Neural Circuit Synthesis with Pre-trained Language Models

FREDERIK SCHMITT, CISPA Helmholtz Center for Information Security, Germany
MATTHIAS COSLER, CISPA Helmholtz Center for Information Security, Germany
BERND FINKBEINER, CISPA Helmholtz Center for Information Security, Germany

This extended abstract reports preliminary results on fine-tuning pre-trained language models for solving reactive synthesis problems end-to-end. In recent work, hierarchical Transformer neural networks have been successfully trained from scratch to synthesize sequential circuits directly out of formal specifications. We improve over existing approaches by fine-tuning CodeT5 models that have been pre-trained on both natural language and programming languages. Our experiments show improved generalization and sample efficiency compared to the previous approach.

1 INTRODUCTION

Reactive synthesis is the problem of automatically constructing a system that satisfies a formal specification (or proving that no such system exists). Starting from the 1960s [3] it has been an active research area as reactive synthesis simplifies the hardware design process to specifying what a system should do instead of how it does it. The wide usage of linear-time temporal logic (LTL) [17] to specify requirements in formal verification makes LTL synthesis of particular interest but at the same time suffers from an intractable doubly exponential complexity. Algorithmic advances such as game-based and bounded approaches lead to solvers for LTL synthesis [1, 6, 7, 10, 13, 14], with their performance being compared in the annual reactive synthesis competition SYNTCOMP [9].

Recently, algorithmic approaches have been complemented by deep learning methods, demonstrating that deep neural networks can be trained end-to-end to synthesize sequential circuits from formal specifications [4, 21]. Deep learning methods can solve unseen problems from the synthesis competition, out-of-distribution examples, and instances where classical tools time out. Although circuits generated by a neural network do not necessarily satisfy a given formal specification, they can be formally verified to guarantee correctness. Model-checking an LTL specification is computationally much cheaper than synthesizing an LTL specification (PSPACE vs. 2-EXPTIME), making this a viable approach.

In this work, we improve over the previous deep learning approach by fine-tuning pre-trained models instead of training from scratch. The process of fine-tuning language models has emerged as a successful approach in natural language processing that benefits from transfer learning and sample efficiency. We show that this holds true for the domain of LTL synthesis. We start by describing the previous deep learning approach to LTL synthesis in Section 2 and pre-trained language models in Section 3 before reporting experiments in Section 4.

Authors' addresses: Frederik Schmitt, frederik.schmitt@cispa.de, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany; Matthias Cosler, matthias.cosler@cispa.de, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany; Bernd Finkbeiner, finkbeiner.cispa.de, CISPA Helmholtz Center for Information Security, Saarbrücken, Germany.

2023. XXXX-XXXX/2023/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 NEURAL CIRCUIT SYNTHESIS

In previous work, a hierarchical Transformer neural network architecture was trained on synthesizing circuits in AIGER format [2] from formal specifications in LTL end-to-end [21].

To account for the lack of large amounts of natural training data, a data generation method is introduced based on benchmarks from the synthesis competition. From the set of benchmarks, assumption and guarantee patterns are collected and filtered to have at most five inputs, at most five outputs, and an abstract syntax tree of size at most 25. Given the set of specification patterns, the dataset is generated by alternating between sampling guarantees until the specification becomes unrealizable and sampling assumptions until the specification becomes realizable. Stopping criteria are implemented that limit the maximal number of guarantees, the maximal number of assumptions, and the runtime of the LTL synthesis tool Strix [14] that is used to synthesize the specifications. For comparison, we use the same dataset without modifications in this work.

The hierarchical Transformer architecture [11] trained on the dataset is an extension of the standard Transformer architecture [22] that has achieved state-of-the-art results on a wide range of natural language processing tasks and has become a foundational architecture in the field. The hierarchical version allows learning individual representations for assumptions and guarantees in a first local step before processing the whole specification in a following global step. In this work, we show that we can improve over existing results without explicitly building the hierarchical structure into the architecture when using pre-trained language models.

To evaluate performance, a syntactic and semantic accuracy is computed, measuring whether the predicted circuit matches the circuit in the test data (syntactic) or whether the circuit satisfies the specification (semantic). We use the same metrics in Section 4 for direct comparison.

