Using LSTM to Predict Tactics in Coq

Xiaokun Luan, Xiyue Zhang, Meng Sun
School of Mathematical Sciences, Peking University, Beijing, China
{luanxiaokun,zhangxiyue,sunm}@pku.edu.cn

***Abstract*—Quality assurance of rapidly evolving systems is increasingly important for their deployment to real-life applications. Despite the challenges posed by the increasing complexity of these systems, various techniques have been developed to check their correctness, such as theorem proving, which is a powerful formal verification method that can provide a complete guarantee. However, the proving process in the interactive theorem provers like Coq highly relies on human interactions, making the proving process difficult and time-consuming. To automate the proving process in Coq, we present a framework for predicting tactics in Coq by using Long Short Term Memory (LSTM). We take into account the effect of the dataset proof style on machine learning and create a new dataset following a specific proof style. We use the generated data to train an LSTM-based neural network that could give tactic predictions based on the proof context. This neural network reaches an accuracy of 58% if we only use the first predicted tactic and reaches an accuracy of 87% if we select the first three tactic suggestions, achieving a 15.2% and 12.8% improvement rate, respectively, compared to the methods in previous work.**

I. Introduction

In the last few decades, our reliance on software systems has rapidly grown. Quality assurance of such systems is thus necessary and crucial for their deployment to real-life applications. Despite the challenges posed by the increasing complexity of these systems, many techniques have been developed to check their functional correctness, e.g., testing and verification. Compared with testing techniques, formal verification could provide a complete guarantee of the critical properties of software systems. Theorem proving is one of the most popular formal verification methods, where systems are modeled in an appropriate mathematical logic, and critical properties are represented as propositions to be proved and verified in *theorem provers*. Up to present, theorem proving has been successfully applied to various domains such as computer science [1], artificial intelligence [2], economy [3], biomedical [4] and self-adaptive systems [5].

Theorem provers are mainly categorized into two types: Automated Theorem Provers (ATPs) such as Alt-Ergo [6], and SPASS[7], and Interactive Theorem Provers (ITPs) such as Coq [8], Isabella [9], and PVS [10]. Though the reasoning process is automated in ATPs, they usually suffer from complexity and expressive power problems. In contrast, the expressive power of ITPs are usually stronger, which makes them more suitable for the formalization of "most non-trivial theorems in mathematics or computer system correctness" [11]. However, they require human interaction with computer in the process of proof construction, which is the reason that they are also called *proof assistants*. Coq is one of the interactive theorem provers. It allows users to declare propositions and prove them. When users are proving propositions, they are actually constructing proof terms with the help of commands called *tactics*. This process of constructing proofs highly relies on human interactions, which can make the proving process difficult and time-consuming. It is quite often that some intuitively obvious facts require tedious proofs in Coq. In general, proving process in most ITPs is labor-intensive due to the lack of automation.

Many efforts have been made to improve the degree of automation of interactive theorem provers. For example, the Coq team provided *Ltac* [12] to support custom tactics and a set of automatic tactics like *auto* and *congruence* that realize partial automation in certain domains. Recently machine learning techniques have been investigated to automate the proving process. [13] presented ML4PG that gathers data from a general proof interface *ProofGeneral*, and used clustering algorithms to learn and predict dependencies of goals. [14] tried several machine learning techniques to learn proof dependencies from formalization done with Coq system. TacticToe was developed in [15] for HOL4 theorem prover, which implements a modified A^*-algorithm to automate the tactic selection in proof search. [16] presented a reinforcement learning environment for theorem proving and a deep learning driven automated theorem prover for higher-order logic. [17] leveraged Recurrent Neural Network (RNN) to predict Coq tactics for property verification in the domain of coordination language. [18] used k-nearest neighbor algorithm to learn from previous proof scripts for tactic proof search in Coq. [16], [17], and [18] all work on tactic level learning, through which the predicted tactics can be directly applied to proceed the proof and custom tactics can also be supported. However, the focus of most existing works has been more on the modeling design than on the dataset construction, where the datasets are mostly directly extracted from raw proof scripts such as standard Coq libraries. We need to put more efforts on the construction of dataset to obtain higher quality data so that, combined with an effective learning model, we can achieve better automating performance.

