Applying GPT-3 and Dense Embeddings to NLProofS

Roma Bhattacharjee
rb4785@princeton.edu

Arya Maheshwari
arya@princeton.edu

Zachary S. Siegel
zss@princeton.edu

Abstract

Automated reasoning is becoming increasingly relevant in modern NLP systems, with many studies focused on the specific task of proof generation: a model is asked to use a set of possible supporting facts (*premises*) to generate a proof of a given hypothesis via an *entailment tree*, described by a series of proof steps that logically combine statements. To this end, Yang et al. (2022) recently proposed the NLProofS model, which generates entailment trees by training separate prover and verifier models for use in a proof search algorithm. We conduct two studies to investigate different components of NLProofS: we (1) evaluate an alternate approach that uses pre-trained dense embeddings for the preliminary task of *retrieval* (retrieving premises to use), and (2) study the effects of using GPT-3 with in-context learning in place of the prover model. In the first study, we find that while sentence embedding-based retrieval underperforms compared to the existing retrieval method by roughly 10% on a Recall@25 metric, the possibility of fine-tuning and using more sophisticated embedding-based algorithms present promising avenues for future improvement. In our second study, we find that combining the existing T5 prover and GPT-3 with in-context learning into a hybrid model yields an improvement in overall proof accuracy. Broadly, these results indicate the feasibility of making modular improvements to retrieval and the proof model to create stronger proof generation systems in the future.[1]

1 Introduction

Modern AI-based explanation systems are often required to output a rationale behind their answers. While most systems simply output a couple of lines of rationale, a more robust goal is to require the system to output a structured set of logical steps. In addition to providing clarity on how explanation systems arrive at their conclusions, this can also make training and debugging such systems simpler. It is therefore a productive goal to research the tasks of proof generation and automated reasoning in natural language, which would require a model to output its chain of logical reasoning when arriving at a conclusion.

In particular, this problem of proof generation involves generating a multistep proof tree or "entailment tree" from a given target hypothesis and a set of assumed statements or premises. Leaf nodes in the tree represent given premises, and internal nodes represent intermediate conclusions that are logically drawn from their (two or more) children. See Figure 1 for a depiction of such an entailment tree.

Dalvi et al. (2021) break the problem of proof generation down by delineate three distinct "tasks" of varying difficulty on which to evaluate proof generation models:

1. *Task 1*: the only premises provided are those in the original ground truth entailment tree, so the task is only to determine which proof steps to generate to ultimately arrive at the

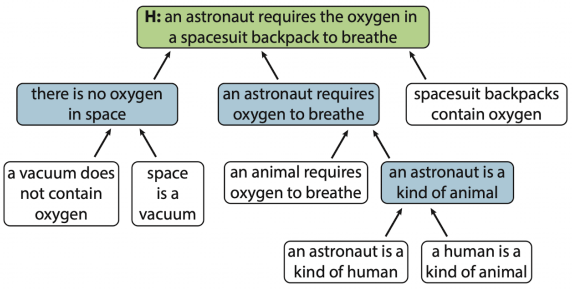

Figure 1: Entailment tree example taken from Dalvi et al. (2021). White nodes are premises, and blue nodes are intermediate steps that are required to arrive at the hypothesis (green node).

[1]The code is available at https://github.com/roma0615/NLProofS and https://github.com/siegelz/entailment_bank. Note that in the former, our code is split across multiple branches: `retrieval`, `nlproofs_pipeline`, and `gpt`.

hypothesis.

2. *Task 2*: some number k of premises (usually $k = 25$) are provided, which includes several "distractor" premises in addition to the required premises. Here, the system must not only generate the correct entailment tree, but it must also discern which facts are relevant for the proof in the first place.

3. *Task 3*: this is a much larger-scale version of *Task 2*—it provides as input all 12K facts from WorldTree, from which the proof generation system must retrieve premises to use for a given hypothesis. This process of *retrieval* is a non-trivial task that is necessary to apply proof generation in "real-world" settings, and has in turn been the focus of other recent research (Neves Ribeiro et al., 2022). It is also the focus of one of the studies in this paper, which is discussed further below.

