# CoProver: A Recommender System for Proof Construction

Eric Yeh, Briland Hitaj, Sam Owre, Maena Quemener, and Natarajan Shankar

SRI International, Menlo Park, CA 94025, USA
{eric.yeh,briland.hitaj,sam.owre,maena.quemener,natarajan.shankar}@sri.com

**Abstract.** Interactive Theorem Provers (ITPs) are an indispensable tool in the arsenal of formal methods experts as a platform for construction and (formal) verification of proofs. The complexity of the proofs in conjunction with the level of expertise typically required for the process to succeed can often hinder the adoption of ITPs. A recent strain of work has investigated methods to incorporate machine learning models trained on ITP user activity traces as a viable path towards full automation. While a valuable line of investigation, many problems still require human supervision to be completed fully, thus applying learning methods to assist the user with useful recommendations can prove more fruitful. Following the vein of user assistance, we introduce CoProver, a proof recommender system based on transformers, capable of learning from past actions during proof construction, all while exploring knowledge stored in the ITP concerning previous proofs. CoProver employs a neurally learnt sequence-based encoding of sequents, capturing long distance relationships between terms and hidden cues therein. We couple CoProver with the Prototype Verification System (PVS) and evaluate its performance on two key areas, namely: (1) Next Proof Action Recommendation, and (2) Relevant Lemma Retrieval given a library of theories. We evaluate CoProver on a series of well-established metrics originating from the recommender system and information retrieval communities, respectively. We show that CoProver successfully outperforms prior state of the art applied to recommendation in the domain. We conclude by discussing future directions viable for CoProver (and similar approaches) such as argument prediction, proof summarization, and more.

## 1 Introduction

Interactive theorem proving (ITP) is a well-entrenched technology for formalizing proofs in mathematics, computing, and several other domains. While ITP tools provide powerful automation and customization, the task of manually guiding the theorem prover toward QED is still an onerous one. For inexperienced users, this challenge translates to crafting mathematically elegant formalizations, identifying suitable proof commands, and diagnosing the root cause of failed proof attempts. Whereas for the expert users, the challenge consists in navigating a large body of formalized content to ferret out the useful definitions and the right lemmas. Both novice and expert users can benefit from recommendations in the form of proof commands and lemma retrieval that can guide proof construction.

The goal of the present project is to scale up proof technology by introducing CoProver as a proof recommender system that discerns suitable cues from the libraries, the proof context, and the proof goal to offer recommendations for ITP users [44]. Building on the proof technology and proof corpora of the Prototype Verification System (PVS) a state-of-the-art proof assistant [37], we focus on recommendations for two key tasks in ITP: Suggesting PVS commands and lemmas. The first is to recommend the likely command an expert user would take given the current proof state. As there are hundreds of possible commands, recommending steps an expert may take would be beneficial, particularly for novices. The second is to identify lemmas for inclusion from a library of lemmas that may help with forward progress on a proof. Currently, only lemmas from user-imported theories are considered and selection of a lemma relies on user familiarity with candidate theories and their lemmas. For the problems PVS is commonly employed on, there are usually several hundred theories with thousands of possible lemmas combined to consider. At this scale, even expert users with decades of experience may not be aware of all possible lemmas (or even their names) that may be relevant for their proof. A mechanism that can automatically identify relevant lemmas at scale would be desirable.

To develop these capabilities, we leverage the expert proof traces for NASA's PVS Library [1] (PVSLib), a large collection of highly polished formal developments centered on safety-critical applications, and the PVS Prelude, a collection of theories built into PVS. We aim to capture the expertise and intuition of the developers by training systems to emulate user decisions on these completed proofs. Key to our approach is the use of recent machine learning techniques that can capture sequential information across a greater window than previously possible. We show how these methods using a simple sequence-based encoding of formulas better capture relationships between proof states with commands and libraries than prior sequence encoding techniques such as bag-of-words or graph-based representations [47].

We start with an overview in Section 2 of the CoProver system, describing the core recommendation tasks. Section 3 contains implementation details of how sequents and states are featurized into a common representation used to provide inputs for the command prediction and lemma retrieval capabilities. Sections 4 and 5 provide specific details of how these are implemented, along with experiments detailing their effectiveness described in Section 6. Section 7 examines and contrasts against prior work, and Section 8 concludes with a discussion of future directions.

## 2 CoProver Overview

Figure 1 illustrates the CoProver system. Featurization converts the sequent and previous commands into a token sequence for the transformer model. This

[1] `https://shemesh.larc.nasa.gov/fm/pvs/PVS-library/`

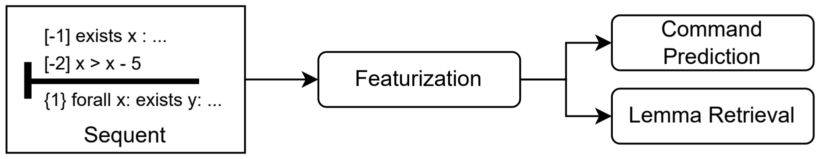


Fig. 1: The CoProver system employs a common sequent featurization, which is used to recommend commands and retrieve relevant lemmas.

