Robustification of Natural Language Proof Generation with Verifier Guided Search

Dwaipayan Saha[†] **Kevin Huang**[†] **Bryan Wang**[†]
[†]Department of Computer Science, Princeton University
`{dsaha, kh20, bw22}@princeton.edu`

Abstract

Proof trees present a collection of facts in a tree structure with intermediate nodes that lead to the root hypothesis one wishes to prove. Using Natural Language Proof generation, there are two main techniques, single-shot and stepwise generation of the proof trees which have faced issues with generating irrelevant and invalid steps when given access to the hypothesis. Yang et al. (2022) remedies this by training an additional verifier component, which requires negative examples which are generated rather than using the human annotated versions in Dalvi et al. (2022). This work conduct ablations studies of the work in Yang et al. (2022). We presents novel ways to reduce computational load, improve the design frameworks of natural language proof generation in pseudo-negative sampling, and offer an implementation for diverse beam search.

1 Introduction

The task of generating proof trees is challenging because it requires logical and argumentative skills that are difficult to automate. However, NLP has proven to be an effective way to parse and infer reasoning structures and patterns in proofs, even for increasing complexity.

The status quo on stepwise methods for proof tree generation has seen limited success on real-world data – particularly in the context of generating valid and relevant proof steps. Instead, these methods can produce invalid steps that may be valid but irrelevant, instead of following a systematic and formally-rigorous path towards the final proof.

Yang et al. (2022) proposes a novel solution to this problem by adding an independent verifier that checks the validity of the proof steps. This verifier serves as an intermediary layer between the model and evaluating the result, and it tackles the problem of irrelevant intermediary steps in existing stepwise methods. This approach has shown promising results, but the large training time of the model makes running time a significant concern for practical applications.

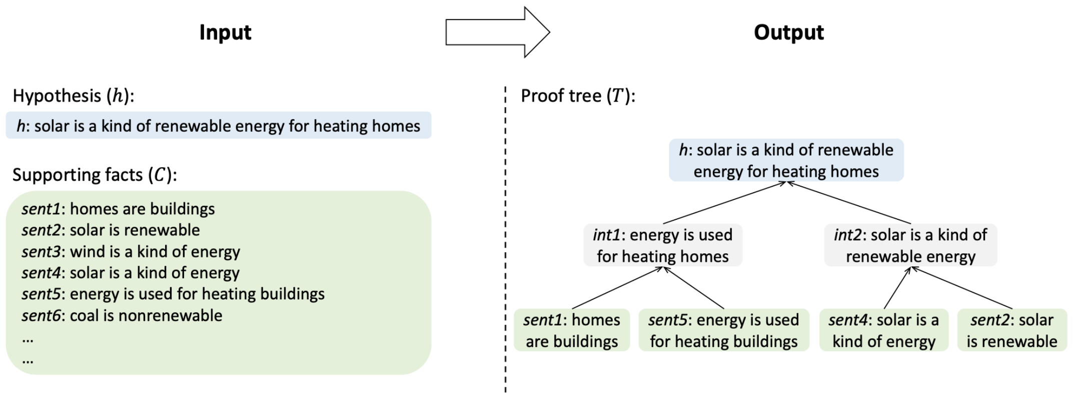
To address this issue, we propose to robustify the model by identifying areas where computational resources can be saved and redistributed towards areas where computational resources can be invested. These optimizations could potentially reduce the training time that currently takes around 20 hours on an A40 or A6000 GPU. **Specifically, we offer multiple ablations and design frameworks** towards the robustification of the natural language generation of proof trees.

In this paper, we suggest switching from the `T5-Large` to the `T5-Small` model and reducing the precision from 32-bit to 16-bit floating point. We also suggest applying more computational resources towards pseudo-negative sampling generation in the verifier and improving the beam search process in the decoder. We aim to investigate the importance of pseudo-negative examples and explore alternative ways of generating different negative examples. We also seek to answer the question of how to reduce training time while maintaining or even improving the performance of the model. To this end, we will conduct a series of experiments to reproduce and generate a new baseline, evaluate and compare our results to the baseline, and offer novel concepts on how to incorporate our design suggestions into the next iteration of the work.