In this paper we present a framework for predicting tactics in Coq using a Long Short Term Memory-based (LSTM-based) neural network [19]. We re-prove a theorem library following a specific proving style to create a new dataset, and use this dataset to train an LSTM neural network to learn and predict potential tactics with several hypotheses and a proof-goal as input. The contributions of our work can be summarized as follows:

1) We take a different approach to create training and test data. The specific proving style we adopted to re-prove

DOI reference number: 10.18293/SEKE2021-047

theorems makes the learning task more consistent and simple for LSTM-based networks.
2) We design an LSTM-based neural network to learn and predict tactics in Coq. This new architecture could stabilize hidden state dynamics and reduce overfitting, resulting in better performance for automating the proving process.
3) We perform experimental evaluation on the effectiveness of our method, which reaches an accuracy of 58% if we only use the first predicted tactic, and reaches an accuracy of 87% if we use the top three tactic suggestions, achieving a 15.2% and 12.8% improvement rate, respectively, compared to the baseline.

The rest of this paper is organized as follows: In Section II we briefly introduce how tactics work in Coq. The construction of our dataset and pre-processing steps are explained in Section III. The design of our LSTM-based neural network is elaborated in Section IV, and evaluation of our approach is provided in Section V. Finally, Section VI concludes this paper and discusses some future research directions.

II. Background

A. Tactics in Coq

As an interactive theorem prover, Coq allows users to declare and prove propositions and then to extract certified programs from the certified proofs. After declaring a proposition in Coq, users enter the proof mode. In this mode, the proposition to be proved is called a *goal*, and users can apply commands called *tactics* to decompose the goal into simpler *subgoal*s or to solve it directly. The decomposition process ends when there are no more subgoals. An *interaction* in Coq is a context-tactic pair as shown in Figure 1, where the *context* contains hypotheses we currently have and a set of subgoals we need to prove.

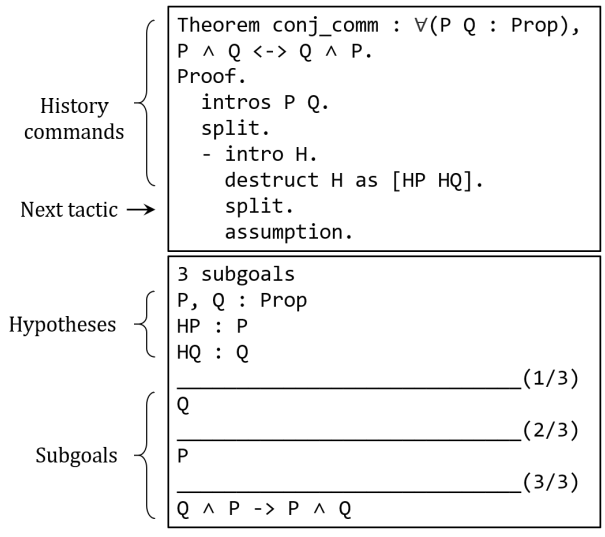

Fig. 1. An Interaction in Coq

Some tactics can be directly applied to the goal, others require arguments. For example, the `split` tactic can be directly applied to a conjunctive goal with no arguments, while the `apply` tactic means applying a known theorem to the goal and thus requires arguments, and the `intros` tactic can be used either with or without arguments, only differing on the names of the introduced hypotheses. In our framework, we only consider predicting tactic names, with tactic arguments excluded, since higher order logic is undecidable.

Different tactics may have the same effect on some goals. For example, when there is only one hypothesis to be introduced, the effects of the `intro` tactic and the `intros` tactic are the same. Besides, in most cases there is more than one way to construct the proof, that is to say, a proposition can be proved by different sequences of tactics, different users may have different proof styles of using tactics.