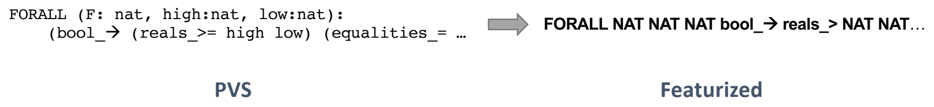
is provided to the Command Prediction and Lemma Retrieval modules. Command Prediction identifies the next likely command an expert user would take in successful proofs, given similar states. When a lemma is to be imported, Lemma Retrieval examines the state and suggests the most relevant ones from a given library, based off a history of human-selected lemmas that have progressed their proofs.

Both of these use RoBERTa [30], a transformers-based neural language model [46] capable of learning long-range sequences, to encode the proof state sequence tokens. Unlike $n$-grams or other Markov window methods, transformers employ a self-attention mechanism that allows features to derived from a significantly wider window of tokens. Tokens are represented as real-valued vectors tailored as inputs for a variety of tasks, and have been used to give state-of-the-art performance across multiple tasks such as large language modeling, text classification, and visual understanding. For command prediction, the RoBERTa-based encoding of the proof state is used as input to a multinomial classifier for predicting the next command an expert user would take. Backpropagated error from the classifier is used to adjust, or to *fine-tune* these representations make them more suitable for the classification task.

Lemma retrieval aims to make forward progress in a proof by identifying relevant lemmas from a library of theories for inclusion. A major challenge is the fact these lemmas may exist in theories that the user is not aware of, nor remember. This is similar to the core problem of information retrieval (IR) [33], where the goal is to retrieve documents from a collection most relevant to a query. IR models rely on heuristics motivated by natural language, such as overlap and term rarity to assess relevance. These assumptions may not hold in theorem-proving, so CoProver is trained on user-made lemma import decisions to fine-tune the proof state representations to learn combinations of sequent and lemma symbols useful for identifying lemma relevance. This focus on human-driven selection also differs from other work in premise selection, where the primary aim is to identify lemmas that allow a hammer to automatically complete the proof.

## 3 Data Generation

For both command prediction and lemma retrieval, we used proof sequences from the PVSLib library, a large set of formal developments containing theorems



```
FORALL (F: nat, high:nat, low:nat):
    (bool_→ (reals_>= high low) (equalities_= …
```

```
FORALL NAT NAT NAT bool_→ reals_> NAT NAT…
```

Fig. 2: Featurization converts formulas in PVS (left) into a more machine learning friendly token stream (right).

proofs for a variety of mathematical and engineering areas. In total PVSLib contains 184,335 proof steps. We note that these are completed and polished proofs so that backtracked sequences of steps are pared and only the successful sequence of proof commands and imported lemmas are retained.

To make the logical formulas amenable to machine learning, we first tokenize them and then use Byte Pair Encoding (BPE) [11,43] from the Huggingface Library [49] to train a token vocabulary customized for PVS. BPE encodes words as a sequences of byte pairs instead of singular tokens, reducing the size of the vocabulary: Rare or unknown words can be encoded constituent byte pairs, while common words are be encoded in their entirety to improve efficiency. Transformer models have fixed width inputs, so a more parsimonious encoding that strips away boilerplate while retaining the original semantics will allow longer formulas to work without truncating them. For this work, we used a window of 1,000 tokens, which was sufficient to capture the majority of the sequents and lemmas in our experiments. Given this, all symbol names for functions and operators are copied over as-is. Constant and variable names are replaced by a placeholder, to generalize the model, while integer values are retained. Syntactic constructs such as parentheses are excluded as the ordering of the above can roughly capture the syntactic arrangement of the original form. Figure 2 provides an example: Symbols such as the FORALL quantifier and implication operator are preserved, while the variables F, HIGH, and LOW are replaced with their type, NAT representing the natural numbers.

Following common practice in transformer-based encodings, we use special tokens <ANT>, <CONS>, and <HID> to delimit the antecedent, consequent, and hidden formulas (formulas reserved from being operated on by PVS commands). The lefthand side of Figure 3 gives an example of a featurized sequent with no antecedents and one consequent.

Our current setup makes the weak Markov assumption; only the current state is sufficient for making our predictions. At least anecdotally knowing which commands were performed can inform what steps are taken next, so incorporating previous commands can capture some non-Markovian information. This is done by prefixing the state representation with the previous three commands issued by the user.

