LEARNING REACTIVE SYNTHESIS FROM MODEL CHECKING FEEDBACK

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning applications to formal verification typically fall into one of two categories: employing reinforcement learning that suffers from slow convergence, or supervised learning that suffers from limited exploration. For reactive synthesis, the problem of automatically constructing a system that satisfies a formal specification, existing approaches fall into the latter category. In this paper, we propose a hybrid approach that only initializes the model with supervised learning and then continues training by reinforcing formally verified predictions. We show that by training the model to synthesize correct solutions rather than fixating on the supervised data, performance substantially improves. We can further utilize our approach to optimize for size without any performance degradation. Finally, we show that we can iteratively reinforce on open problems that synthesis tools are unable to solve. Our approach is demonstrated for both deep neural networks trained from scratch and pre-trained models fine-tuned on reactive synthesis, establishing new state-of-the-art results for learning reactive synthesis.

1 INTRODUCTION

Reactive synthesis is one of the fundamental problems in formal verification: Given a specification in a formal logic, a synthesis algorithm automatically constructs a system that satisfies the specification (Church, 1963). The promise of making the manual implementation of systems superfluous has sparked research for more than half a century, ranging from early theoretical contributions (Buchi & Landweber, 1969) to modern algorithms and tools (Meyer et al., 2018; Renkin et al., 2022; Kretínský et al., 2025). Nowadays, an annually held competition tracks the progress in the field (Jacobs et al., 2022). Most research interest revolves around reactive synthesis from specifications expressed in linear-time temporal logic (LTL) (Pnueli, 1977) for which the synthesis results are hardware circuits. The success story of LTL started in hardware model-checking, where a multitude of industry-level model-checkers have been developed Kuppe et al. (2019), which eventually led to a widely applied industry standard (IEEE-Commission, 2005). The computationally harder synthesis problem from LTL specifications is theoretically complex and often intractable in practice, as existing synthesis algorithms often time out even for small specifications. One promising way to overcome these barriers is deep learning: in recent years, reactive synthesis was turned into a deep learning problem (Schmitt et al., 2021) to build on the promising success of deep learning in program synthesis (Austin et al., 2021) and code generation (Rozière et al., 2023).

Current deep learning solutions, however, suffer from the drawbacks of supervised learning: They generalize over the training data and mimic the behavior of the synthesis tool used for data generation. To enable supervised learning, large datasets of specification-circuit pairs are constructed with the help of algorithmic synthesis tools for synthesizing the training targets (Schmitt et al., 2021). While supervised learning on such datasets allows for some generalization, the resulting systems are ultimately confined to the abilities of the synthesis tool that generated the training data. In the reinforcement learning literature, such problem settings are regarded as imitation learning or learning from demonstration (Schaal, 1996), i.e., training an agent to imitate a teacher that is providing labeled data. In the context of synthesis, the learning algorithm can be seen as imitating the specific synthesis algorithm and tool that was used to generate the dataset. Imitation learning is well-known to have limited ability to fully generalize or substantially improve over the labeled data (Ross & Bagnell, 2010). Therefore, it appears unlikely that a fully supervised learning approach alone will overcome the limitations of reactive synthesis tools.

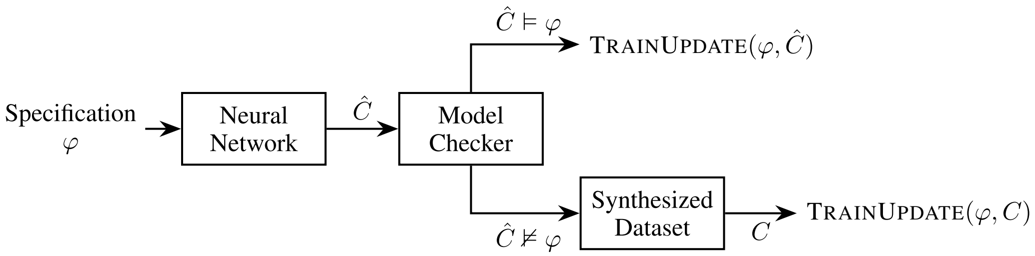

Specification
φ
Neural Network
$\hat{C}$
Model Checker
$\hat{C} \models \varphi$
TrainUpdate$(\varphi, \hat{C})$
$\hat{C} \not\models \varphi$
Synthesized Dataset
C
TrainUpdate(φ, C)

Figure 1: An overview of our method showing the conditional training update. For a given specification φ, the neural network predicts a circuit $\hat{C}$ that is verified against φ by a model-checker. In case $\hat{C}$ satisfies the specification ($\hat{C} \models \varphi$), we perform a gradient update with φ and $\hat{C}$. In the dual case $\hat{C} \not\models \varphi$, a synthesis tool is used to compute a correct circuit C for the gradient update.