B. Assumptions

Several assumptions on proving process in Coq are made in [17] based on the observations and expert experience, which turns out to be effective in tactic prediction. We follow the assumptions (1-3) made in [17] in our framework. Besides that, we make an extra assumption (4). Basically, the first two assumptions allow us to use only hypotheses and the first subgoal to make predictions without considering all the subgoals. The third and the forth assumptions serve as the basis for data pre-processing and data augmentation respectively, which we will describe in more details later on. All the assumptions are summarized as follows:

1) Proofs of subgoals are *independent* of each other, which means that tactics to prove a subgoal do not depend on proofs of any other subgoals.
2) Rearranging the order of subgoals is not considered in our proving process, so the tactic we use and suggest is only applied to the first subgoal or current hypotheses.
3) When looking for a proper tactic to use, the *structure* of a Coq term is more important than its content.
4) Tactics can be applied to either a subgoal or one or more hypotheses.

III. Dataset and Data Pre-Processing

A. Dataset Construction

Using a neural network to predict appropriate tactics for proving process requires a dataset for training and testing. As we mentioned in the previous section, proofs in Coq can be written in different proof styles. There are good and bad proof styles for neural network learning. Although it is difficult to formally specify what a good proof style is, we provide several heuristics to distinguish a good proof style from a bad one. A good proof style for learning is supposed to be consistent, otherwise it can cause confusion for the learning process. For example, if we use different tactics in similar context, in other words, we follow an inconsistent proof style, then the minimization of loss function will be hindered, resulting in poor prediction performance. Besides, we should also consider the inherent learning difficulty of the proof style. A good proof style should not use overly complicated tactic mechanism, nor should it have tactics that are seldom used, so that it would be more simplified and consistent for a machine learning model to learn.

The standard Coq libraries constitute a large dataset for machine learning, many approaches are developed based on this dataset. However, according to the heuristics these theorem libraries' proof styles are not suitable for neural network learning for the following reasons:

1) Inconsistency: Since these libraries are developed by different authors, they usually have different proof styles. For example, *SSReflect*is a collection of libraries for the *SSReflect* [20] proof language and its proof style is quite different from the others.
2) Complexity: Advanced tactics are intensively used for conciseness and robustness, such as using tactical to combine several tactics as a compound tactic, which makes it more difficult to learn.
3) Infrequently used tactics: Some libraries define custom tactics to reduce repetition, but these tactics will neither be used in other libraries nor in practical proving process.

Therefore, we cannot utilize the standard Coq libraries as our dataset, instead we need to create a new dataset with a specific proof style that is not only consistent but also easy to learn.

Following the above heuristics, we manually create a theorem library about the properties of Reo [21] connectors in the domain of coordination language as our dataset. This domain-specific theorem library is constructed based on the rough proof scripts provided in [17]. The proof style we use when building the theorem library is quite like that of a novice, so we call this proof style *the novice proof style*, the benefits of which are summarized as follows:

1) Consistency. All proofs are written in a consistent style, we prioritize different tactics, so that for similar context we always use the same tactic to proceed the proof.
2) Simplicity. We only use one tactic at a time, and we avoid using unnecessary repeated tactics, for example, we use a single `intros` to introduce all the hypotheses and variables instead of a series of `intros` (Note that `intro` and `intros` are two different tactics with similar names and functions).
3) Restricted tactics. We restrict ourselves to a set of frequently used tactics to write proofs, including 23 tactics in total, as illustrated in Figure 2.

`apply`	`cut`	`reflexivity`
`assert`	`destruct`	`rewrite`
`assumption`	`exfalso`	`simpl`
`auto`	`exists`	`split`
`change`	`generalize`	`subst`
`clear`	`induction`	`symmetry`
`cofix`	`intro`	`unfold`
`constructor`	`intros`	

Fig. 2. Supported Tactics in Our Framework

When building our theorem library, we add a few new lemmas to complete the original proof. In the end, the constructed dataset contains 31 theorems and lemmas, all of which are fully proved, with a total of 830 lines of codes, while the original one contains 1 fully proved theorem and 9 partially proved theorems, with a total of 383 lines of codes.