Follow-up work (including ours) follows this three-task framework in the evaluation of proof generation models (Dalvi et al., 2021; Yang et al., 2022; Neves Ribeiro et al., 2022).

Recent work in Yang et al. (2022)'s paper, "Generating Natural Language Proofs with Verifier-Guided Search," introduces a method called NLProofS, which sets out to tackle the issue of provers "hallucinating" proof steps. This occurs when the model, being aware of the target hypothesis, creates proof steps that do not actually logically entail from the premises it chooses. Attempting to mitigate this result by removing the hypothesis from the prover's input leads to the opposite problem; without knowledge of the hypothesis, the prover is much more likely to meander and generate irrelevant proof steps.

NLProofS aims to solve this hallucination problem by training a separate verifier model based on RoBERTa (Liu et al., 2019) to score the logical validity of a given proof step. This is used to score each step that their prover—a fine-tuned T5 model (Raffel et al., 2020)—generates. At inference time, NLProofS first greedily generates a high-scoring proof, and then non-greedily searches for the proof that achieves the highest aggregate validity score across many possible proof paths.

Our work performs various experiments with the goal of improving upon NLProofS's baseline. Given that the two broad tasks for proof generation are (1) premise retrieval (relevant in particular to Task 3) and (2) proof generation, we conduct two separate studies that cover these areas:

1. **Retrieval using (dense) sentence embeddings.** Yang et al. (2022)'s contribution is the pairing of a prover and verifier for entailment tree generation; they do not explicitly focus on retrieval. Dalvi et al. (2021) train a retrieval model based on BERT (Devlin et al., 2018) and RoBERTa (refer to Appendix A in their paper for details), and NLProofS simply uses the results returned from that same model when evaluating performance in the Task 3 setting. This first study evaluates NLProofS's performance with alternative retrieval methods that use pre-trained dense sentence embeddings to see if gold tree premise recall can be improved.

2. **Replacing NLProofS's T5 prover with GPT-3 + in-context learning.** NLProofS's architecture implements a stepwise prover by fine-tuning a T5 model. Yang et al. (2022) performed an ablation that showed that using GPT-3 to generate the entire proof at once ("single-shot") performed worse than NLProofS, but they did not try using GPT-3 with in-context learning to only generate stepwise proofs. Thus, this study investigates NLProofS's performance after replacing just the T5 prover model with GPT-3.

In our first study, we find that the pre-trained embedding-based retrieval methods perform worse than Dalvi's retrieval model—this is not surprising, since Dalvi's model was specifically trained on the retrieval task, while our embedding-based approaches used pre-trained embeddings. The second study finds that while GPT-3 alone fails to meet the baseline set by NLProofS's T5 prover, using a hybrid of both—T5 for greedy proof generation and GPT-3 for non-greedy proof search—does result in improved performance. Ultimately, these results indicate that it may be fruitful to consider alternative or hybrid approaches to retrieval and proof generation methods.

2 Related Work

In past research, generating natural language proofs has been attempted in two broad ways. One is to generate all proof steps at once (called a "single-shot" approach), and one is to generate proof steps

and intermediate conclusions one at a time ("stepwise" approach). Empirically, the stepwise approach performs better than single-shot approaches, perhaps because generating a single entailment step is less complex than generating an entire proof all at once, and training a model on individual steps can also be better generalized to other examples (Tafjord et al., 2021; Yang et al., 2022). Since NLProofS takes a stepwise approach to proof generation, our work also only focuses on stepwise generation.

Recent work by Neves Ribeiro et al. (2022) investigates iterative retrieval methods as a way to improve relevant premise recall. The motivation is that iterative retrieval allows the model to use generated intermediate steps to fetch related and relevant premises for making further steps, rather than having to intuit which premises will be useful at the beginning. While our first study only investigates non-iterative retrieval in tandem with NLProofS, we do identify integrating a variant of Ribero's iterative retrieval into NLProofS as a compelling avenue of future work.