For the lemma retrieval experiments, we modified the above procedure to allow constant and variables to be replaced with their type name. This was done to allow matching by type, as arguments for imported lemmas also need to match by type. Higher-order and custom types are currently represented by placeholders. Accounting for matches on higher-order types and on advanced type operations such as predicate-based approaches is reserved for future work.

As with other transformers-based works, the model is first trained using a series of self-supervised tasks, where supervised targets are generated from unabeled data. Masked language modeling is one such task, where random tokens are masked and the model is trained to predict its identity [8]. By conducting this type of self-supervised training on a large corpus, the resulting representations can capture distributional information about the domain that makes training downstream components easier. For our experiments, the language model was trained for $1,100,000$ steps[2] over our dataset, using the default set of self-supervised language tasks used by RoBERTa.

We note that some works start from a model trained on natural language, in order to capture correspondences based on human naming. In our experience this applies for tasks where wider distributional knowledge of natural language is required to perform the main task. At least for PVS and our tasks, the structure of the formulas tends to be more important. Examination of the effect of human language understanding is also reserved for future investigation.

## 4 Command Prediction

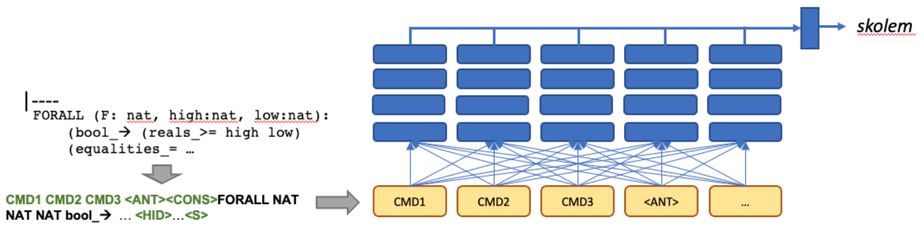

skolem
|----
FORALL (F: nat, high:nat, low:nat):
(bool_→ (reals_>= high low)
(equalities_= …
CMD1 CMD2 CMD3 <ANT><CONS>FORALL NAT
NAT NAT bool_→ … <HID>…<S>
CMD1
CMD2
CMD3
<ANT>
…

Fig. 3: The process for featurizing a sequent and then using repeated self-attention to create representations capturing information for predicting the next command.

Command prediction's task is to predict the command an expert would take given the current step in the proof. We use the T5 sequence to sequence training framework [39] implemented in Huggingface [49], with the RoBERTa encoding of the proof state used as input to predict the user selected command (Figure 3. The sequent and command history are tokenized and converted into classification-suitable vector representations via repeated applications of self-attention. These are then integrated by the classifier to emit predicted command. The top-$N$ most confident hypotheses can be emitted, allowing for a window of predictions to be

[2] A step is a single forward-predict pass over a training instance, and multiple steps can be performed over the same data during the training phase.

generated. We note that as with other ITPs, PVS allows users to program their own commands. For this work, we focus on the closed set of existing commands, leaving the program synthesis aspect for future work.

## 5 Lemma Retrieval

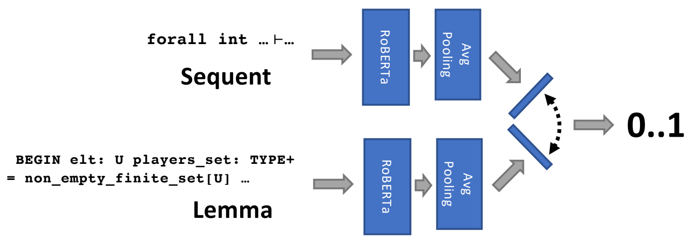


Fig. 4: Siamese architecture used to determine whether a lemma is relevant to a given sequent.

Following the information retrieval approach, the proof state acts as a query against a library of available lemmas. We employ user-imported lemmas in PVS-lib proof traces to train a neural information retrieval model [35], which learns the best combination of features between sequents and lemmas to assess lemma relevance. Figure 4 our lemma retrieval approach, which uses a Siamese Network [6] implemented in the SBERT [40] framework to score the relevance of a lemma to the sequent. The lemma and sequent token sequences are encoded using RoBERTa encoder to construct token-level representations that are averaged to give a single characterizing vector. The relevance of the lemma and sequent vectors is scored using cosine similarity, with 0 indicating no relevance and 1 indicating maximal relevance. The representation is tuned for the similarity task with supervised training over known relevant and irrelevant pairs. This approach scales well as the bulk of the representations can be pre-computed. In addition, this has been used to learn ranking functions for tasks with a large amount of data, such as using clickthrough data [18].