B. Data Pre-Processing

We follow the pre-processing method in [17], which contains three steps as follows:

1) Use a Python script to write the Coq proof scripts line by line to SerAPI [22], which performs machine-to-machine interaction with Coq through S-expressions and extracts context-tactic pairs.
2) Refactor extracted Coq terms by adding corresponding term types as structural information based on the assumption (3).
3) Perform word encoding on hypotheses and the goal to get fixed-length vectors, since Coq programming language does not have finite dictionary nor semantics similarity. We fix a maximal length of words, and for each word we map its character to its ASCII code and fill the rest part with zero.
4) Apply one-hot encoding on tactic names to get one-hot vectors as sample labels.

However, we use different parameters for word encoding to eliminate redundant zeros. The purpose of this adjustment is to reduce input dimensions (dimensions of input are reduced from 5120 to 1360) and computational complexity. Since we use a restricted set of tactics, the dimension of the output is also reduced.

In the end, we obtain 526 samples from the constructed theorem library. These samples constitute a database in our learning framework, each of which is composed of a variable length sequence as input and a one-hot vector as output. The sequence consists of the first subgoal and several hypotheses if there is any, and the subgoal is always the last item in the sequence. The dataset in [17] contains 173 samples, but there are 12 samples labeled by the '`admit`' tactic, which cannot be used in a full proof, thus reducing the actual samples from 173 to 161.

IV. Learning Tactics Through LSTM

According to the assumptions, our problem can be regarded as a sequence classification problem, for which the most popular solution is RNNs. However, vanilla RNNs suffer from issues of vanishing gradient. To reduce impacts of the gradient vanish problem, LSTM (with cell memory and gate control) [19] is proposed and widely adopted to deal with tasks when long-term dependencies need to be captured. Therefore, we choose to build an LSTM-based neural network in our framework. The structure of our neural network is shown in Figure 3, including an *LSTM* layer, a *layer normalization* layer, a *dropout* layer and a *fully-connected* layer.

- LSTM Layer: The neural network has an LSTM layer containing 512 self-connected hidden units. The activation function is *tanh* and the gate activation function is *Sigmoid* function.
- Layer Normalization Layer: After the LSTM layer we add a layer normalization layer where the output of LSTM layer is normalized to zero mean and unit variance.
- Dropout Layer: Cells in this layer are randomly disconnected according to dropout rate (set as 0.5 in our framework) when training.