The rest of the paper ends with further work and important takeaways, and presents an extended appendix section for coding details.

2 Original Model

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 with leaf nodes $\texttt{sent}_i$ and root h with intermediate nodes $\texttt{int}_i$ that demonstrate proof steps leading to the conclusion. Consider the image below for completeness:



Input
Output
Hypothesis (h):
h: solar is a kind of renewable energy for heating homes
Supporting facts (C):
sent1: homes are buildings
sent2: solar is renewable
sent3: wind is a kind of energy
sent4: solar is a kind of energy
sent5: energy is used for heating buildings
sent6: coal is nonrenewable
...
...
Proof tree (T):
h: solar is a kind of renewable energy for heating homes
int1: energy is used for heating homes
int2: solar is a kind of renewable energy
sent1: homes are buildings
sent5: energy is used for heating buildings
sent4: solar is a kind of energy
sent2: solar is renewable

Figure 1: Inputs to the model and generation of a proof tree.[1]

As we will see later, the output tree T can be formed in several ways, namely through single-shot and stepwise methods that we will discuss further in Section 4. The paper we seek to modify (Yang et al., 2022) uses a stepwise method, alongside training an independent verifier to score proof generations, and a mechanism to search for high scoring proof trees.

2.1 Stepwise Prover

The stepwise prover is implemented by finetuning a pretrained `T5` model. The authors in Yang et al. (2022) create their training data by extracting proof trees from ground truth proofs. Due to the stepwise nature of generation, they incrementally build partial proofs leading to some intermediate conclusion u. They define feasibility rules that specific partial proof trees are subject to. For example, u's proof may not contain itself or its ancestors but can contain it descendants or nodes that do not fall into either category. This technique can lead to generation of invalid proof steps, which are thereby filtered out by generating a collection from beam search and heuristically checking validity. In this work, we make an attempt to improve this by implementing diverse beam search as one of the ablation studies.

2.2 Verifier

Oftentimes, it is the case that having access to the hypothesis h leads to the model hallucinating and generating invalid proof steps. Thus, they train an independent verifier in order to score these proof steps, any of which take in a set of premises and the conclusion and output a real number score in $[0, 1]$. In order to do so, they finetune the pretrained RoBERTa model (Liu et al., 2019) which allows them to classify these proof steps' validity in a binary classification setting. Naturally, in order to accomplish such a task, one needs both positive and negative examples, and while the positive examples are readily available from ground truth proofs, there do not exist negative examples. Previous work such as (Dalvi et al., 2022) used human annotated negative samples, whereas their work generated pseudo-negative samples. We further their work in pseudo-negative sampling by showing its importance and extending it via several ablations in Section 6. Note that one needs to define a mathematical formula to aggregate the proof step scores for the entire tree. They present a formula in the text, however, one could explore different formulae as long as they are monotonically non-increasing w.r.t step and its children's scores.

2.3 Proof Search

Combining the prover and verifier defined above gives the proof search algorithm in Yang et al. (2022) which seeks to return proofs with high validity scores. Their method was inspired by Russell and Norvig (2010) and unlike greedy stepwise proof generation, they search a larger proof graph, effectively exploring more and achieving better results. We defer the interested reader to Section 4.3 of Yang et al. (2022), since we do not make modifications in this part of the architecture.

3 Datasets

3.1 EntailmentBank

EntailmentBank (Dalvi et al., 2022) is a human-curated dataset consisting of 1,840 proof trees. Importantly, all the data within EntailmentBank was created by expert annotators and are representa-

[1]Image taken from Yang et al. (2022)

tive of real-world proof generation problems. The dataset is split into 3 tasks, each increasing in difficulty based on the given set of facts and distractors.