## 6 Experiments and Results

### 6.1 Command Prediction

From the full PVSLib library of proofs, we subsampled 20,000 proof steps to create a tractable command prediction training set [3]. From these we randomly

[3] Initial experiments with larger samples showed no difference in performance with a system trained with the smaller set

sampled 90% of these for the training data, and used the remaining 10% as a held out test-set. We trained for 10 epochs on four NVIDIA GeForce RTX 3090 cards using distributed data parallel training (DDP) [28] implemented using PyTorch Lightning [4], selecting the model with the best validation error. We follow prior literature on tactic prediction [13] and used classifiers trained over term-frequency inverse document frequency (TF-IDF) [23,32] weighted feature counts of the CoProver featurized tokenization observed in the sequent for our baseline. TF-IDF incorporates frequency of occurrence of a term and its distinguishability against the backdrop of the entire collection. We experimented with multiple classifiers to strengthen this baseline: Linear support vector classifier (Linear SVC), support vector machines using a radial basis kernel (RBF) and one using a polynomial kernel (Poly), and a k-nearest neighbor classifier (k-NN). We used the Scikit-Learn[5] implementations with default parameters. For the k-nearest neighbor classifier, we used a distance weighted variant with $n = 5$ following prior literature [13].

Table 1: Command prediction test accuracies by method and combinations of sequent and command history information.

| **Method** | **Acc. cmdhist + sequent** | **Acc., sequent only** | **Acc., cmdhist only** |
|---|---|---|---|
| Linear SVC | $0.30 \pm 1.1 \times 10^{-2}$ | $0.20 \pm 9.1 \times 10^{-3}$ | $0.30 \pm 1.1 \times 10-2$ |
| SVM (RBF) | $0.29 \pm 1.0 \times 10^{-2}$ | $0.22 \pm 9.5 \times 10^{-3}$ | $0.30 \pm 1.0 \times 10^{-2}$ |
| SVM (Poly) | $0.20 \pm 8.9 \times 10^{-3}$ | $0.20 \pm 8.9 \times 10^{-3}$ | $0.22 \pm 1.0 \times 10^{-2}$ |
| k-NN | $0.28 \pm 1.0 \times 10^{-2}$ | $0.19 \pm 8.6 \times 10^{-3}$ | $0.27 \pm 9.6 \times 10^{-3}$ |
| CoProver | $\mathbf{0.48 \pm 7.3 \times 10^{-3}}$ | $0.28 \pm 9.8 \times 10^{-3}$ | $0.21 \pm 9.3 \times 10^{-3}$ |

Table 1 shows the test command predication accuracy for each of the methods on different combinations of the sequent and the command history. We find that CoProver predictions are more significantly more accurate when the full sequent and command histories are used. Most of baseline performance is from the command history, whereas CoProver is able to integrate the sequent and command history together to score significantly better than the next-best baseline, k-NN. Variances for each method were estimated using bootstrap resampling [9] and significance was determined using a two-sample $t$-test with $\alpha = 0.001$.

To assess the significance of structural information, we tested with TF-IDF sequent featurizations of increasing maximum $n$-gram degree, where a $n$-gram featurization consists of all symbol sequences of length $n$. Table 2 shows the accuracies for each classification method by the maximum $n$-gram degree. With the exception of the SVM using the polynomial kernel (SVM Poly), every method benefits from increasing structural information. We suspect that model's poorer performance may be due to the greater number of hyperparameters given

[4] https://www.pytorchlightning.ai/
[5] https://scikit-learn.org/

Table 2: Command prediction accuracy for baseline methods using features that incorporate more structural information (left to right).

| **Method** | **n**=1 | **n**=2 | **n**=3 |
|---|---|---|---|
| Linear SVC | 0.30 | 0.37 | 0.30 |
| SVM (RBF) | 0.29 | 0.32 | 0.33 |
| SVM (Poly) | 0.20 | 0.18 | 0.19 |
| k-NN | 0.28 | 0.30 | 0.32 |

the polynomial kernel, which greatly increases the risk of overfitting on sparse data [16].

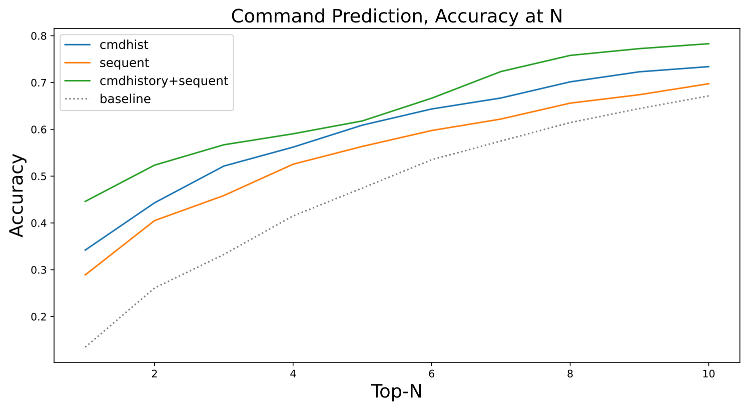


Fig. 5: Command prediction test accuracies by method, with and without command history information.