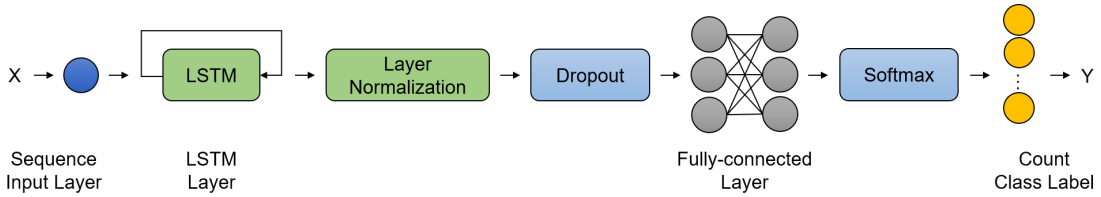

Fig. 3. Network Architecture

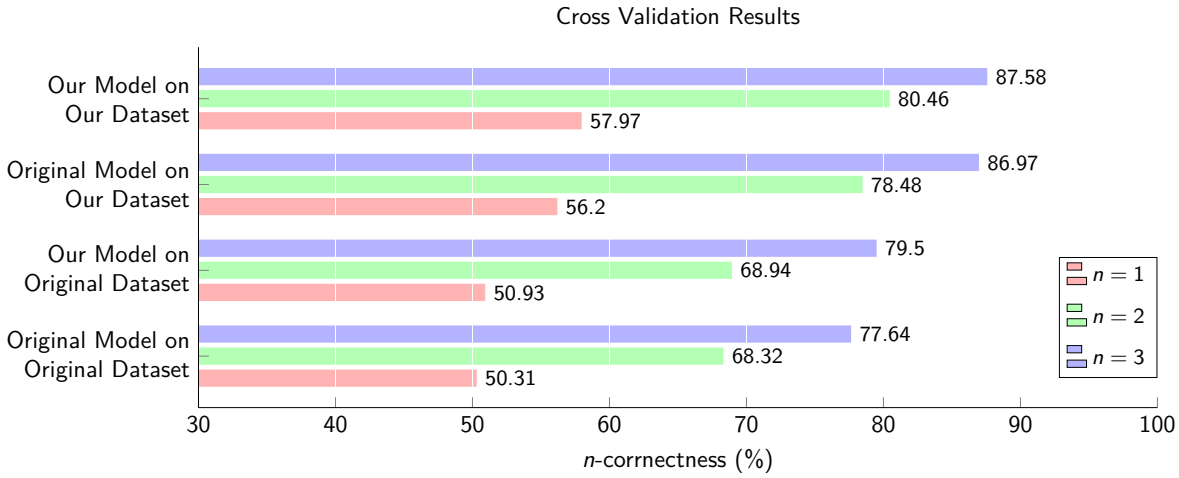

Fig. 4. Cross validation results of different models on original dataset and our dataset

- Fully-connected Layer: A fully-connected layer uses *softmax* as its activation function to normalize its output as a probability distribution.

The loss function of our neural network is *cross-entropy* function, which measures the difference between two probability distributions. For two probability distributions p and q, their cross-entropy is defined as:

$$H(p,q) = -\sum_{x} p(x) \cdot \log q(x)$$

The layer normalization [23] can stabilize hidden state dynamics for recurrent neural network and also help with reducing training time. The normalization is not implemented in unit-level but in layer-level, because the unit-level approach is much more complicated and more computational expensive but has roughly the same performance as layer-level approach.

In order to overcome the overfitting problem caused by the limited data and high input dimensions, we add the dropout layer. We also use label smoothing [24] technique, which is another frequently used regularization method. With label smoothing, the hard 0 and 1 classification targets in the ground truth one-hot vector $\mathbf{y}$ will be replaced with targets of $\frac{\varepsilon}{k-1}$ and $1-\varepsilon$ respectively, where ε is the smoothing parameter and k is the class number, thus prevents the pursuit of hard probabilities without discouraging correct classification.

We also use data augmentation to generate more training data to deal with the overfitting problem. In the pre-processing step, the context in an interaction is transformed into a sequence, where the last item is the goal to be proved and the others are hypotheses. Users who are familiar with Coq should be aware that the order of hypotheses is independent of the tactic that can be used. This fact inspires us to perform data augmentation by shuffling hypotheses. We perform shuffling on the goal and hypotheses together, since tactics can be applied to either a subgoal or any hypotheses according to assumption (4), which means that there is no essential difference between the goal and hypotheses for predicting tactics. Each sample sequence in the training set is shuffled 440 times in a way that every synthetic sample is different from other samples whenever possible. For those samples with too few hypotheses to get enough distinct synthetic samples (less than 5 hypotheses since $5! < 440 \leq 6!$), we use their full permutations as the generated samples.

In the training process, the neural network is trained for 30 epochs with a batch size of 256, and we use the RMSprop optimizer with learning rate set as 0.001, ρ set as 0.9, ε set as 10^{-7} and clipnorm set as 0.8. The label smoothing is 0.1. Initial kernel weights of LSTM layer are set by Glorot uniform initializer, initial recurrent weights of LSTM layer are set orthogonally, and initial bias is set as zero vector.