3 Datasets

To more easily research the proof generation problem, Dalvi et al. (2021) curated a dataset called EntailmentBank. The dataset consists of 1,840 examples of question/answer pairs and the multistep entailment trees necessary to arrive at the answer, generated from ARC science and general knowledge questions. The premises in each tree are sourced from the WorldTree V2 corpus, which contains 11,941 facts about general knowledge and science (Jansen et al., 2018; Xie et al., 2020).

In more detail, each example in EntailmentBank consists of a target hypothesis, a "context" string containing the input premises, and a multistep entailment tree of how to arrive at the hypothesis from the relevant premises. The dataset is partitioned into train (1313 examples), validation (187 examples), and test sets (340 examples). There are three versions of the dataset, one for each task, as described in the introduction. The only aspect of the data that varies between versions is the "context"—the input to the proof generation task. Task 1 and Task 2 contexts are as described in the introduction; to populate the contexts in the Task 3 dataset, Dalvi uses their retrieval model to score all the facts in WorldTree V2 according to their relevance to each example's hypothesis, and returns the top 25.

EntailmentBank also contains annotations about "relevant facts" for each example. While we do not use these annotations in our studies, we do discuss its potential use in our proposed future work (see Section 6.1).

4 Methods

4.1 Study 1

As described above, the task of retrieval involves retrieving, for a given hypothesis h, k facts from a much larger corpus of facts $\mathcal{C}$ (i.e. $k \ll |\mathcal{C}|$) to be used as premises for generating the entailment tree of h. Retrieval is thus a crucial preliminary step of the proof generation pipeline in the more general setting defined by Task 3, where correct premises are not given to the model. In this first study, we focus on this retrieval phase in the context of NLProofS by investigating the effects of replacing the $k = 25$ premises used by Yang et al. for Task 3 (obtained via the outputs of Dalvi et al.'s retrieval) with an alternate retrieval approach that employs sentence embeddings to retrieve premises from the WorldTree corpus for a given hypothesis.

Recent work in information retrieval has suggested that dense embeddings have the potential to provide simpler, more efficient (and thus more practical) methods for retrieval in real-world scenarios, if they can be developed to yield sufficiently good performance (Lin et al. (2023), Thakur et al. (2021)). The key idea that motivates applying this approach to entailment tree premise retrieval is the following: intuitively, facts that score highly in terms of embedding similarity to a given hypothesis seem likely to be relevant premises in proving that hypothesis. As such, our high-level idea is to map a given hypothesis h and all facts in the WorldTree corpus $\mathcal{C}$ to dense vectors using a sentence embedding model, and to then quantify the relevance of candidate premises to the hypothesis through a vector similarity metric, ultimately using this score to determine which premises to retrieve from $\mathcal{C}$ for h.

To this end, we first implement a baseline retrieval algorithm that simply returns the k premises that, as embeddings, yield the highest cosine similarity score to the hypothesis embedding (see pseudocode in Algorithm 1; we set $k = 25$ like in Yang et al.). We evaluate performance of the algorithm on the retrieval task by computing recall (i.e. Recall@25) of the premises used in the gold

Algorithm 1 Baseline Embedding Retrieval

Input: corpus $\mathcal{C}$, hypothesis h, embedding model Emb, context size k

Output: list of k premises for h

procedure BASERETRIEVE
  $\mathbf{h} \leftarrow Emb(h)$
  $\mathbf{C} := \{\mathbf{c}_i\}_{1 \leq i \leq |\mathcal{C}|} \leftarrow Emb(\mathcal{C})$
  scores $\leftarrow \emptyset$ `(dict)`
  for each $c_i \in \mathcal{C}$ **do**
    $scores[c_i] = sim_{cos}(\mathbf{c}_i, \mathbf{h})$
  end for
  scores $\leftarrow$ `sort`(*scores*)
    in descending order by value
  return top k keys of *scores*
end procedure

entailment tree. We additionally run the retrieved premises through the trained NLProofS model to evaluate the "full pipeline" performance against the baseline.

