ProofDB: A prototype natural language Coq search engine*

Thomas Reichel[1] and Talia Ringer[2]

[1] University of Illinois, `reichel3@illinois.edu`
[2] University of Illinois, `tringer@illinois.edu`

Introduction Proving theorems from scratch is extremely tedious and verbose, so formal proof developments typically build on that which has been previously proven. However, the pool of available formal knowledge is both wide and deep, so even locating relevant premises is challenging. In interactive theorem provers (ITPs) with strong dependent type systems like Coq and Lean the type of a theorem alone is highly descriptive of the content of a theorem, motivating search over types[18, 28]. Unfortunately, it may be the case that the individual using the ITP only has an informal notion of the theorem they wish to locate and doesn't know a fragment of the desired type to provide to a search engine. To support those individuals searching for natural language notions we present ProofDB, which can take a plain-English query and retrieve relevant results from a database of formal proofs. The contributions of this work are a LLM fine-tuned for theorem embedding over a novel dataset, a front-end website for searching databases of theorems, and a Coq plugin that can provide natural language search over custom theorems in a user's local proving environment. A list of links to artifacts are given in Appendix A. In the talk we will demonstrate the user interfaces of ProofDB and discuss challenges and lessons learned throughout the end-to-end process of data collection, training, and deployment of our model.

Related work There is a large body of work in type-oriented search, such as Loogle[18] for theorems in Lean, Hoogle[21] for functions in Haskell, type-aware auto-complete for Coq[9], and built-in searches present in many interactive theorem provers[28]. Several of these tools focus on searching for fragments of types or type patterns using hand-tuned heuristics as opposed to natural language search. On the other hand, Moogle[12], released by Morph Labs, provides natural language search over Lean. Moogle was deemed useful in high profile formalization work by Terence Tao[27], but other than this, we are unaware of any publications or substantial evaluations of Moogle.

Generally, our model is an embedding model which embeds input text into semantically rich vectors. These embeddings can be used for clustering related texts, retrieving relevant information from a database (this work), or even individually as features for text classification. Specifically, our model is trained on synthetic data generated by a causal language model, emphasizing generation of challenging datasets to improve model performance. These strategies are common to other works such as SFR-Embedding-Mistral[20], which is currently ranked first on the MTEB[7] for text retrieval, as well as its predecessor E5-Mistral[15].

Data & Model Current trends in machine learning[2, 4] push the state of the art with training sets and models exponentially larger[6, 10] than those conventionally used in previous decades. Training contemporary competitive models from scratch is prohibitively expensive,[1] but researchers with relatively modest resources can leverage advances in resource-efficient

*Thanks to Max Fan (myfan3@illinois.edu) for early help on infrastructure.

[1]Training the Llama-2 7B param model took 184,320 hours of GPU time on Nvidia A100s[13] which would cost over $200,000 if bought at Lambda Lab's current cloud prices.

training techniques[11] to fine-tune large models pre-trained on a wide domain of tasks to excel in a narrow domain. These trends led us to fine-tune the Llemma[8] language model, which was trained on a dataset of mathematics and formal methods texts, including Coq code, making it an appropriate base for learning insightful representations of Coq theorems.

Although Llemma is a causal language model, we trained it to become an embedding model that converts the text of theorems to semantically rich vectors by fine-tuning less than two percent of the base model's parameters with QLORA[11]. In order to fine-tune the model, we needed a dataset that captures notions of relevancy between theorems and natural language searches. As far as we are aware, no such public dataset existed prior to this work. For our first dataset, we synthesized positive examples by prompting an open-weight LLM[14] to write example searches given a brief natural language description of the target theorem summarized from documentation (data example in Appendix C). Negative examples were generated by randomly associating a theorem and search query from the pool of all theorems and search queries, which are unrelated on average. We fine-tuned Llemma on this dataset to move positive pairs of text closer in the embedding space and negative pairs of text further away, which gave us our first iteration 'proofdb'.