V. Evaluation

Our neural network is implemented in *Keras* [25], a high-level neural networks API in Python. Multiple popular machine learning frameworks are supported by Keras, and we use *Tensorflow* as its backend. We train the neural network on our dataset and [17]'s dataset separately. Experiments are run

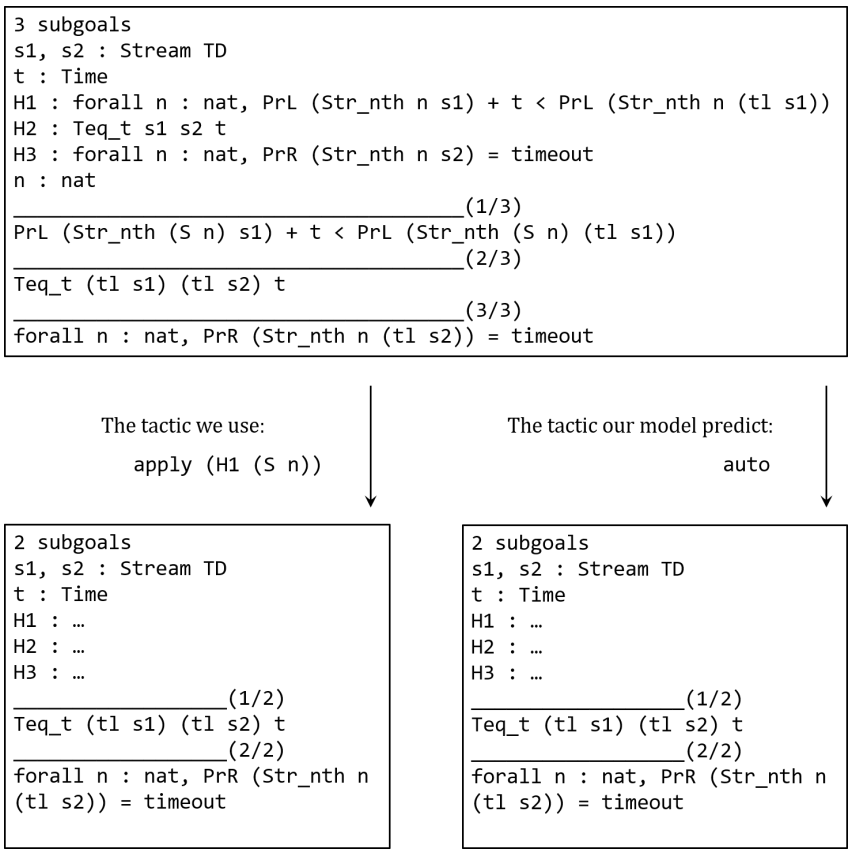

Fig. 5. The prediction for an validation sample and its true label

with an NVIDIA Tesla P100 GPU, 2 cores of Intel Xeon as CPU and 13 GB memory.

As there are often multiple tactics that can help with the proof process, the evaluation should not be constrained by a single correct answer provided by the proof scripts. Instead, we use *n-correctness rate* [17] to evaluate the tactic prediction performance. *n-correctness rate* measures the likelihood that the top *n* tactics predicted by the network are actually useful for theorem proving, which is defined as follows.

Definition 5.1 (n-correctness rate): The output of the network is a probability distribution, if the probability of the targeted tactic (provided by the dataset) is in top *n*, we say that this prediction is *n*-correct, and the corresponding correctness rate is called *n*-correctness rate.

We use cross validation to evaluate the tactic prediction performance of our approach, with the performance of the method in [17] as comparison baseline. We perform ablation experiments to evaluate the effectiveness of the constructed dataset and network design. Specifically, the original neural network is trained on our new dataset to evaluate the usefulness of the dataset. The proposed neural network is trained on the original dataset to evaluate the effectiveness of the network design. On our dataset, we evaluate neural network performance by repeating 10-fold cross validation 10 times. In other words, 10-fold cross validation procedure is repeated 10 times and the mean result across all runs is regarded as our final evaluation. While on the original dataset, we use leave-one-out cross validation to evaluate the performance, given that this dataset is smaller. Data augmentation is applied when training our model, but each sample in the original training set is shuffled 130 times, not 440 times. We use this evaluation method due to the limited size of the datasets, where the train-test split method may result in different distributions of training and test sets. The experiment results are shown in Figure 4.