Recommender systems often present the top $N$-most relevant predictions, as users can usually scan a set of candidates. To assess performance in this regime, we score the top-$N$ test set accuracy, where matches are made if the correct prediction is within the top $N$ predictions. Figure 5 shows CoProver accuracy at different sizes of $N$ using just the command history (`cmdhist`), sequent (`sequent`) and both (`cmdhist+sequent`) for $N$ ranging from 1 to 10. As baseline we use the top-$N$ most frequent commands in the training set as the candidate window. We find combining the sequent and command history information gives consistently higher accuracy than using either alone, while all methods outperform the baseline.

## 6.2 Lemma Retrieval

For lemma retrieval, we examined PVSLib traces where users imported lemmas using the `lemma` command. While other commands also add lemmas, we focus on the explicit import action for this work. From these $20,221$ imports, $12,132$ were randomly selected for training, with $8,089$ for testing. These reference both PVSLib and PVS Prelude, giving $9,468$ candidate lemmas.

This model is trained on `lemma` commands in the training set of proof sequences in the PVSLib. For each `lemma` invocation, we record the sequent at that point in the proof and the name of the referenced lemma. PVSLib has $20,221$ such pairs in its proof traces, consisting of the sequent state when the `lemma` command was entered by the user and the name of the lemma. We randomly split them into $12,132$ train and $8,089$ test pairs. This is against a combined library of PVS and PVSLib theories, with a total of 747 theories and $9,468$ available lemmas. For training the Siamese network, observed sequents and lemmas in the training invocations have a score of 1. An equal number of negative sequent and lemma pairs were sampled, with their score set to 0, as a randomly selected lemma is very unlikely to be relevant.

We evaluated the resulting network on the test pairs and measured performance using mean reciprocal rank (MRR), an IR metric that assesses relevance-ranking ability [33]. MRR is computed from the rank position of the ground truth lemma in the relevance-ordered scores given the sequent (Formula 1), with higher values indicating better ranking ability.

$$MRR = \frac{1}{N} \sum_{i}^{N} \frac{1}{r_i} \tag{1}$$

For a given test pair consisting of a sequent $s$, selected lemma $l_{gt}$ and library of lemmas $L$ with $l_{gt} \in L$, we score $f_{rel}(s, l)$, the relevance of lemma $l \in L$ to the sequent. We derive a rank ordering over all lemmas $L$, where $r_i$ is the rank of the ground truth lemma for the $i^{th}$ lemma pair.

| **Method** | **MRR** |
|---|---|
| Baseline | 0.0015 |
| Count | 0.0030 |
| TF-IDF | 0.043 |
| CoProver | **0.51** |

We find that the CoProver approach to outperform the other methods including representation used by previous work [5,12]. A MRR of 0.51 corresponds to an average mean rank of 1.98, which corresponds to the relevant lemma appearing around position 2 in a score-based rank ordering of all lemmas.

## 7 Related Work

In the last decade, there has been a significant amount of activity in applying machine learning to automated deduction, and can be classified in terms of the predictive goal of machine learning [38]:

1. Learning search heuristics (E-prover [41], SAT/SMT solvers [4]): The systems ENIGMA [20], MaLeCoP [45] and FEMaLeCoP [24] augment a tableau-based prover LeanCoP [36] with a naïve Bayes classifier for clause selection. Graph neural nets have been used to predict the clauses that are in the unsatisfiable core set of clauses from a clause set [42] and to guide SMT solvers [2].
2. Premise selection from a library of facts (DeepMath [19], CoqHammer, HOLy-Hammer, HOList [3], Thor [21]): Several proof assistants invoke hammers (theorem provers and SAT/SMT solvers) on each subgoal together with a set of background lemmas (the premises). These hammers can fail if there are too many premises. Machine learning has been used to identify the most promising premises to pick from the background library [26]. As with tactic selection below, a range of learning techniques have been employed for premise selection, including sequence, tree, and graph representations [47].
3. Step or tactic selection (GPT-f, Holophrasm [48], CoqGym [51], HOL4RL [50], HOList [31], GamePad [17], Tactic-Toe [13], proof synthesis [10,22,27]): Interactive proof assistants build proof trees by applying tactics to goals to generate zero or more subgoals. The SEPIA system [15] predicts tactics for the Coq proof assistant based purely on analyzing proofs. GPT-f uses the GPT-3 transformer model to train on Goal/Proof pairs from the Metamath corpus (augmented with synthesized proofs) to predict the proof given the goal. This is used to construct a proof tree by applying the proof steps suggested by the model to the open subgoals in the tree. The system was able to find shorter proofs for 23 theorems in the Metamath corpus. CoqGym [51] uses a much larger training corpus spanning 71,000 proofs from various Coq libraries. TacticToe [14] is trained on proofs from HOL4 libraries and combines tactic prediction using k-nearest neighbors with A* search, and in some cases yields better (more perspicuous and maintainable) proofs than the alternatives using Hammers. IsarStep [29] uses a transformer encoding to identify intermediate formulas in a declarative Isar proof. ML4PG [25] extracts useful statistical patterns from higher-order logic proofs using unsupervised learning techniques like K-Means clustering.