Literature on training embedding models strongly emphasizes the importance of difficult training examples when training strong embedding models [20, 15, 1], but in the dataset previously described our negative training examples were easy to distinguish from positive examples because randomly selected topics are, on average, clearly unrelated. In order to improve the dataset's difficulty, we used generation probabilities as a proxy for relevance: synthetic queries that are generated with higher probability are deemed more relevant than those with lesser probability. This let us sample for a dataset of (POSITIVE THEOREM, NEGATIVE THEOREM, ANCHOR SEARCH) triples where the 'anchor' is a generated search result that is relevant for the positive theorem and less relevant (but not excessively irrelevant) for the negative theorem. Limiting the irrelevancy of the negative theorem should make it harder to rank the positive theorem over the negative theorem for the given anchor. This 'phase-2' dataset is explorable through a link in Appendix A. We continued training our 'proofdb' model on this more difficult dataset with a triplet loss[1] appropriate for learning this relationship. The resultant model, 'proofdb-HN', scored competitively on our benchmarks (Appendix D).

Web Search and Client To provide the capabilities of our model to the end-user, we wrote a website that serves inference of our model over the theorems contained within a collection of popular Coq packages[19]. For users that need search on more specific theorems and do not have the time or resources needed to set up their own proofdb website instance, we have written a local client which searches over the theorems present in the user's active Coq session. In order to support low-resource clients, we serve them embeddings from an API co-hosted with the web search. This presents additional challenges including embedding theorems on the fly, transmitting large amounts of theorems/embeddings, and efficiently searching a small local vector cache on the client. We partially mitigate these challenges by caching embeddings on both the server and client side and employing embedding compression techniques[3] to reduce the size of the embeddings while maintaining probablistic bounds on the degradation introduced by compression. Both our web search and local client provide deterministic search filters that complement the capabilities of our fuzzy natural language search. Sources (A) for these clients and example searches (B) are available in the appendix.

One limitation of our search implementations is that we do not currently search every single theorems present in a switch or local environment, which we partially attribute to outlier theorems of extremely long length and other edge cases.

References

[1] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, June 2015. DOI: 10.1109/cvpr.2015.7298682. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[2] OpenAI. *Deep double descent*. 2019. URL: https://openai.com/research/deep-double-descent.

[3] Xinyang Yi, Constantine Caramanis, and Eric Price. *Binary Embedding: Fundamental Limits and Fast Algorithm*. 2019. arXiv: 1502.05746 [cs.DS].

[4] Jared Kaplan et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG].

[5] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[6] William Fedus, Barret Zoph, and Noam Shazeer. *Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity*. 2022. arXiv: 2101.03961 [cs.LG].

[7] Niklas Muennighoff et al. "MTEB: Massive Text Embedding Benchmark". In: *arXiv preprint arXiv:2210.07316* (2022). DOI: 10.48550/ARXIV.2210.07316. URL: https://arxiv.org/abs/2210.07316.

[8] Zhangir Azerbayev et al. *Llemma: An Open Language Model For Mathematics*. 2023. arXiv: 2310.10631 [cs.CL].

[9] Hjalte Dalland, Jakob Israelsen, and Simon Kristensen. *Expanding Coq With Type Aware Code Completion*. 2023. URL: https://github.com/Jakobis/vscoqComparison/blob/main/Expanding_Coq_with_Type_Aware_Code_Completion.pdf.

[10] Sajal Dash et al. *Optimizing Distributed Training on Frontier for Large Language Models*. 2023. arXiv: 2312.12705 [cs.DC].

[11] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG].

[12] Morph. *Moogle: Semantic search over mathlib4*. 2023. URL: https://moogle.ai.

[13] Hugo Touvron et al. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. 2023. arXiv: 2307.09288 [cs.CL].

[14] Baptiste Rozière et al. *Code Llama: Open Foundation Models for Code*. 2024. arXiv: 2308.12950 [cs.CL].

[15] Liang Wang et al. *Improving Text Embeddings with Large Language Models*. 2024. arXiv: 2401.00368 [cs.CL].

[16] *all-distilroberta-v1*. URL: https://huggingface.co/sentence-transformers/all-distilroberta-v1.

[17] *all-mpnet-base-v2*. URL: https://huggingface.co/sentence-transformers/all-mpnet-base-v2.

[18] Joachim Breitner. *Loogle!* URL: https://loogle.lean-lang.org/.

[19] *Coq Platform*. URL: https://github.com/coq/platform.

[20] Rui Meng et al. *SFR-Embedding-Mistral: Enhance Text Retrieval with Transfer Learning.* URL: `https://blog.salesforceairesearch.com/sfr-embedded-mistral/#teacher-models-for-hard-negative-mining`.