Compared to baseline, our approach achieves a 15.2%, 17.8%, and 12.8% improvement rate on the 1-correctness, 2-correctness, and 3-correctness rate. With regard to the effectiveness of the proposed network design, on the original dataset the 1-correctness, 2-correctness, and 3-correctness rate of our neural network is 0.62%, 0.62%, and 1.86% higher than the original neural network, respectively. On our dataset, the improvements are 1.77%, 1.98%, and 0.61%, respectively. Regarding the usefulness of the constructed dataset, the 1-correctness, 2-correctness, and 3-correctness rate of the original neural network on our dataset is improved by 11.7%, 17.4%, and 12.0%, respectively, compared to the original dataset. As for our neural network, the improvement rates are 13.8%, 16.7%, 10.2%, respectively.

The design of our neural network, including the architecture and the use of regularization techniques, has led to improved performance. But the constructed dataset serves as a major factor contributing to the performance improvement, which significantly reduces the machine learning difficulty. In summary, our LSTM-based neural network outperforms the original neural network on both datasets, and our dataset is

easier for LSTM-based neural network to learn, both of which together make our approach perform better.

After taking a closer look at how our neural network performs on the validation set, we find an interesting phenomenon. On some samples, our neural network gives different predictions from the ground truth. However, these suggested tactics actually can solve the goal, which is exactly the situation we mentioned before. Figure 5 is an illustration for such phenomenon, where both the tactic we use and the prediction of our network can solve the goal. This interesting phenomenon indicates that our n-correctness-based evaluation of our model performance is very pessimistic, the actual performance should be even better.

VI. Conclusion

In this paper, we present a framework for predicting tactics to automate the process of proving properties of a specific domain in Coq. In order to automate the proving process in Coq, we create a new dataset by re-proving a theorem library used in [17] in the novice proof style, and train an LSTM-based neural network on this dataset to predict tactics based on proof context. Experiment results show that our approach to creating dataset makes the learning task easier for LSTM-based networks and that the proposed neural network outperforms the baseline, where the correctness of our network is almost 90% if we select first three suggested tactics. Besides, we find that our model is capable of giving suggestions which differ from ground truth in our dataset but can actually proceed the proof. This phenomenon indicates that the actual performance of the network is even better.

In the future, we plan to try reinforcement learning on this problem and try more loss function design for higher correctness rate. Gathering more data is another future work to improve network performance. Since manually building proofs is inefficient, we can try generating simple proofs from existing complex proofs.

Acknowledgments

This research was supported in part by the Guangdong Science and Technology Department (Grant No.2018B010107004); the National Natural Science Foundation of China under grant No.61772038, 61532019.

References

[1] J. H. Gallier, *Logic for Computer Science: Foundations of Automatic Theorem Proving*. USA: Harper & Row Publishers, Inc., 1985.

[2] H. Wang, *Computer Theorem Proving and Artificial Intelligence*. Dordrecht: Springer Netherlands, 1990, pp. 63–75.

[3] M. Kerber, C. Lange, and C. Rowat, "An introduction to mechanized reasoning," *Journal of Mathematical Economics*, vol. 66, pp. 26–39, 2016.

[4] A. Rashid, O. Hasan, U. Siddique, and S. Tahar, "Formal reasoning about systems biology using theorem proving," *PLOS ONE*, vol. 12, no. 7, pp. 1–27, 07 2017.

[5] D. Weyns, M. U. Iftikhar, D. G. de la Iglesia, and T. Ahmad, "A survey of formal methods in self-adaptive systems," in *Fifth International C* Conference on Computer Science & Software Engineering, C3S2E '12, Montreal, QC, Canada, June 27-29, 2012*. ACM, 2012, pp. 67–79.