Labels (C)	**Fact**
sent1	earth is a kind of celestial object
sent2	stars appear to move relative to the horizon during the night
sent3	a star is a kind of celestial object / celestial body
sent4	the earth rotating on its axis causes stars to appear to move across the sky at night
sent5	apparent motion is when an object appears to move relative to another object 's position

Table 1: Sample example from the EntailmentBank dataset. Hypothesis (h): the earth rotating on its axis causes stars to move relative to the horizon during the night

3.2 RuleTaker

RuleTaker is a synthetic dataset created using simple English words and sentence structures. As a synthetic dataset, the number of available training and validation examples is substantially larger: 129K examples in total. Due to the rigid structure of examples in RuleTaker, hypotheses can either be provable, disprovable, or neither.

Labels (C)	**Fact**
sent1	the circuit has the switch
sent2	if the circuit is complete and the circuit has the light bulb then the light bulb is glowing
sent3	if the circuit is complete and the circuit has the bell then the bell is ringing
sent4	if the circuit does not have the switch then the circuit is complete
sent5	if the circuit is complete and the circuit has the radio then the radio is playing
sent6	if the circuit has the switch and the switch is on then the circuit is complete
sent7	the circuit has the light bulb
sent8	the switch is on

Table 2: Sample example from the RuleTaker dataset. Hypothesis (h): the circuit has the light bulb

4 Related Work

As mentioned earlier, the task of proof generation using natural language processing can be approached from two poles: single-shot methods or stepwise methods. Previous work has established useful proof trees, but falls short due to various limitations in each technique. Single-shot methods generate the entire proof tree in one take by using linear programming in order to enforce specific feasibility constraints (Saha et al., 2020) or use pretrained text to text transformers (Gontier et al., 2020). In fact, Yang et al. (2022) provides a framework to produce single shot proofs by finetuning text to text transformers, namely the T5 model. However, such techniques fail to take into account the relationships for longer, more nuanced proofs. This is especially important when considering more complex datasets such as EntailmentBank as highlighted in Section 3.1.

There also exist alternative stepwise methods from Tafjord et al. (2021) that build the proof trees incrementally. It turns out that such techniques suffer from generating invalid steps when the hypothesis is available. This is because with knowledge of the hypothesis, the prover outputs required intermediate conclusions that are not logically sound given the current premises. There is work (Tafjord et al., 2021; Sanyal et al., 2022) that attempts to fix this by disallowing the model to see the hypothesis *apriori*, by only providing known premises and intermediate steps. It turns out that this technique also falls short because, while it generates valid logical proof steps, they are often irrelevant to the hypothesis itself.

This motivates the need for the verifier component introduced earlier. It turns out that Yang et al. (2022) was not the first work to do this, rather, this was first seen in Dalvi et al. (2022). However, they made greedy choices contrary to making an overall optimal choice by using the aggregate score alongside using a notion of pseudo-negative sampling. This work expands upon the idea that implementing an additional verifier suite can help a stepwise model minimize invalid steps and satisfy the proof hypothesis.

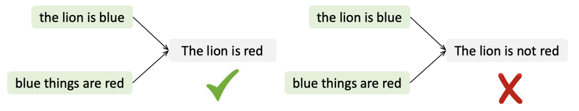


Figure 2: Negative sampling from Yang et al. (2022).

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. 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.

5 Implementation and Setup

We conducted our experiments by setting up multiple code pipelines on multiple clusters. Initially, for a proof of concept, we utilized a Google Colab Pro instance, which uses NVIDIA V100 or A100 Tensor Core GPUs, and setup our local environment via caching and access to Google Drive. For our main experiments, we created accounts on the Ionic cluster from the Princeton Computer Science Department and the Adroit cluster from Princeton Research Computing. The Ionic cluster utilized NVIDIA A5000 and A6000 GPUs and the Adroit cluster utilized NVIDIA A100 GPUs. Note that the majority of our results came from using the Adroit cluster. For both clusters, we incorporated batch job processing by writing slurm files with job specifications. Details regarding number of nodes, number of tasks, GPU hours capacity can be found in the GitHub repository linked at the end of the paper.

In terms of overall training time, the time to run the entire pipeline can be calculated by summing the time to train the model, the time to train the verifier, and the time to compute the test and validation results. The average time for model training took around ∼5 hours, the average time for verifier training took around ∼5 hours, and test and validation combined took around ∼18 hours. For all our experiments, we had a total amount of GPU time around ∼170 hours.