In this paper, we present a new learning approach for reactive synthesis that overcomes the restrictions of imitation learning. Our approach uses supervised learning only as an initialization of the neural network, after which a second training stage is entered that allows the neural network to self-improve its own predictions. Depicted in Figure 1, we formally verify the circuits $\hat{C}$ that was predicted by the neural network and utilize this feedback to train on circuits that the neural network already predicts correctly. If the prediction was correct, we reinforce the model with a training update on the specification and its prediction. We only fall back to a training target C from a synthesis tool if the neural network is not yet able to predict a correct circuit ($\hat{C} \not\models \varphi$). The formal verification of the prediction is achieved by utilizing existing model-checking tools and benefits from the lower complexity of the model-checking problem for LTL. We thereby change the training objective from imitating a synthesis tool to the actual objective, i.e., the synthesis of a circuit that satisfies the specification. The effects of this change are substantial, as shown in our experiments. Our approach not only improves sample efficiency but also allows our model to generalize better. We can amplify the results by searching over multiple predictions of the model and make use of it for optimizing the size of circuits. Finally, we show that our method clearly scales beyond the capabilities of the synthesis tool used to generate the data by including open problems into the training process.

In summary, we make the following contributions:

1. We introduce a novel deep learning approach to reactive synthesis, combining both supervised and reinforcement learning to optimize correctness over imitating synthesis tools.
2. We generalize our approach with a search component to an expert iteration method and demonstrate its ability to further improve performance and optimize the size of circuits.
3. We utilize our framework to iterate on synthesis problems that reactive synthesis tools are unable to solve and show the potential of our methods to gradually improve on them.

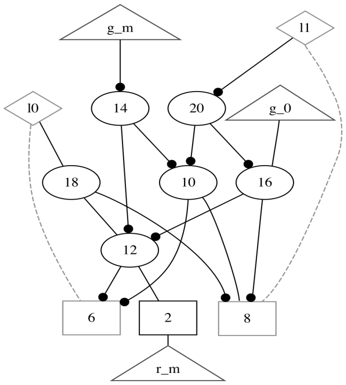
The remainder of the paper is structured as follows. Related work is discussed in Section 2. Background on reactive synthesis and deep learning methods thereof are described in Section 3. Our method for learning reactive synthesis from model checking feedback is presented in Section 4 followed by an experimental evaluation in Section 5. We conclude and discuss future work in Section 6.

2 Related Work

Expert Iteration and Theorem Proving. Expert iteration (Anthony et al., 2017) has been applied with great success to automated theorem proving. An early example in the context of large language models is the GPT-f work by Polu & Sutskever (2020) which was later developed into a full curriculum learning approach by Polu et al. (2023). Recent applications to Lean corpora such as the Lean-workbook corpus (Ying et al., 2024) were presented by Wu et al. (2024) and Xin et al. (2025).

Deep Learning Aided Verification and Synthesis. The importance of formal methods in domains such as hardware design has led to extensive application of deep learning to verification and

(assumptions)
$\Box\Diamond\neg r_m$
$\rightarrow$
(guarantees)
$\Box(\neg g_m \vee \neg g_0)$
$\Box(r_0 \rightarrow \Diamond g_0)$
$\Box(r_m \rightarrow (\neg g_0 \, \mathcal{U} \, g_m))$



```
aag 10 2 2 2 6
2
4
6 18
8 21
15
16
10 7 8
12 2 7
4 11 13
16 9 13
18 9 12
20 11 17
```

Figure 2: We show an example for an LTL specification that specifies a prioritized arbiter on the left. A circuit implementing the specification is provided as a graphical representation (middle) and in the AIGER format (right).

synthesis tasks, covering all steps of the verification process. Beginning with formal specifications, deep learning was used to automatically formalize natural language into specifications (Chen et al., 2023; Cosler et al., 2023a; Mendoza et al., 2024). For the verification of formal specifications neural networks were used as proof certificates (Giacobbe et al., 2024). Deep learning has been applied to hardware systems themselves at various levels of abstraction, including boolean circuits (Neto et al., 2021; Chowdhury et al., 2024; Wang et al., 2024), hardware description languages (Vasudevan et al., 2021; Thakur et al., 2024; Zhu et al., 2022; Zheng et al., 2025) and chip placement (Mirhoseini et al., 2021). For reactive synthesis specifically, a neural-symbolic portfolio solver was developed (Cosler et al., 2024) following the work of Schmitt et al. (2021). The same architecture that was applied to reactive synthesis was also applied to the related problem of circuit repair (Cosler et al., 2023b).

Deep Learning for Formal Logics. Deep learning has proven itself to be a promising direction for solving formal logic problems. For Boolean Satisfiability (SAT) supervised (Selsam & Bjørner, 2019; Selsam et al., 2019; Li et al., 2024), unsupervised (Amizadeh et al., 2019; Ozolins et al., 2022) and expert iteration approaches (Ghanem et al., 2024) were explored for both predicting and proving satisfiability. For quantified Boolean formulas (QBF) (Lederman et al., 2020) and Satisfiability Modulo Theories (SMT) (Balunovic et al., 2018) deep neural networks were successfully integrated into algorithmic solvers. For temporal logics such as LTL, most research focused on learning traces for satisfiability prediction (Hahn et al., 2021; Luo et al., 2022; Isik et al., 2024). Closely related to temporal logics, representation of Büchi automata were learned with Graph Neural Networks (Stammet et al., 2022).