[21] Neil Mitchell. *Hoogle.* URL: `https://hoogle.haskell.org/`.

[22] *multi-qa-distilbert-cos-v1.* URL: `https://huggingface.co/sentence-transformers/multi-qa-distilbert-cos-v1`.

[23] *multi-qa-miniLM-L6-v2.* URL: `https://huggingface.co/sentence-transformers/multi-qa-MiniLM-L6-v2`.

[24] *multi-qa-mpnet-base-dot-v1.* URL: `https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-dot-v1`.

[25] *Pretrained Models – Sentence-Transformers Documentation.* URL: `https://sbert.net/docs/pretrained_models.html`.

[26] *stdpp.* URL: `https://gitlab.mpi-sws.org/iris/stdpp`.

[27] Terence Tao. URL: `https://mathstodon.xyz/@tao/111360264941915326`.

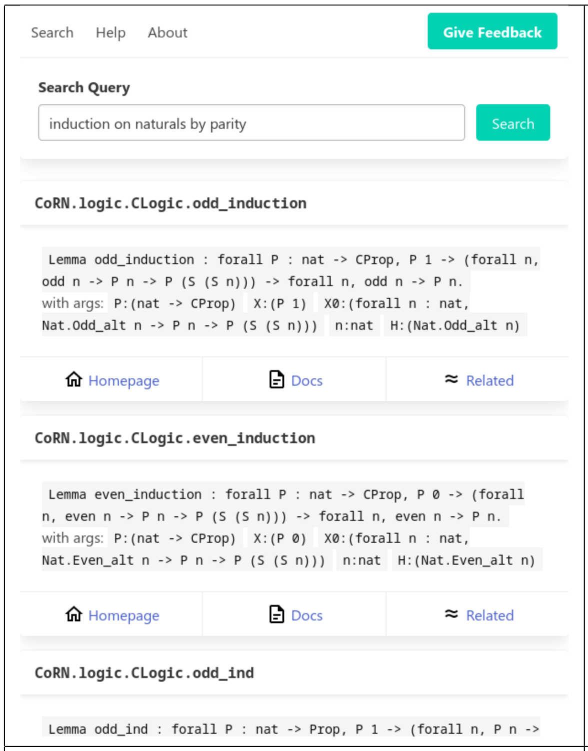
[28] *Vernacular commands.* URL: `https://coq.inria.fr/doc/master/refman/proof-engine/vernacular-commands.html?highlight=search#coq:cmd.Search`.

Appendix

A Appendix: Full List of Artifact Links

1. Live Web UI Demo (https://proofdb.tompreichel.com), availability not guaranteed!
2. Models & Datasets
   (a) Phase 1 training data (https://huggingface.co/datasets/tomreichel/proofdb-training-phase-1)
   (b) Phase 2 training data (https://huggingface.co/datasets/tomreichel/proofdb-training-phase-2)
   (c) Human-written test set (https://huggingface.co/datasets/tomreichel/proofdb_human_eval)
   (d) GPT-written test set (https://huggingface.co/datasets/tomreichel/proofdb-synthetic-eval)
   (e) ProofDB model (https://huggingface.co/tomreichel/proofdb)
   (f) ProofDB-HN model (https://huggingface.co/tomreichel/proofdb-HN)
3. Software Source
   (a) Web UI Source (https://github.com/tom-p-reichel/proofdb-webui/)
   (b) Client Source (https://github.com/tom-p-reichel/proofdb-webui-client)