In terms of code files changed, for the first ablation, the configuration files were changed to implement the change to `T5-Small` and `BFloat16`. For the second ablation, alternative `datamodule.py` files were written and relabeled at time of job submission. For the third ablation, diverse beam search was implemented in the `prover/model.py` file.

6 Ablations and Design Frameworks

In the following section we discuss the collection of ablations we do in order to potentially improve on the results presented in Yang et al. (2022). Specifically, we explain the motivation and how we went about implementing each of the ablations mentioned below.

6.1 Ablation 1: Using `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 Yang et al. (2022) uses the `T5-Large`, we use the `T5-Small` instead in order to save on training time for the prover. Furthermore, we specifically work with `BFloat16` which is a custom 16-point floating point format used frequently in machine learning rather than 32-point floats in order to create further computational savings and reduce time to convergence, as was suggested to us by the authors from Yang et al. (2022).

The hope is that we are able to maintain similar accuracy results while saving computational time on both datasets, EntailmentBank and RuleTaker.

6.2 Pseudo-negative Generation

We offer several design alternative frameworks to how pseudo-negative examples are generated in the original paper (Yang et al., 2022). We are interested in the follwing questions: how important are these pseudo-negative examples? Can we come up with other ways of generating different negative examples? Note that we operate purely on the RuleTaker dataset for this section for computational purposes and in order to establish a proof of concept.

6.2.1 Design Framework 2: Removing psuedo-negative examples

This ablation involves a technique to decrease model performance by removing negative example generation entirely. Since we are interested in measuring whether pseudo-negative examples are useful at all, we can simply run the verifier without any explicit additional generation of negative examples. The expected behavior is to see a decrease in the proof and answer accuracy.

6.2.2 Design Framework 3: Increasing robustness by adding specific negations

This design framework involves a technique to improve model performance by adding negations to the training data to help the model better understand the context of negative examples. Specifically, we found that the current pseudo-negative generation only detects certain "not" keywords and transitions them into negative examples. Although this methodology will catch half of the dataset (sentences with a "not" phrase or some other form of "no"), the existing methodology will not be able to create negative samples for sentences without "not" phrases. So, we add keyword detection in the other direction (affirming phrases to denying phrases) as shown in the table below. This ensures that there is a 1:1 ratio of positive to pseudo-negative samples for every sentence in the dataset.

Keyword	New Keyword
`“Not”`	`“<Empty String>”`
`“Does Not”`	`“<Empty String>”`
`“Cannot”`	`“Can”`

Table 3: Keyword detection for pseudo-negative sampling from Yang et al. (2022).

Keyword	New Keyword
`“Does”`	`“Does Not”`
`“Do”`	`“Do Not”`
`“Can”`	`“Cannot”`
`“Is”`	`“Is Not”`

Table 4: Additional keyword detection for pseudo-negative sampling.

Some example cases taken from our model are as follows: 1. "the mouse is nice" becomes "the mouse is not nice"; 2. "the squirrel does visit the bald eagle" becomes "the squirrel does not visit the bald eagle".

6.2.3 Design Framework 4: Adding perturbations to conclusions

This design framework involves a technique to improve model robustness by adding noise to the data, which forces the model to learn more generalizable features. Our perturbations are generated by grabbing the list of top 1000 nouns in the English language by frequency, and then randomly grabbing one noun and modifying the conclusion to include this noun.

Keyword	New Keyword
`“Not”`	`“Not <Random Word>”`
`“Does Not”`	`“Does Not <Random Word>”`
`“Cannot”`	`“Cannot <Random Word>”`

Table 5: Addition of perturbations to pseudo-negative samples.

Note that we were careful not to produce any false negatives in our perturbation casework. A false negative is accidentally generated when the addition of a word would still make the final conclusion true. However, by specifically adding perturbations right after the detection of a verb phrase ("does not, cannot, not"), we ensure that the following perturbed sentence cannot be true as the sentence would not follow grammatically.

