satisfy the proof hypothesis.

In the context of design rationale for pseudo-negative sampling, a similar model by Dalvi et al (2022) utilizes a verifier but with human-annotated pseudo-negative examples [1]. This paper's decision to use automatically generated pseudo-negative examples presents a time-saving method towards training the verifier, and the ablations performed in our experiment validate and quantify the degree to which pseudo-negative sampling affects proof generation accuracy.

Acknowledgements

We would like to thank our Professor Danqi Chen for her support and advice throughout the duration of this project, which helped us incredibly to achieve the results we did. We also want to thank Dr. Yang for his prompt responses and advice regarding this project alongside Alex Wettig and Simon Park who also helped us.

Our contributions: Ablations and Extensions

In the scope of this project we make several ablations and baseline extensions to the work in [3]. As they do we focus on step-wise generation with a *trained verifier* throughout the course of our work. Below are our implemented ablations and design changes that we measure performance differences for:

- We train the *prover* and *verifier* on `t5-small` rather than `t5-large` in order to make computational savings.
- We also train with `train:precision: bf16` in order to work with `BFloat16` rather than `Float32` to make computational savings.
- We offer several design alternative frameworks to how pseudo-negative examples are generated in [3]:
  Removing pseudo negative examplesMaking them more robust by adding sepecific negationsAdding frequent random words (perturbations) to conclusions

Here are our stretch goals/extensions that we observe change in performance for:

- We design an extension the current beam search decoder to **diverse beam search**.

All our results are shown to right.

Experimental Results with `T5-Small` and `BFloat16`

The `T5` model is a pretrained encoder-decoder model that can used for a variety of tasks, where all tasks are in a text to text format. They come in various sizes, and while [3] uses the `T5-Large` we use the `T5-Small` instead in order to save train time. Furthermore, we specifically work with 16 point floats rather than 32 in order to create further computational benefit. Using these give us the following result on (dataset/task) in comparison to their results:

Table 1. Results Using Alternate Computation on EntailmentBank Task 2 (`T5-Small` and `bf16`)

Metric	Test Results (Large +32)	Validation Result	Test Result
ExactMatch_leaves_val	58.8	34.2	30.3
ExactMatch_proof_val	37.8	26.7	23.5
ExactMatch_steps_val	34.4	26.7	23.5
F1_leaves_val	90.3	79.1	75.5
F1_proof_val	70.2	26.7	23.5
F1_steps_val	47.2	30.8	29.5

We see that our test results are only slightly worse than the results in found in [3] using `T5-Large` and `Float32`.

Table 2. Comparison of Results to [3] using Alternate Computation on Ruletaker

	Answer Accuracy	Proof Accuracy
NLProofs (Large +32)	99.3	99.2
NLProofs (Small +16)	96.9	96.7

As can be seen our choice of model makes significant saving in train time, by a factor of approximately 5 where an entire experiment took approximately 25 hours on 2 CPUs and 1 A100 Nvidia GPU, while maintaining almost the same accuracy across different metrics.

Experiments with Pseudo Negative Sampling Variants

As noted earlier we experiment with the pseudo negative sampling that is done in the RuleTaker dataset due to its synthetic structure. The first test is to see the importance of these pseudo negative samples, which we do by considering performance upon their removal:

Table 3. Design Framework 1 with no pseudo-generation

	0	1	2	3	N/A	All
Proof Accuracy (Val)	99.83	88.64	80.99	70.65	99.56	92.68
Answer Accuracy (Val)	100.0	99.68	99.74	81.78	99.56	97.27
Proof Accuracy (Test)	99.64	88.35	79.70	68.97	99.25	92.22
Answer Accuracy (Test)	100.0	99.70	95.04	79.68	99.25	96.86

Our finding suggest that Design Framework 1 validates the original experiment design choice: validation and test answer accuracy on all proofs is worse by 4% (92.68 and 92.22 compared to 97.05 and 96.70). We demonstrate our findings for frameworks 2 and 3 similarly.

Sampling	Answer Accuracy	Proof Accuracy
No negative sampling	96.86	92.22
Robust negative sampling	*	*
Perturbations	*	*

Table 4. Comparison of Design Frameworks (Test). * denotes in-progress.

We can see from the results that applying alternative design frameworks to pseudo-negative sampling nuances the rationale behind the original negative sampling offered in the paper. Depending on the method chosen, researchers can choose to dedicate more resources towards making negative sampling more robust by utilizing our work.

Conclusions: Results and Future Work

Our work presents novel ways to improve the design frameworks of natural language proof generation in pseudo-negative sampling. We statistically validate the importance of generating pseudo-negative samples as opposed to without, and we show that alternative designs to pseudo-negative sampling can affect the model accuracy by 4%. We also show the potential for a model weight change in the degree of precision from 32 to 16 floating point, suggesting that computational resources may be better allocated towards pursuits such as robust sampling techniques.

For future work, we plan to implement an extension of the current beam search decoder to diverse beam search in addition to making the decoding invariant to permutation of premises to improve efficiency. We are in discussion with the authors of the original paper to further handoff and evaluate our design frameworks.

References

[1] Dalvi, Bhavana, Oyvind Tafjord, and Peter Clark. "Towards teachable reasoning systems."(2022).

[2] Dalvi, Bhavana, et al. "Explaining answers with entailment trees." (2021).

[3] Yang, Kaiyu, Jia Deng, and Danqi Chen. "Generating natural language proofs with verifier-guided search." (2022).

[4] Clark, Peter, Oyvind Tafjord, and Kyle Richardson. "Transformers as soft reasoners over language." (2020).

[5] Tafjord, Oyvind, Bhavana Dalvi Mishra, and Peter Clark. "Proofwriter: Generating implications, proofs, and abductive statements over natural language." (2020).