To generate the embeddings themselves, we experiment with three sentence embedding models that are *pre-trained* on other (different) information retrieval tasks: `SimCSE` (Gao et al., 2021), `Contriever` (Izacard et al., 2021), and `DRAGON-RoBERTa` (Lin et al., 2023).[2]

Manual inspection of results from Algorithm 1 on validation data showed that the premises missed by our retrieval were often shorter facts less similar to the full hypothesis but necessary as "building blocks" in the final reasoning (e.g. facts like "an astronaut is a kind of human" for obtaining the final hypothesis "an astronaut requires oxygen in a spacesuit backpack to breathe" as shown in the Figure 1 entailment tree). This suggests one weakness of our underlying assumption that required premises will all be similar to the full hypothesis: some required building block premises may in fact bear little resemblance to the hypothesis, and thus will not be retrieved with this naive similarity score approach.

Based on this observation, as a first step in refining the baseline algorithm to overcome this limitation, we implement a modified algorithm

[2]Specifically, we use the supervised SimCSE model initialized with BERT base (`bert-base-uncased` on HuggingFace) and trained on MNLI/SNLI datasets; and the base (unsupervised) Contriever model, trained with data from CC-net and Wikipedia. Dragon-RoBERTa is a specific RoBERTa-based model in the DRAGON family, which is based on the idea of diverse augmentation. All of these models use a contrastive learning framework.

Algorithm 2 Split-Hypothesis Embedding Retrieval

Input: corpus $\mathcal{C}$, hypothesis h, embedding model Emb, context size k, each-half focused context size w

Output: list of k premises for h

procedure SPLITHYPRETRIEVE
  $\ell_{\text{full}} \leftarrow$ BaseRetrieve($\mathcal{C}$, h, Emb, $k - 2w$)
  $\mathbf{h_l} \leftarrow Emb(\text{right half of } h)$
  $\mathbf{h_r} \leftarrow Emb(\text{right half of } h)$
  $\mathbf{C} := \{\mathbf{c}_i\}_{1 \leq i \leq |\mathcal{C}|} \leftarrow Emb(\mathcal{C})$
  $scores_l, scores_r \leftarrow \emptyset$ `(dict)`
  for each $c_i \in \mathcal{C}$ **do**
    $scores_l[c_i] = sim_{cos}(\mathbf{c}_i, \mathbf{h}_l)$
    $scores_r[c_i] = sim_{cos}(\mathbf{c}_i, \mathbf{h}_r)$
  end for
  $scores_l \leftarrow$ `sort`$(scores_l)$ *by value*
  $scores_r \leftarrow$ `sort`$(scores_r)$ *by value*
  $\ell_l =$ top w keys in $scores_l$ (*)
  $\ell_r =$ top w keys in $scores_r$ (*)
  (*) ensuring ℓ_l, ℓ_r, ℓ_{full} are all distinct
  return $\ell_{full} \cup \ell_l \cup \ell_r$
end procedure

(see pseudocode in Algorithm 2) that selects some premises based on similarity to *part* of the hypothesis rather than the full statement. In particular, we split the hypothesis h into two halves and select w of the final k premises based on similarity to each half (requiring $2w \leq k$), while still selecting $k - 2w$ premises based on the full h (and ruling out premises we have already selected at each step of selection, to ensure the final k are all distinct). [3]

Finally, we evaluate each of these retrieval methods by passing them through the full Task 3 NLProofS pipeline. In particular, we run the NLProofS model trained on Task 2 and evaluate its zero-shot performance on Task 3,[4] using each set of retrieved premises corresponding to each of the six combinations of the embedding models (`SimCSE`, `Contriever`, `DRAGON-RoBERTa`) and retrieval algorithms (Algorithm 1 or 2).

4.2 Study 2

In the second study, we replace the `T5` prover with `gpt-3.5-turbo` (Ouyang et al., 2022) for step-

[3]We briefly discuss alternate splitting implementations in the conclusions and future work section; see 6.1)

[4]Yang et al. also evaluated its performance on Task 3 in this way.

wise proof generation. We theorize that since `gpt-3.5-turbo` was trained on a much larger dataset than `T5`, it could use additional its background knowledge to help at the proof completion task by reasoning with knowledge from different domains.