Some example cases taken from our model are as follows: 1. "harry is not furry" becomes "harry is not party furry"; 2. the tiger does not eat the dog" becomes "the tiger does not government eat the dog".

6.3 Ablation 5: Diverse Beam Search

This ablation was a stretch goal on top of our existing work. As explained in the Introduction, we use a trained prover in our proof generation process. A major limitation highlighted by the authors of Yang et al. (2022) was that there was a severe lack of variety in the potential proof steps that were generated and that it would be extremely helpful in order to generate a variety of candidates for the search algorithm to explore. A potential solution to this using a more sophisticated decoding model rather than just using beam search is to use Diverse Beam Search as presented in (Vijayakumar et al., 2018). This allows our algorithm to have better coverage and we hope this leads to an increase in performance in both datasets. We implement by using the `HammingDiversityLogitsProcessor` from HuggingFace which allows us to specify two additional parameters in our generation, namely `num_beam_groups` and `diversity_penalty`. The first parameter specifies how many different diverse generations/groups it will return, and the second is a measure of how different it will make the outputs. Overall, one can easily see why this should make a significant improvement to coverage.

7 Results

Here, we give the specific results from each of our proposed ablations corresponding to Section 6.

7.1 Ablation 1: Using `T5-Small` and `BFloat16`

Upon utilizing the `T5-Small` and `BFloat16` we get the following results on the RuleTaker and EntailmentBank datasets respectively. We provide further detailed results in Appendix A.3:

	Large +32	Small +16
Answer Acc.	99.3	97.1
Proof Acc.	99.2	96.9

Table 8: Comparison of Results to [3] using Alternate Computation on RuleTaker

Metric	Test Results (Large + 32)	Test Result (Small + 16)	Validation Result
`ExactMatch_leaves_val`	58.8	30.3	34.2
`ExactMatch_proof_val`	37.8	23.5	26.7
`ExactMatch_steps_val`	34.4	23.	26.7 5
`F1_leaves_val`	90.3	75.5	79.1
`F1_proof_val`	70.2	23.5	26.7
`F1_steps_val`	47.2	29.5	30.8

Table 6: Results Using Alternate Computation on EntailmentBank Task 2 (`T5-Small` and `bf16`)

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

Table 7: Ablation 1 with no pseudo-generation

From tables 8 and 7, we see that our test results are range from only slightly worse to considerably worse than the results in found in Yang et al. (2022) using `T5-Large` and `Float32`.

We see that our choice of model performs worse (how much depends on the metric) than the original setup on EntailmentBank (up to roughly 10 percentage points worse). However, we see that on the RuleTaker dataset, it only performs slightly worse (approximately 3 percentage points lower). This is likely due related to our discussion of the datasets in Section 3. Unlike EnatailmentBank, the RuleTaker dataset is synthetic and generated using simple templates. The types of sentence included in the dataset follow fixed rules (which is what allows them to be synthetically generated). As a result, it is expected that using a weaker model and lower precision causes the model to perform worse, but not by a significant amount. The EntailmentBank dataset, on the other hand, is human-curated and much more representative of real-world examples and English usage. We see that our weaker model and lower precision performs considerably worse on this, showing us that the more computationally-intensive model is necessary for greater accuracy on more complex real-world language.

Therefore, we are able to make significant saving in train time, by a factor of approximately 5 while maintaining almost the same accuracy across different metrics for the RuleTaker dataset, but were unable to do so for the EntailmentBank dataset.

7.2 Results: Pseudo-Negative Generation

7.2.1 Design Framework 2: Removing psuedo negative examples

From table 7, our findings suggest that Design Framework 1 validates the original experiment design choice: validation and test answer accuracy on all proofs is worse by at least 4.5%. The results of our ablation give 92.22% answer and 96.86% proof accuracy compared to the baseline as presented in Section 7.1 with 97.10% answer and 96.87% proof accuracy. We see a demonstrated drop in the performance on the answer accuracy. This makes sense as we expected the pseudo-negative sampling to give the verifier both positive and negative samples in order to better distinguish on test data. The degree of improvement to around half of 10% is a significant finding, and suggests that further iterations of the verifier should include pseudo-negative sampling.