Prior work in step selection focused on the ability of the system to fully automate the proof, with performance was measured in number of proofs that can be automatically completed. CoProver's focus is on the ability of the system to capture human-selected proof steps. Prior work in this area featurized sequents as histograms of tokens. As we have shown, structural information matters, and using neural language modeling technology captures this over wider portions of the formulas. Premise selection has been a topic of investigation, albeit focused on selection of useful premises for application of hammers and evaluating based on automatic completion [1,34,47]. In contrast, we focus on a broader use case, retrieving useful recommendations that can progress the proof via additional user interactions as well as application of hammers. Treating lemma retrieval as an information retrieval problem has been done in prior work [5,13], which used term-weighted histograms of the sequent (query) and lemma (document) for comparison. While this "bag-of-words" approach removes sequentiality and thus structural information, vocabulary overlap between query and candidate document is a good approximator for relevance. However, bag-of-words modeling and the TF-IDF weighting scheme are assumptions targeting how relevance appears for natural language queries and documents. Logical formula observed in lemmas and sequents may not exhibit the same behavior, particularly for determining if a lemma is relevant to moving proof progress in a sequent. Indeed, neural information retrieval has focused on using supervised queries and document pairs to learn relevance functions that may not be captured by assumptions taken in standard IR modeling. Here, the Magnushammer approach is similar to ours [34], leveraging a Transformer-based architecture to encode the string representation of the proof-state and the lemma. However, we include variable type information into the encodings and conduct an evaluation focusing on retrieval quality.

## 8 Conclusions

In this work, we have demonstrated how a simple featurization of proof state can be used to perform two recommendation tasks, predicting next commands and retrieving relevant lemmas. For command prediction, CoProver's approach has been shown to outperform prior methods, giving significantly higher accuracies. In the context of recommendation systems, showing the top $3-5$ commands is a reasonable amount, with these windows capturing $50\% - 70\%$ of the original prediction correctly on the validation set. As with systems trained on user interaction traces, there are often cases where the system can learn a solution that the user did not consider. As one internal user commented, the application of an automated hammer (the `grind` command) in a convergence proof was unexpected, but lead to completion of the proof. Similarly, using a neural learning approach with CoProver's featurization can give significantly better performance on lemma retrieval, in comparison with retrieval using IR-derived baselines.

In spite of these results, we note that the neural learning mechanism are not necessarily learning deep reasoning structures, and may more likely be learning complex structural cues. Indeed, an analysis of large language models found them to be impressive memorization machines that are incapable of performing arithmetic [7]. A cursory examination of the attention heads in the command prediction task revealed the model's attention weightings did not consistently align with experienced users' intuitions about what should govern the direction of the proof.

Future directions of proof command recommendation include identification of arguments used for these commands. These primarily consist of the formula to use, but in some cases more complex arguments are needed. Of particular interest is pairing this capability with explanation mechanisms. Perhaps the simplest explanation capability is to run the top-$N$ commands in the background and displaying the results provides a look-ahead capability that allows users to see the envelope of outcomes. When paired with heuristics that measure proof completion, this may be beneficial for developing an intuitive understanding.

To the best of our knowledge, we are the first work to treat lemma retrieval as an information retrieval problem. Previous work focused on if a selected lemma can progress a proof towards completion in an automated solver. For the type of problems addressed by interactive theorem provers, automatic completion may not be feasible in all cases. Importing a lemma to progress the proof becomes useful, similar to how retrieving the right document can help a querying user perform a task. To that end, we have demonstrated how a neurally trained architecture can determine which lemma an expert user would have selected. This approach can provide a better relevance ranking for lemmas, as opposed to the representation and scoring methods discussed in previous work. We note that setup only considers lemmas selected by the user as relevant for a given sequent. This disregards the possibility that another lemma may be just as useful for the proof as well. This is a well known issue in natural language information retrieval corpora, and thus measures like MRR are used more to compare system performance as opposed to acting as a standalone performance measure.

Possible future work in this area can focus on analyzing relevant structural elements that trigger a match between a sequent and a lemma. While the final comparison is performed using a cosine similarity computation, the nature of the highest scoring feature matches can be hard to discern. In particular, it is possible that the formula for the lemma and sequent may not have much apparent overlap, but relevant token sequences may map to the same feature.

For future work, we are examining the application of CoProver's transformer based sequent representation towards tasks such as nominating witnesses, proof repair, and developing a measure for proof progress. The code and the data used for this work are open-sourced and will be available at `https://github.com/SRI-CSL/coproof`.