We train `gpt-3.5-turbo` using in-context learning since fine-tuning would be cost prohibitive. We utilize the same training data and dataloader of the original NLProofS methodology. We randomly sample proof examples to feed into `gpt-3.5-turbo`. The number of training examples is limited by the maximum number of tokens `gpt-3.5-turbo` can take, 4096 tokens. This token limit allows for a maximum of about 8 examples.

In the first experiment, we replace the entire `T5` model with `gpt-3.5-turbo` for both the initial greedy tree generation and for the search method. Therefore, the model works by generating a proof-tree, step-by-step, using `gpt-3.5-turbo`. However, as seen in Table 3, this model performs poorly.

In the second experiment, we elect to use hybrid between the `T5` and `gpt-3.5-turbo` provers since `gpt-3.5-turbo` did not perform well. Specifically, we use `T5` when generating the initial greedy tree in the NLProofS search algorithm, and then use `gpt-3.5-turbo` during non-greedy graph search to improve upon the initial generated tree. This method ensures that the model performance never gets worse than the baseline greedy method, since we use `T5` to generate a baseline tree, and then only use the suggestions of `gpt-3.5-turbo` when it improves upon the initial greedily generated tree, as determined by the verifier score.

Model	Retrieval Algorithm	Recall@25
Dalvi et al.*	–	**0.719**
SimCSE	Baseline (Alg. 1)	0.530
SimCSE	Split-Hyp (Alg. 2)	0.569
Contriever	Baseline (Alg. 1)	0.570
Contriever	Split-Hyp (Alg. 2)	0.586
DRAGON-RoBERTa	Baseline (Alg. 1)	0.603
DRAGON-RoBERTa	Split-Hyp (Alg. 2)	<u>0.625</u>

Table 1: Recall of gold tree premises for each retrieval model (test set). * indicates that model was trained on the EntailmentBank dataset. Best recall **bolded**, runner-up <u>underlined</u>.

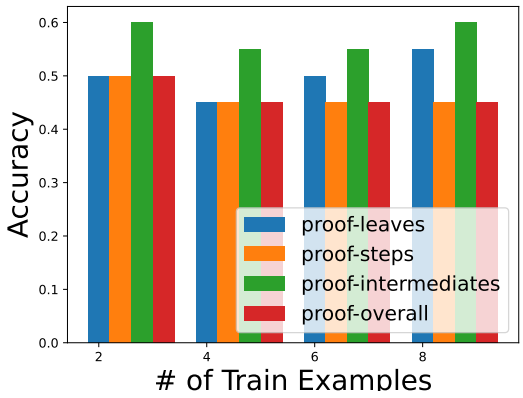

Figure 2: Few Shot Performance of T5 (Greedy) + GPT (Search)

5 Results

5.1 Study 1

Table 1 presents the average Recall@25 of the six combinations of sentence embedding models and retrieval algorithms compared to Dalvi et al.'s retrieval across the 340 examples of the EntailmentBank test dataset. Overall, our embedding-based retrieval methods have weaker performance compared to Dalvi et al.'s retrieval approach, with roughly 10% – 15% worse values of Recall@25. However, this is likely a result, at least in part, of Dalvi et al.'s model being specifically trained on retrieval from WorldTree data, while our algorithm is based on non-tailored, pre-trained embeddings.

Table 2 presents the final performance results after running each retrieval method through the full NLProofS pipeline. That is, for each example in the test set for Task 3, we replace the original context with the custom retrieved context from each evaluated method, and score the accuracy of leaf nodes, structural steps, and intermediate nodes. Results seem to follow those presented in Table 1—higher premise retrieval recall scores loosely correspond with higher F1 and accuracy scores in proof generation. This makes sense, since the prover would not be able to perform accurately if it doesn't have all the required premises at its disposal to build the proof tree.