Note that the columns in the table labeled $0, 1, 2, 3$, and `N/A` correspond to different bins of the dataset. The number value bins correspond to dataset entries with the corresponding numerical length of the testing proofs. The `N/A` bin reflects testing examples without ground truth proofs as these cannot be proved or disproved and thus would expect an answer of "neither." Finally, the 'All' bin represents the amalgamation of all these bins.

7.2.2 Design Framework 3: Increasing robustness by adding specific negations

For Design Framework 3, we find that the validation and test answer accuracy is similar to the baseline validation and test accuracy within a .5% error range.

	Answer Acc.	Proof Acc.
Original	97.10	96.87
Robust Sampling	96.90	96.68

Table 9: Comparison of Robust Pseudo Negative Sampling and Baseline result with `T5-Small` and `BFloat16`

From table 9, the results of our ablation give 96.90% answer and 96.68% proof accuracy compared to the baseline as presented in Section 7.1 with 97.10% answer and 96.87% proof accuracy. This could suggest that adding more negative sampling does not significantly affect the verifier's ability to parse between positive and negative examples. This could be due to the law of diminishing returns: since the verifier already has some pseudo-negative examples to begin with, it already has a baseline to distinguish between the positive annotated examples and the automatically generated negative ones. This finding suggests that further iterations of the verifier should not expect an extension of the existing negative sampling methodology to significantly affect model accuracy.

Note that the full table with validation and test results can be given in the Appendix.

7.2.3 Design Framework 4: Adding perturbations to conclusions

For Design Framework 4, we find that the validation and test answer accuracy is similar to the baseline validation and test accuracy within a .5% error range. From table 10, the results of our ablation give 96.84% answer and 96.89% proof accuracy compared to the baseline as presented in Section 7.1 with 97.10% answer and 96.87% proof accuracy. This could suggest that adding more negative sampling does not significantly affect the verifier's ability to parse between positive and negative examples. This could suggest that the model is already robust enough in that adding perturbed negative sentences does not affect the model's ability to distinguish between logically valid and logically invalid conclusions. While an initial impression would assume that perturbed conclusions would lead the model astray in determining the relevant patterns for correct and incorrect proof trees, this finding suggests that further iterations of the verifier should expect perturbed sentences to not affect model accuracy.

	Answer Acc.	Proof Acc.
Original	97.10	96.87
Perturbations	96.84	96.89

Table 10: Comparison of Perturbation Sampling and Baseline result with `T5-Small` and `BFloat16`

Note that the full table with validation and test results can be given in the appendix.

7.3 Ablation 5: Using Diverse Beam Search

Our implementation finished training for the prover on the RuleTaker Dataset. One can extend the work by incorporating a verifier of choice with our checkpoints as referred to in the further work section.

As this was a stretch goal for the project, we have designed the code change, modified the code, and trained the new model; the next step is for the researcher to implement testing.

8 Future Work

For future work, the first step involves the complete testing and validation with our implementation of Diverse Beam Search and seeing if this provides an edge on results for either dataset. One interesting premise is to see if making the decoder invariant to permutations of premises, which would be a significant computational saving. The authors in (Yang et al., 2022) already make a heuristic attempt at this, by implementing `PermutationInvarianceLogitsProcessor` which did not yield performance improvements but can built upon.

The second piece of future work involves applying pseudo-negative generation techniques to EntailmentBank. Our three methods from pseudo-negative generation can be applied to the EntailmentBank dataset; those results can be comparatively analyzed to our results on the RuleTaker dataset. This would allow us to mimic our analysis in 7.1 of applying `T5-Large` to the `T5-Small` model and reducing the precision from 32-bit to 16-bit floating point; and seeing the respective differences in accuracy on EntailmentBank and RuleTaker.

The third piece of future work also includes adding more alternative pseudo-negative sampling techniques. One additional idea for sampling involves noting that RuleTaker only makes use of the same group of names: "bob, fiona, anne, harry, charlie, gary, dave, erin." One experiment could involve replacing a name with a randomly drawn name from the remaining names, or a completely different name not in the dataset altogether. We are in discussion with the authors of the original paper to further handoff and evaluate our design frameworks.