**Acknowledgements** This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR00112290064 and by the National Institute of Aeronautics. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA.

## References

1. Alama, J., Kühlwein, D., Tsivtsivadze, E., Urban, J., Heskes, T.: Premise selection for mathematics by corpus analysis and kernel methods. CoRR **abs/1108.3446** (2011), `http://arxiv.org/abs/1108.3446`
2. Balunovic, M., Bielik, P., Vechev, M.T.: Learning to solve SMT formulas. In: NeurIPS. pp. 10338–10349 (2018)
3. Bansal, K., Loos, S., Rabe, M., Szegedy, C., Wilcox, S.: Holist: An environment for machine learning of higher order logic theorem proving. In: International Conference on Machine Learning. pp. 454–463. PMLR (2019)

4. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press (2009)
5. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. J. Autom. Reason. **57**(3), 219–244 (2016). https://doi.org/10.1007/s10817-016-9362-8, `https://doi.org/10.1007/s10817-016-9362-8`
6. Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., Shah, R.: Signature verification using a "siamese" time delay neural network. Advances in neural information processing systems **6** (1993)
7. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners. CoRR **abs/2005.14165** (2020), `https://arxiv.org/abs/2005.14165`
8. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota (Jun 2019). https://doi.org/10.18653/v1/N19-1423, `https://aclanthology.org/N19-1423`
9. Efron, B., Tibshirani, R.J.: An Introduction to the Bootstrap. No. 57 in Monographs on Statistics and Applied Probability, Chapman & Hall/CRC, Boca Raton, Florida, USA (1993)
10. First, E., Brun, Y., Guha, A.: Tactok: semantics-aware proof synthesis. Proceedings of the ACM on Programming Languages **4**(OOPSLA), 1–31 (2020)
11. Gage, P.: A new algorithm for data compression. C Users J. **12**(2), 23–38 (feb 1994)
12. Gauthier, T., Kaliszyk, C., Urban, J.: Learning to reason with hol4 tactics. arXiv preprint arXiv:1804.00595 (2018)
13. Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., Norrish, M.: Learning to prove with tactics. CoRR **abs/1804.00596** (2018), `http://arxiv.org/abs/1804.00596`
14. Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., Norrish, M.: Tactictoe: Learning to prove with tactics. Journal of Automated Reasoning **65**(2), 257–286 (2021)
15. Gransden, T., Walkinshaw, N., Raman, R.: Sepia: search for proofs using inferred automata. In: International Conference on Automated Deduction. pp. 246–255. Springer (2015)
16. Hsu, C.W., Chang, C.C., Lin, C.J.: A practical guide to support vector classification. Tech. rep., Department of Computer Science, National Taiwan University (2003), `http://www.csie.ntu.edu.tw/~cjlin/papers.html`
17. Huang, D., Dhariwal, P., Song, D., Sutskever, I.: Gamepad: A learning environment for theorem proving. arXiv preprint arXiv:1806.00608 (2018)
18. Huang, P.S., He, X., Gao, J., Deng, L., Acero, A., Heck, L.: Learning deep structured semantic models for web search using clickthrough data. In: Proceedings of the 22nd ACM International Conference on Information and Knowledge Management. p. 2333–2338. CIKM '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2505515.2505665, `https://doi.org/10.1145/2505515.2505665`