B Appendix: Selected Web UI Search Examples



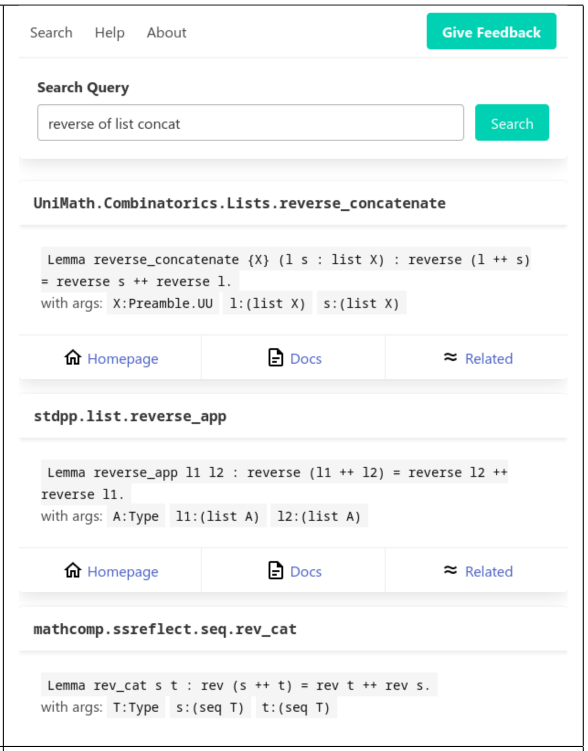
Search Query

induction on naturals by parity

```
CoRN.logic.CLogic.odd_induction

Lemma odd_induction : forall P : nat -> CProp, P 1 -> (forall n,
odd n -> P n -> P (S (S n))) -> forall n, odd n -> P n.
with args: P:(nat -> CProp) X:(P 1) X0:(forall n : nat,
Nat.Odd_alt n -> P n -> P (S (S n))) n:nat H:(Nat.Odd_alt n)
```

```
CoRN.logic.CLogic.even_induction

Lemma even_induction : forall P : nat -> CProp, P 0 -> (forall
n, even n -> P n -> P (S (S n))) -> forall n, even n -> P n.
with args: P:(nat -> CProp) X:(P 0) X0:(forall n : nat,
Nat.Even_alt n -> P n -> P (S (S n))) n:nat H:(Nat.Even_alt n)
```

```
CoRN.logic.CLogic.odd_ind

Lemma odd_ind : forall P : nat -> Prop, P 1 -> (forall n, P n ->
```



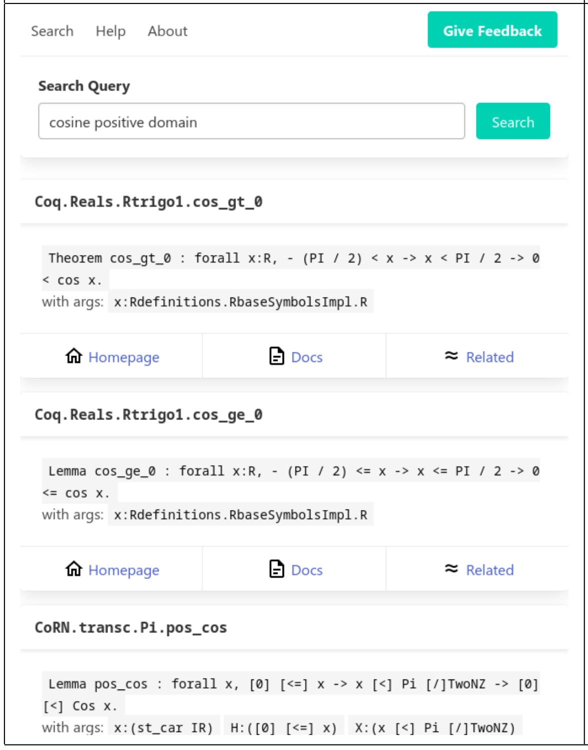
Search Query

reverse of list concat

```
UniMath.Combinatorics.Lists.reverse_concatenate

Lemma reverse_concatenate {X} (l s : list X) : reverse (l ++ s)
= reverse s ++ reverse l.
with args: X:Preamble.UU l:(list X) s:(list X)
```

```
stdpp.list.reverse_app

Lemma reverse_app l1 l2 : reverse (l1 ++ l2) = reverse l2 ++
reverse l1.
with args: A:Type l1:(list A) l2:(list A)
```

```
mathcomp.ssreflect.seq.rev_cat

Lemma rev_cat s t : rev (s ++ t) = rev t ++ rev s.
with args: T:Type s:(seq T) t:(seq T)
```