We do observe slight performance differences between the three sentence embedding models that aligns with reported performance of the three models in previous work (Lin et al., 2023; Gao et al., 2021; Izacard et al., 2021); furthermore, the split-hypothesis algorithm does provide a small performance boost (both in retrieval and the full NL-

Retrieval Method	Leaves		Steps		Intermediates		Overall
	F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	AllCorrect
Dalvi	**43.33**	**8.82**	**10.48**	**6.76**	**42.97**	**16.76**	**6.76**
SimCSE	34.66	6.76	8.17	5.88	36.72	14.41	5.88
SimCSE (split)	37.15	8.24	9.86	**6.76**	37.95	15.59	**6.76**
Dragon	37.65	7.65	9.26	6.47	37.95	14.12	6.47
Dragon (split)	40.15	8.53	10.02	**6.76**	40.59	16.18	**6.76**
Contriever	37.23	7.94	9.81	6.47	39.12	16.47	6.47
Contriever (split)	37.22	7.06	9.18	5.88	38.44	15.00	5.88

Table 2: Results on running NLProofS pipeline using various retrieval methods (test set). Best performance **bolded**, runner-up underlined.

Prover Model	Leaves		Steps		Intermediates		Overall
	F1	AllCorrect	F1	AllCorrect	F1	AllCorrect	AllCorrect
T5 (Greedy + Search)	**86.21**	**50.00**	43.15	35.00	**68.15**	41.67	35.00
T5 (Greedy) + GPT (Search)	81.21	46.67	**45.40**	**36.67**	66.84	**43.33**	**36.67**
GPT (Greedy + Search)	63.74	20.00	15.00	10.00	50.38	10.00	10.00

Table 3: Results on incorporating GPT-3 prover into NLProofS pipeline. Best performance **bolded**, runner-up underlined.

ProofS pipeline result) for all three methods, suggesting the utility of such a finer-grained handling of the hypothesis and motivating the last point in our future work (Section 6.1.1).

5.2 Study 2

As shown in Table 3, replacing `T5` with `gpt-3.5-turbo` for both the initial greedy tree generation and also the proof search method yields poor results. We hypothesize this is due to the lack of in-context examples we could use when training `gpt-3.5-turbo`. Since we are limited by context-size and the proof generation task is different from the underlying distribution `gpt-3.5-turbo` is trained on, we would likely need a larger context size for it to learn effectively. Based on manual inspection of the results, we see that `gpt-3.5-turbo` will often generate invalid syntactic steps (for instance, it will generate a proof step and then attempt to explain the step in natural language) which causes a lower accuracy score.

The hybrid approach of using `T5` for initial greedy tree generation and `gpt-3.5-turbo` for the proof search algorithm yields much better results, in some cases outperforming the full `T5` model. Since `T5` is used to generate the initial greedy tree, `gpt-3.5-turbo` can never do worse than greedy `T5`, which helps explains the good results. But then, `gpt-3.5-turbo` can make improvements to the initial greedily generated tree, which boosts the results even further.

To further demonstrate the impact of `gpt-3.5-turbo` in the proof search algorithm, we plot the accuracy as a function of the number of examples used for in-context learning in Figure 2. As seen, the accuracy never drops below a certain value, even when using a few in-context examples because the model relies completely on the initial greedily generated tree. However, the results marginally improve when increasing the number of in-context examples, showing the additional benefit `gpt-3.5-turbo` confers on overall performance.

6 Conclusions and Future Work

6.1 Study 1

Based on our observed results from Study 1, we ultimately conclude that using pre-trained embeddings does not achieve strong performance compared to Dalvi's model, which was trained *specifically* for retrieval. This indicates that premise retrieval is indeed a complex task that simple similarity scoring cannot fully capture. It has been widely noted in the literature (e.g. as in Thakur et al. (2021); Dai et al. (2022)) that retrievers generalize poorly to datasets across different domains, for example due to different notions for "relevance" that vary significantly. This perhaps explains why

our algorithm may be missing required premises that bear little semantic resemblance to the hypothesis, and is a key performance limitation in our use of pre-trained embeddings.

We also note that the split-hypothesis algorithm does seem to improve recall performance (and consequently the full NLProofS pipeline performance), which indicates that it may be worthwhile to consider alternate ways of drawing out relevant information from the hypothesis; this can be done with a more sophisticated splitting algorithm, or perhaps even by referencing intermediate proof steps to draw out further relevant information for additional premise recall.