3 PRE-TRAINED LANGUAGE MODELS

Transfer learning has become a powerful technique in natural language processing (NLP). First, a model is pre-trained on a more general data-rich task before being fine-tuned on a specific downstream task. The pre-trained model acts as a powerful weight initialization for the downstream task training (fine-tuning). Transfer learning often requires less training data on the downstream task resulting in faster training. Additionally, such models often show better generalization capabilities compared to regularly trained models.

Widely used general-purpose language models for transfer learning are GPT-2 [18], BERT [5], RoBERTa [12], and T5 [19]. While GPT-2 is a decoder-only Transformer model, BERT and RoBERTa are encoder-only Transformer models, and T5 is an encoder-decoder Transformer model. Following the success of such general-purpose language models on natural language, pre-training methods have

Table 1. Accuracy reported on test data, SYNTCOMP benchmarks, timeouts, and smart home benchmarks for different beam sizes. For the test data we show the syntactic accuracy in parenthesis.

Dataset	Model	Beam Size 1	Beam Size 4	Beam Size 8	Beam Size 16
`Testset`	Transformer [21]	53.6 (31.1)	70.4 (39.0)	75.8 (41.9)	79.9 (44.5)
	CodeT5	**58.9** (33.3)	**75.6** (36.7)	**80.5** (36.6)	**83.6** (39.5)
`SYNTCOMP`	Transformer [21]	**51.9**	60.0	63.6	66.8
	CodeT5	50.3	**60.7**	**66.2**	**71.0**
`Timeouts`	Transformer [21]	11.7	21.1	25.9	30.1
	CodeT5	**12.4**	**22.6**	**29.5**	**34.4**
`Smart Home`	Transformer [21]	22.9	31.4	**44.8**	40.0
	CodeT5	**31.8**	**40.9**	36.4	**54.5**

been successfully applied to programming languages. Prominent examples include CodeBert [8], CodeGen [15], and CodeT5 [23].

In this work, we base our fine-tuning on the LTL synthesis task on the pre-trained CodeT5 model class. Following the T5 architecture, CodeT5 is, in contrast to other mentioned NLP and code models, an encoder-decoder model, leveraging the duality of code and natural language. CodeT5 has been trained on code from various programming languages with accompanying documentation using multiple pre-training tasks. These tasks are designed to learn code structure as symbolic data, the importance of identifiers, and the bidirectional conversion of code and natural language. Both the encoder and decoder are trained on language data as well as on code data. A comprehensive set of open-source pre-trained models for NLP and code, including the model used in this work, is available through the popular HuggingFace Transformer library [24].

4 EXPERIMENTS

In the following, we report details on fine-tuning CodeT5 on the LTL synthesis task and the results of our experiments.

4.1 Representation

The previously introduced dataset consists of pairs of decomposed LTL specifications and circuits. More specifically, the circuits contain the realizability information and the circuit in AIGER format that satisfies the specification or represents a counter-strategy if the specification is unrealizable. For fine-tuning CodeT5, we represent the LTL specification as a single LTL formula in prefix notation. In contrast to previous work, we do not consider the decomposition when representing the specification and use a standard positional encoding instead of a tree positional encoding. The training target is formed from a string stating whether the specification is realizable concatenated with a string representing the AIGER circuit. We do not use any additional natural language prompting. To split the textual representations into individual tokens we rely on the RoBERTa tokenizer that was used to pre-train the CodeT5 model [23].

4.2 Training Details

We fine-tune the `small` version of the open-source CodeT5 model with about 60 million parameters. We train on an NVIDIA DGX A100 system for 20 000 training steps which takes about 7 hours. The batch size is set to 128 and the learning rate starts at 0.0005 and linearly decreases throughout the training. We use PyTorch [16], the HuggingFace transformers library [24], and the ML2 framework [20] to implement and evaluate the experiments.

4.3 Results

We evaluated the fine-tuned model on the same four datasets as in previous work [21]. `Testset` contains held-out instances obtained by the data generation method, `SYNTCOMP` consists of synthesis competition benchmarks from 2020, `Timeouts` is a set of generated specifications on which Strix timed out ($< 120s$), and `Smart Home` is an out-of-distribution (OOD) benchmark consisting of specifications for smart homes. For `SYNTCOMP` and `Smart Home`, we apply the same size restrictions as in previous work. The results are shown in Table 1. While previous work [21] highlighted the importance of the hierarchical Transformer architecture and a tree positional encoding for a better representation of the specification structure, this work shows that a standard pre-trained Transformer architecture, can match and exceed the results of task-specific architectures. Across all four datasets we observe improved results for beam size 16, which gives the best result on the respective datasets. The results support that the fine-tuned CodeT5 model generalizes better on unseen specifications. At the same time, the model is more sample efficient as it is only trained for 20 000 steps with a batch size of 128, whereas models in previous work have been trained for at least 30 000 steps with a batch size of 256.