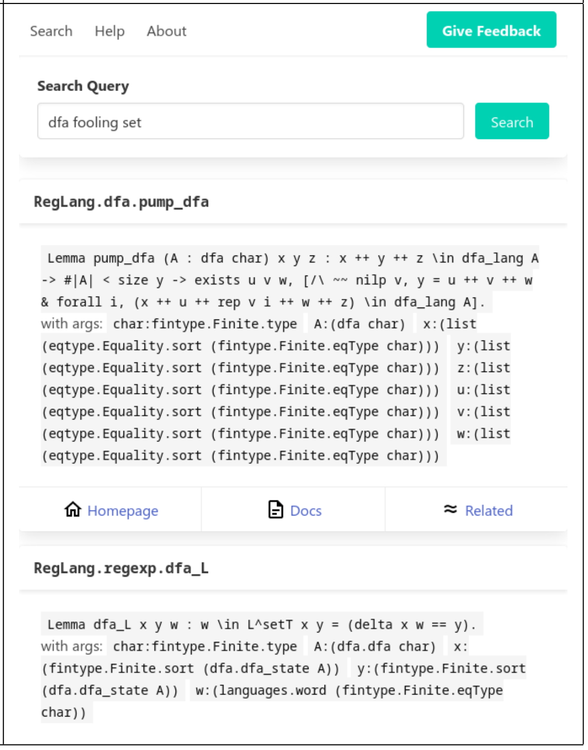
Search Query

cosine positive domain

```
Coq.Reals.Rtrigo1.cos_gt_0

Theorem cos_gt_0 : forall x:R, - (PI / 2) < x -> x < PI / 2 -> 0
< cos x.
with args: x:Rdefinitions.RbaseSymbolsImpl.R
```

```
Coq.Reals.Rtrigo1.cos_ge_0

Lemma cos_ge_0 : forall x:R, - (PI / 2) <= x -> x <= PI / 2 -> 0
<= cos x.
with args: x:Rdefinitions.RbaseSymbolsImpl.R
```

```
CoRN.transc.Pi.pos_cos

Lemma pos_cos : forall x, [0] [<=] x -> x [<] Pi [/]TwoNZ -> [0]
[<] Cos x.
with args: x:(st_car IR) H:([0] [<=] x) X:(x [<] Pi [/]TwoNZ)
```



Search Query

dfa fooling set

```
RegLang.dfa.pump_dfa

Lemma pump_dfa (A : dfa char) x y z : x ++ y ++ z \in dfa_lang A
-> #|A| < size y -> exists u v w, [/\ ~~ nilp v, y = u ++ v ++ w
& forall i, (x ++ u ++ rep v i ++ w ++ z) \in dfa_lang A].
with args: char:fintype.Finite.type A:(dfa char) x:(list
(eqtype.Equality.sort (fintype.Finite.eqType char))) y:(list
(eqtype.Equality.sort (fintype.Finite.eqType char))) z:(list
(eqtype.Equality.sort (fintype.Finite.eqType char))) u:(list
(eqtype.Equality.sort (fintype.Finite.eqType char))) v:(list
(eqtype.Equality.sort (fintype.Finite.eqType char))) w:(list
(eqtype.Equality.sort (fintype.Finite.eqType char)))
```

```
RegLang.regexp.dfa_L

Lemma dfa_L x y w : w \in L^setT x y = (delta x w == y).
with args: char:fintype.Finite.type A:(dfa.dfa char) x:
(fintype.Finite.sort (dfa.dfa_state A)) y:(fintype.Finite.sort
(dfa.dfa_state A)) w:(languages.word (fintype.Finite.eqType
char))
```

C Appendix: Synthetic Data Example

As an example, we will look at how our data pipeline generated positive synthetic search examples for the theorem Exists_vlookup from the stdpp[26] library, which is defined in the following way:

```
Lemma Exists_vlookup {A} (P : A -> Prop) {n} (v : vec A n) :
  Exists P (vec_to_list v) <-> exists i, P (v !!! i).
```

Firstly, we prompt a LLM (Code Llama 33B[14]) to give us a natural language description of the theorem given the source of the theorem and the surrounding source (including developer comments) as context. For this theorem, the following description was generated:

```
The theorem ‘Exists_vlookup‘ states that there exists an element ‘x‘ in the vector ‘v‘
such that the proposition ‘P‘ is true for ‘x‘. In other words, it checks if there is
at least one element in the vector that satisfies the given property. The theorem is
written in a way that makes it easy to use with the ‘Exists‘ quantifier, which is often
used in formal proofs to state that there exists at least one object with a certain property.
```