These conclusions motivate the following proposed avenues for future work:

1. **Fine-tuning embeddings on ENTAILMENTBANK data**: To improve the performance of retrieval based on dense embeddings, we propose fine-tuning sentence embedding models by training specifically on ENTAILMENTBANK's "relevant fact" data, which could boost our retrieval's recall performance.

2. **Devising a more sophisticated hypothesis splitting algorithm**: Currently, we split hypotheses in half simply by word count. Future work could find the optimal split location (perhaps a semantic split that most differentiates the meanings of each half) or explore splitting complex hypotheses in multiple locations.

3. **Integrating an iterative retrieval method into NLProofS**: Rather than retrieving all the necessary premises up front, the model could leverage intermediate steps to retrieve other related premises, creating an iterative retrieval process *embedded within* proof generation. This method was first introduced by Neves Ribeiro et al. (2022), and demonstrated strong results.

6.2 Study 2

Although our results from combining `gpt-3.5-turbo` with `T5` produces slightly better results with some metrics, there is still room for future improvement. The main goal would be to improve the quantity and quality of training examples fed into `gpt-3.5-turbo`, which is a major current bottleneck. Such improvements include:

1. **Fine-Tuning:** OpenAI allows users to fine tune the `gpt-3` weights for a specific dataset, and we suspect this model would outperform our current results since it could see many more examples of ENTAILMENTBANK compared to in-context learning.

2. **GPT-4:** Once publicly released, `gpt-4` has a context size four times larger than `gpt-3`, and the underlying model is more powerful. We suspect that these two advantages would allow `gpt-4` to outperform our current method.

3. **Tailored Examples:** We have currently been using a static set of training examples for the in-context learning. However, if we chose in-context examples that are similar to the specific hypothesis we wish to prove, those examples could boost the performance of the model since its task would be more similar to the training examples.

Acknowledgements

We thank Prof. Danqi Chen for her guidance throughout this project and Tianyu Gao for correspondence regarding SimCSE.

References

Zhuyun Dai, Vincent Y. Zhao, Ji Ma, Yi Luan, Jianmo Ni, Jing Lu, Anton Bakalov, Kelvin Guu, Keith B. Hall, and Ming-Wei Chang. 2022. Promptagator: Few-shot dense retrieval from 8 examples.

Bhavana Dalvi, Peter Jansen, Oyvind Tafjord, Zhengnan Xie, Hannah Smith, Leighanna Pipatanangkura, and Peter Clark. 2021. Explaining answers with entailment trees. *EMNLP*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Empirical Methods in Natural Language Processing (EMNLP)*.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning.

Peter Jansen, Elizabeth Wainwright, Steven Marmorstein, and Clayton Morrison. 2018. WorldTree: A corpus of explanation graphs for elementary science questions supporting multi-hop inference. In

Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan. European Language Resources Association (ELRA).

Sheng-Chieh Lin, Akari Asai, Minghan Li, Barlas Oguz, Jimmy Lin, Yashar Mehdad, Wen tau Yih, and Xilun Chen. 2023. How to train your dragon: Diverse augmentation towards generalizable dense retrieval. *arXiv e-print 2302.07452*.

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. *CoRR*, abs/1907.11692.

Danilo Neves Ribeiro, Shen Wang, Xiaofei Ma, Rui Dong, Xiaokai Wei, Henghui Zhu, Xinchi Chen, Peng Xu, Zhiheng Huang, Andrew Arnold, and Dan Roth. 2022. Entailment tree explanations via iterative retrieval-generation reasoner. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 465–475, Seattle, United States. Association for Computational Linguistics.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

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

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *CoRR*, abs/2104.08663.

Zhengnan Xie, Sebastian Thiem, Jaycie Martin, Elizabeth Wainwright, Steven Marmorstein, and Peter Jansen. 2020. WorldTree v2: A corpus of science-domain structured explanations and inference patterns supporting multi-hop inference. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 5456–5473, Marseille, France. European Language Resources Association.

Kaiyu Yang, Jia Deng, and Danqi Chen. 2022. Generating natural language proofs with verifier-guided search. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.