[6] S. Conchon, A. Coquereau, M. Iguernlala, and A. Mebsout, "Alt-Ergo 2.2," in *SMT Workshop: International Workshop on Satisfiability Modulo Theories*, Oxford, United Kingdom, Jul. 2018.

[7] C. Weidenbach, D. Dimova, A. Fietzke, R. Kumar, M. Suda, and P. Wischnewski, "Spass version 3.5," in *Automated Deduction – CADE-22*, R. A. Schmidt, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 140–145.

[8] "Coq proof assistant," http://coq.inria.fr/.

[9] "Isabella, a generic proof assistant," https://isabelle.in.tum.de/.

[10] "PVS specification and verification system," https://pvs.csl.sri.com/.

[11] J. Harrison, J. Urban, and F. Wiedijk, "History of interactive theorem proving," in *Computational Logic*, ser. Handbook of the History of Logic, J. H. Siekmann, Ed. Elsevier, 2014, vol. 9, pp. 135–214.

[12] D. Delahaye, "A tactic language for the system coq," in *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1955. Springer, 2000, pp. 85–95.

[13] E. Komendantskaya, J. Heras, and G. Grov, "Machine learning in proof general: Interfacing interfaces," in *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, UITP 2012, Bremen, Germany, July 11th, 2012*, ser. EPTCS, vol. 118, 2012, pp. 15–41.

[14] C. Kaliszyk, L. Mamane, and J. Urban, "Machine learning of Coq proof guidance: First experiments," in *6th International Symposium on Symbolic Computation in Software Science, SCSS 2014, Gammarth, La Marsa, Tunisia, December 7-8, 2014*, ser. EPiC Series in Computing, vol. 30. EasyChair, 2014, pp. 27–34.

[15] T. Gauthier, C. Kaliszyk, and J. Urban, "Tactictoe: Learning to reason with HOL4 tactics," in *LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, ser. EPiC Series in Computing, T. Eiter and D. Sands, Eds., vol. 46. EasyChair, 2017, pp. 125–143. [Online]. Available: https://easychair.org/publications/paper/WsM

[16] K. Bansal, S. Loos, M. Rabe, C. Szegedy, and S. Wilcox, "HOList: An environment for machine learning of higher order logic theorem proving," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 454–463. [Online]. Available: http://proceedings.mlr.press/v97/bansal19a.html

[17] X. Zhang, Y. Li, W. Hong, and M. Sun, "Using recurrent neural network to predict tactics for proving component connector properties in Coq," in *2019 International Symposium on Theoretical Aspects of Software Engineering, TASE 2019, Guilin, China, July 29-31, 2019*. IEEE, 2019, pp. 107–112.

[18] L. Blaauwbroek, J. Urban, and H. Geuvers, "Tactic learning and proving for the coq proof assistant," in *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, ser. EPiC Series in Computing, E. Albert and L. Kovacs, Eds., vol. 73. EasyChair, 2020, pp. 138–150. [Online]. Available: https://easychair.org/publications/paper/JLdB

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] G. Gonthier, A. Mahboubi, and E. Tassi, "A Small Scale Reflection Extension for the Coq system," Inria Saclay Ile de France, Research Report RR-6455, 2016. [Online]. Available: https://hal.inria.fr/inria-00258384

[21] F. Arbab, "Reo: a channel-based coordination model for component composition," *Mathematical Structures in Computer Science*, vol. 14, no. 3, pp. 329–366, 2004.

[22] E. J. Gallego Arias, "SerAPI: Machine-Friendly, Data-Centric Serialization for Coq," MINES ParisTech, Tech. Rep., Oct. 2016. [Online]. Available: https://hal-mines-paristech.archives-ouvertes.fr/hal-01384408

[23] L. J. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *CoRR*, vol. abs/1607.06450, 2016.

[24] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2016, pp. 2818–2826. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2016.308

[25] F. Chollet *et al.*, "Keras," https://keras.io, 2015.