Then we prompt for concise searches a user might use in order to find this theorem, given the theorem's type and description as context. In order to get consistently well formatted output from the model, we found it was necessary to give a well-formatted example. The following searches were proposed by the model:

```
* "vector exists"
* "vector contains"
* "vector has element"
* "vector property"
* "vector quantifier"
* "vector exists property"
* "vector property exists"
* "vector contains property"
* "vector has property"
* "vector quantifier property"
* "vector property quantifier"
* "vector element property"
* "vector property element"
* "vector quantifier element"
* "vector element quantifier"
* "vector element property quantifier"
```

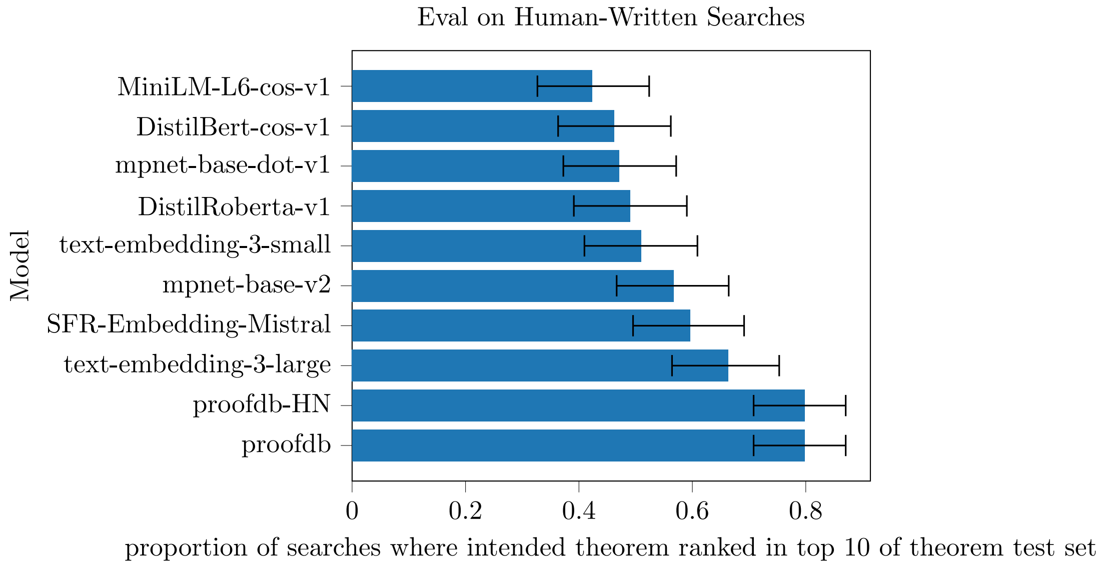
Finally, we perform certain preprocessing steps on the generated theorems to eliminate common failure modes of the model, such as quoting the theorem name verbatim.

D Appendix: Model Evaluation

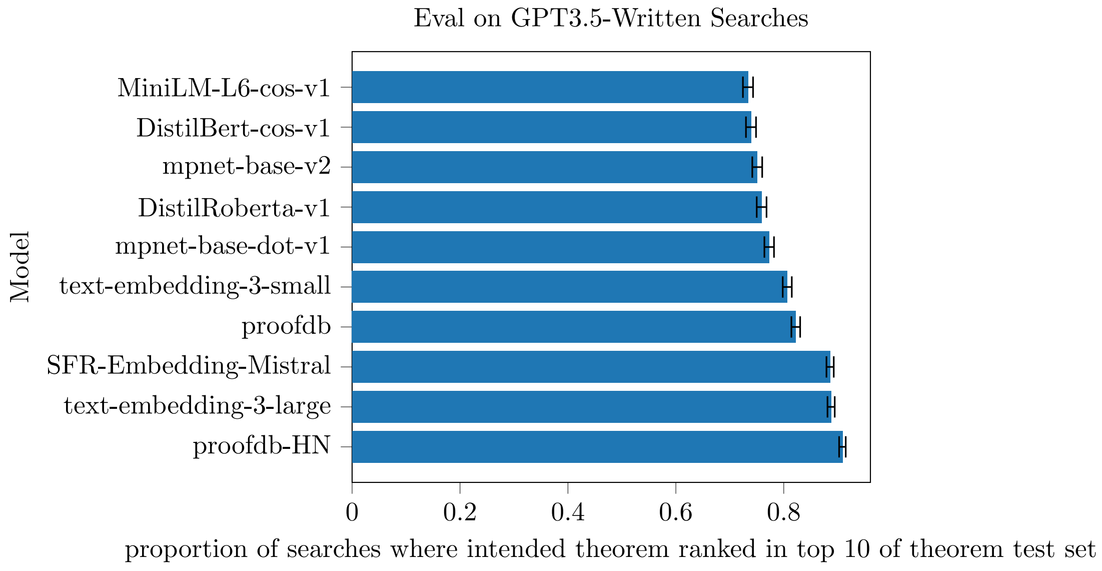
To evaluate our model, we run searches on a set of held-out test theorems and count how often the model can retrieve the intended label theorem within the top 10 theorems. We evaluate other embedding models in the same way to compare against our models. For our evaluations we provide 95% confidence interval error-bars on the proportion of intended theorems retrieved in the top 10 results as calculated by SciPy's[5] 'binomtest'. Here are the list of models we evaluate in this way:

1. Our models.
   (a) proofdb, trained to move synthetic positive (theorem,search) pairs closer together and sampled negative pairs further away.
   (b) proofdb-HN, trained on triplets of (theoremA,theoremB,search) to move the search closer to theoremA than it is to theoremB. Uses 'proofdb' as a pretrained base.
2. Models featured on the SentenceTransformers pretrained models page[25], described as general purpose embedding models and trained on at least 1 billion training pairs.
   (a) all-mpnet-base-v2[17], best overall performer on Sentence-Transformer's (selected) pretrained model list.
   (b) all-distilroberta-v1[16]
3. Models featured on the SentenceTransformers pretrained models page[25], trained for retrieval tasks specifically.
   (a) multi-qa-mpnet-base-dot-v1[24], best performer on the SentenceTransformer pretrained models page when evaluated on semantic search, second best overall.
   (b) multi-qa-distilbert-cos-v1[22]
   (c) multi-qa-MiniLM-L6-v2[23]
4. OpenAI's text embedding models (proprietary).
   (a) text-embedding-small
   (b) text-embedding-large
5. Models from the MTEB[7] leaderboard.
   (a) SFR-Embedding-Mistral[20], currently the best general purpose text retrieval model on the MTEB.

Our models perform promisingly well on our small scale human written search dataset (linked in Appendix A), even against much larger embedding models such as OpenAI's text embedding models (text-embedding-3-*), and SFR-Embedding-Mistral, which is currently ranked first on the Massive Text Embedding Benchmark[7] for text retrieval. However, these test results have limited generalizability because they were written by one author on a small number of arbitrarily chosen theorems. The same author also provided the few-shot example used in the context of the positive example synthesis pipeline, so the generation of the training set may be disproportionately prone to the biases of the author.



For a larger scale test, we let GPT3.5 write searches for the theorems in our test split. Note that we did not give GPT3.5 an example so as to minimally contaminate the output with our preferences. The precise prompt for GPT3.5 is given in the next appendix section (E). For the most part our synthetic trial agrees with the small scale human trial. Notably, proofdb without hard-negative fine-tuning (the phase 2 dataset) does not perform as well as the human written evaluation would indicate. We are unsure if this is a quirk of synthetic trials or a quirk of the human trials, but we plan to conduct more thorough evaluation of human search preferences at a later date. Either way, proofdb-HN performs well on both human-written searches and synthetic searches.



It should be noted that a limitation of both of these evaluations is that the test set may have theorems that are *very similar* to theorems in the training set: many Coq libraries prove results present in other libraries in only slightly different ways, so multiple similar variants of common theorems were likely split into both test and training sets. This can compromise the significance of the benchmark because the model may behave poorer on theorems markedly distinct from both the test and training set. Although, our models are relatively cheap to retrain and new training data can be synthetically generated, so new libraries can more easily become part of the training set than in other settings, which slightly alleviates concerns about model generalization performance.

E Appendix: Verbatim Prompt Given to GPT3.5 to Generate Test Synthetic Searches

Newlines added for readability.

~~~
Consider the following Coq theorem definition:

```coq
{theorem}
```

What are some searches a user might provide to a hypothetical natural language Coq theorem
search engine that would best represent this theorem?
Your searches should be concise.
Don't write full sentences.
There is no need to write in your search that you are looking for a 'Coq theorem'
because the search engine only searches Coq theorems.
Give your response as a JSON object with a single key 'searches' containing a list of strings.
~~~