5 CONCLUSION

We reported new experiments on fine-tuning pre-trained language models on synthesizing circuits from formal specifications in LTL. Compared to previous work that trains deep neural networks from scratch, our results show improved generalization on unseen specifications and improved sample efficiency. The experiments constitute two conclusions. (1) Pre-trained models could play an essential role in developing novel algorithms for LTL synthesis that integrate deep learning. Our evaluation shows improved performance on both practical synthesis benchmarks and benchmarks where algorithmic approaches fail. (2) Pre-training can compensate for not explicitly representing the structure of formal languages. When training deep neural networks from scratch, representing the decomposable tree structure of LTL specifications has been shown to be important. This encourages research in methods that incorporate both pre-training and representation of structure.

REFERENCES

[1] R. Abraham. 2021. Symbolic LTL Reactive Synthesis. http://essay.utwente.nl/87386/

[2] Armin Biere. 2007. The AIGER and-inverter graph (AIG) format version 20071012. (2007).

[3] Alonzo Church. 1964. Logic, arithmetic, and automata. (1964).

[4] Matthias Cosler, Frederik Schmitt, Christopher Hahn, and Bernd Finkbeiner. [n. d.]. Iterative Circuit Repair Against Formal Specifications. In *The Eleventh International Conference on Learning Representations*.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[6] Alexandre Duret-Lutz, Etienne Renault, Maximilien Colange, Florian Renkin, Alexandre Gbaguidi Aisse, Philipp Schlehuber-Caissier, Thomas Medioni, Antoine Martin, Jérôme Dubois, Clément Gillard, and Henrich Lauko. 2022. From Spot 2.0 to Spot 2.10: What's New?. In *Proceedings of the 34th International Conference on Computer Aided Verification (CAV'22) (Lecture Notes in Computer Science, Vol. 13372)*. Springer, 174–187.

[7] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. 2017. BoSy: An Experimentation Framework for Bounded Synthesis. In *Computer Aided Verification (Lecture Notes in Computer Science)*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 325–332.

[8] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. 2020. CodeBERT: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155* (2020).

[9] Swen Jacobs, Guillermo A. Perez, Remco Abraham, Veronique Bruyere, Michael Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaetan Staquet, Clement Tamines, Leander Tentrup, and Adam Walker. 2022. The Reactive Synthesis Competition (SYNTCOMP): 2018-2021. https://doi.org/10.48550/arXiv.2206.00251

[10] Ayrat Khalimov. 2021. Game-based bounded synthesis via BDDs. https://github.com/5nizza/sdf-hoa

[11] Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C Paulson. 2021. IsarStep: a Benchmark for High-level Mathematical Reasoning. In *International Conference on Learning Representations*.

[12] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[13] Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. 2020. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica* 57, 1-2 (2020), 3–36. https://doi.org/10.1007/s00236-019-00349-3

[14] Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. 2018. Strix: Explicit Reactive Synthesis Strikes Back!. In *Computer Aided Verification (Lecture Notes in Computer Science)*, Hana Chockler and Georg Weissenbacher (Eds.). Springer International Publishing, Cham, 578–586.

[15] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. CodeGen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474* (2022).

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[17] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ieee, 46–57.

[18] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research* 21, 1 (2020), 5485–5551.

[20] Frederik Schmitt, Matthias Cosler, and Christopher Hahn. 2023. ML2 - Machine Learning for Mathematics and Logics. https://github.com/reactive-systems/ml2

[21] Frederik Schmitt, Christopher Hahn, Markus N Rabe, and Bernd Finkbeiner. 2021. Neural circuit synthesis from specification patterns. *Advances in Neural Information Processing Systems* 34 (2021), 15408–15420.

[22] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (Eds.). 5998–6008. https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[23] Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. *arXiv preprint arXiv:2109.00859* (2021).

[24] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Online, 38–45. https://www.aclweb.org/anthology/2020.emnlp-demos.6