9 Conclusion

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 4.5%. 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. We finally design and implement the change for diverse beam search, offering this step as the next progression in the robustification of the model.

Acknowledgements

We would like to give our thanks to our mentor, Professor Danqi Chen, for her advice and feedback on iterations of our work. We are also extremely thankful to Professor Kaiyu Yang for his prompt responses, without which this project would not have been possible. Lastly, we are thankful to Alexander Wettig and Simon Park for meeting with us. Also a thanks to Princeton University for access to compute clusters.

References

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2022. Explaining answers with entailment trees.

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

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach.

Stuart Russell and Peter Norvig. 2010. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Swarnadeep Saha, Sayan Ghosh, Shashank Srivastava, and Mohit Bansal. 2020. Prover: Proof generation for interpretable reasoning over rules.

Soumya Sanyal, Harman Singh, and Xiang Ren. 2022. Fairr: Faithful and robust deductive reasoning over natural language.

Oyvind Tafjord, Bhavana Dalvi Mishra, and Peter Clark. 2021. Proofwriter: Generating implications, proofs, and abductive statements over natural language.

Ashwin K Vijayakumar, Michael Cogswell, Ramprasath R. Selvaraju, Qing Sun, Stefan Lee, David Crandall, and Dhruv Batra. 2018. Diverse beam search: Decoding diverse solutions from neural sequence models.

Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search.

A Appendix

A.1 Code

The code can be found at https://github.com/dsaha04/NLProof_Final. Any other work can be requested by reaching out to the authors.

A.2 Code Setup and Versioning Consistencies

The code had a few inconsistencies from when the repository was published to now. The first was that the versioning of transformers had been updated and as a result we had to run the following commands in order for the code to successfully run: `conda uninstall transformers` and `pip install transformers` in order to get the more updated version of the library. Secondly, initially all the code evaluation was done by the use of the `torchmetrics` library and the code used the following components: `[Accuracy(threshold=0), AveragePrecision(pos_label=1), Precision(), Recall(), Specificity(), F1Score()]`. Since then, `torchmetrics` had updated their versioning – this was an inconsistency we spotted and reported to the authors who had received similar comments from other groups and they subsequently updated the repository to use the newer `BinaryAccuracy` metrics instead. Thus we imported and the used the following from `torchmetrics.classification`: `[BinaryAccuracy(), BinaryAveragePrecision(pos_label=1), BinaryPrecision(), BinaryRecall(), BinarySpecificity(), BinaryF1Score()]`.

A.3 Collaboration with Original Paper Authors

This section is provided to give transparency on the collaboration process between this paper's authors and the authors of the original paper. We worked with author Kaiyu Yang and Danqi Chen via multiple correspondence. We received feedback regarding our design decisions multiple times, and we are presenting this work as a final version of the edits and suggestions from the original paper authors.

A.4 Complete Test and Validation Results

The tables for Design Framework 3: Increasing robustness of adding specific negations and Design Framework 4: Adding perturbations to conclusions are given on the next page.

	0	1	2	3	N/A	All
Proof Acc. (Val)	100.0	99.89	94.63	79.35	99.56	96.90
Answer Acc. (Val)	100.0	99.89	94.63	80.53	99.56	97.03
Proof Acc. (Test)	100.0	99.65	94.35	77.47	99.57	96.68
Answer Acc. (Test)	100.0	99.65	94.41	79.47	99.57	96.90

Table 11: Test and Validation Results for Design Framework 3

	0	1	2	3	N/A	All
Proof Acc. (Val)	100.0	99.63	99.63	95.74	79.97	97.10
Answer Acc. (Val)	100.0	99.63	99.79	95.06	78.93	96.93
Proof Acc. (Test)	100.0	99.65	94.89	79.11	99.51	96.89
Answer Acc. (Test)	100.0	99.57	94.44	79.25	99.51	96.84

Table 12: Test and Validation Results for Design Framework 4