19. Irving, G., Szegedy, C., Alemi, A.A., Eén, N., Chollet, F., Urban, J.: Deepmath-deep sequence models for premise selection. Advances in Neural Information Processing Systems **29**, 2235–2243 (2016)
20. Jakubuv, J., Urban, J.: Enigma: efficient learning-based inference guiding machine. In: International Conference on Intelligent Computer Mathematics. pp. 292–302. Springer (2017)
21. Jiang, A.Q., Li, W., Tworkowski, S., Czechowski, K., Odrzygóźdź, T., Miłoś, P., Wu, Y., Jamnik, M.: Thor: Wielding hammers to integrate language models and automated theorem provers (2022). https://doi.org/10.48550/ARXIV.2205.10893, https://arxiv.org/abs/2205.10893
22. Jiang, A.Q., Welleck, S., Zhou, J.P., Li, W., Liu, J., Jamnik, M., Lacroix, T., Wu, Y., Lample, G.: Draft, sketch, and prove: Guiding formal theorem provers with informal proofs (2022). https://doi.org/10.48550/ARXIV.2210.12283, https://arxiv.org/abs/2210.12283
23. Jones, K.S.: A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation **28**, 11–21 (1972)
24. Kaliszyk, C., Urban, J.: Femalecop: Fairly efficient machine learning connection prover. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 88–96. Springer (2015)
25. Komendantskaya, E., Heras, J., Grov, G.: Machine learning in proof general: Interfacing interfaces. arXiv preprint arXiv:1212.3618 (2012)
26. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: Mash: machine learning for sledgehammer. In: International Conference on Interactive Theorem Proving. pp. 35–50. Springer (2013)
27. Lample, G., Lachaux, M.A., Lavril, T., Martinet, X., Hayat, A., Ebner, G., Rodriguez, A., Lacroix, T.: Hypertree proof search for neural theorem proving (2022). https://doi.org/10.48550/ARXIV.2205.11491, https://arxiv.org/abs/2205.11491
28. Li, S., Zhao, Y., Varma, R., Salpekar, O., Noordhuis, P., Li, T., Paszke, A., Smith, J., Vaughan, B., Damania, P., Chintala, S.: PyTorch Distributed: Experiences on accelerating data parallel training. CoRR **abs/2006.15704** (2020), https://arxiv.org/abs/2006.15704
29. Li, W., Yu, L., Wu, Y., Paulson, L.C.: Isarstep: a benchmark for high-level mathematical reasoning. arXiv preprint arXiv:2006.09265 (2020)
30. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A robustly optimized BERT pretraining approach (2019). https://doi.org/10.48550/ARXIV.1907.11692, https://arxiv.org/abs/1907.11692
31. Loos, S., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. arXiv preprint arXiv:1701.06972 (2017)
32. Luhn, H.P.: A statistical approach to mechanized encoding and searching of literary information. IBM Journal of Research and Development **1**(4), 309–317 (1957). https://doi.org/10.1147/rd.14.0309
33. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge, UK (2008), http://nlp.stanford.edu/IR-book/information-retrieval-book.html
34. Mikuła, M., Antoniak, S., Tworkowski, S., Jiang, A.Q., Zhou, J.P., Szegedy, C., Łukasz Kuciński, Miłoś, P., Wu, Y.: Magnushammer: A transformer-based approach to premise selection (2023)
35. Mitra, B., Craswell, N.: (2018)

36. Otten, J., Bibel, W.: leancop: lean connection-based theorem proving. Journal of Symbolic Computation **36**(1-2), 139–161 (2003)
37. Owre, S., Rushby, J.M., Shankar, N.: PVS: A prototype verification system. In: International Conference on Automated Deduction. pp. 748–752. Springer (1992)
38. Rabe, M.N., Szegedy, C.: Towards the automatic mathematician. In: International Conference on Automated Deduction. pp. 25–37. Springer, Cham (2021)
39. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. CoRR **abs/1910.10683** (2019), `http://arxiv.org/abs/1910.10683`
40. Reimers, N., Gurevych, I.: Sentence-BERT: Sentence embeddings using Siamese BERT-Networks. CoRR **abs/1908.10084** (2019), `http://arxiv.org/abs/1908.10084`
41. Schulz, S.: E – A Brainiac Theorem Prover. Journal of AI Communications **15**(2/3), 111–126 (2002)
42. Selsam, D., Bjørner, N.: Guiding high-performance SAT solvers with unsat-core predictions. In: International Conference on Theory and Applications of Satisfiability Testing. pp. 336–353. Springer (2019)
43. Sennrich, R., Haddow, B., Birch, A.: Neural machine translation of rare words with subword units. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1715–1725. Association for Computational Linguistics, Berlin, Germany (Aug 2016). https://doi.org/10.18653/v1/P16-1162, `https://aclanthology.org/P16-1162`
44. Shankar, N.: Automated reasoning, fast and slow. In: Proceedings of the 24th international conference on Automated Deduction. pp. 145–161. CADE'13, Springer-Verlag, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38574-$2_10$, `http://dx.doi.org/10.1007/978-3-642-38574-2_10`
45. Urban, J., Vyskočil, J., Štěpánek, P.: Malecop machine learning connection prover. In: International Conference on Automated Reasoning with Analytic Tableaux and Related Methods. pp. 263–277. Springer (2011)
46. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
47. Wang, M., Tang, Y., Wang, J., Deng, J.: Premise selection for theorem proving by deep graph embedding. arXiv preprint arXiv:1709.09994 (2017)
48. Whalen, D.: Holophrasm: a neural automated theorem prover for higher-order logic. arXiv preprint arXiv:1608.02644 (2016)
49. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T.L., Gugger, S., Drame, M., Lhoest, Q., Rush, A.M.: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45. Association for Computational Linguistics, Online (Oct 2020), `https://www.aclweb.org/anthology/2020.emnlp-demos.6`
50. Wu, M., Norrish, M., Walder, C., Dezfouli, A.: Tacticzero: Learning to prove theorems from scratch with deep reinforcement learning. arXiv preprint arXiv:2102.09756 (2021)
51. Yang, K., Deng, J.: Learning to prove theorems via interacting with proof assistants. In: International Conference on Machine Learning. pp. 6984–6994. PMLR (2019)