3 Background

Reactive synthesis is a fundamental problem in computer science (Church, 1963), with theoretical solutions already established in the 1960s (Buchi & Landweber, 1969). The most common version is reactive synthesis, where a circuit is synthesized for a provided temporal, e.g., linear-time temporal logic (LTL) (Pnueli, 1977). LTL combines propositional boolean logic operators such as $\neg, \wedge, \vee, \implies$ with temporal operators such as $\bigcirc$ - *next*, $\mathcal{U}$ - *until*, $\Box$ - *always*, and $\Diamond$ - *eventually* which allows to specify the behavior of a reactive system that maintains a continuous interaction with its environment. We give an example of an LTL specification for a prioritized arbiter in Figure 2. It specifies that, under the assumption that $\neg r_m$ is infinitely often true, g_m and g_0 are never true at the same time, whenever r_0 holds, then at some later point g_0 holds, and that whenever r_m holds, g_0 does not hold until g_m holds. A system that is implementing such a specification is typically represented as a sequential circuit that translates an infinite stream of inputs into an infinite stream of outputs. A standard representation for sequential circuits is And-Inverter Graphs extended with simple memory elements, so-called latches. The AIGER format (Brummayer et al., 2007), which we employ in this work, is a widely adopted textual encoding for such And-Inverter Graphs. In Figure 2, we show a graphical representation of an And-Inverter Graph and its corresponding

AIGER format. The graphical representation is read from bottom to top, with triangles being the input and outputs. Diamonds are latches that store their input value for one clock cycle, and ellipses describe AND-gates. Negations are depicted by a dot on the wire between gates. Algorithms for solving LTL synthesis can be broadly categorized into game-based (Rabin, 1972) and bounded synthesis (Schewe & Finkbeiner, 2007). All algorithms face the challenges related to the problem being 2EXPTIME-complete (Pnueli & Rosner, 1989).

The complexity bound of the algorithmic approach to the problem of reactive synthesis motivated the use of machine learning methods. Supervised learning has recently been applied to reactive synthesis (Schmitt et al., 2021). To enable their approach, Schmitt et al. (2021) proposed a data generation method that leverages the synthesis tool Strix (Meyer et al., 2018) to generate large numbers of synthetic specification-circuit pairs. Since reactive synthesis specifications typically consist of conjunctions of smaller assumption and guarantee properties (see Figure 2), the authors collected a set of such properties from the annual synthesis competition (Jacobs et al., 2022) and randomly combined them to form new specifications. The generation process begins with a single property and incrementally adds new properties until either the specification becomes unrealizable or the synthesis tool times out after 120 seconds. Using this approach, the authors constructed a dataset containing hundreds of thousands of training samples. The reactive synthesis problem is then phrased as a sequence-to-sequence learning problem. The authors demonstrated that a hierarchical transformer architecture can be successfully trained on the synthetic dataset and is able to generalize to both the synthesis competition benchmarks and specifications that the synthesis tool cannot solve. The same dataset was later used to fine-tune code generation models on the reactive synthesis task, which exhibited superior generalization compared to the hierarchical transformer (Schmitt et al., 2023). We will adopt the same architectures and datasets to evaluate our approach, as detailed in Section 5.

4 METHOD

In the following, we describe our approach for combining supervised learning with model-checking feedback for reactive synthesis. We begin by describing the general idea and present three adapted versions in the subsequent paragraphs. Our method requires a dataset $D = \{(\varphi_1, C_1), \ldots, (\varphi_n, C_n)\}$ of specification-circuit pairs and access to a model-checking tool to automatically verify a circuit C against a specification φ. Given dataset D, we start by building an initial model M_0 with standard supervised learning. Assuming that the labels in dataset D were generated with a synthesis tool, we can view this first stage as an imitation learning phase that trains the model to imitate the synthesis tool. Following the initialization with imitation learning, we begin to iteratively improve our model with its own predictions. We distinguish between using top-1 and top-k predictions, as well as access to the circuit from the dataset that we can fall back to. For simplicity, we describe the methods for a single specification-circuit pair. It is straightforward to generalize this to the mini-batched algorithm that we have implemented in our experiments.

Reinforcing Learned Semantics. Our first method is motivated by the observation that some model predictions differ from the training targets but still satisfy the input specification. This is an expected situation since, in theory, there exist infinitely many correct circuits for each specification. A practical reason can, for example, be as simple as changing the order of gates or as complicated as creating a completely different circuit. We propose reinforcing such predictions as described in Algorithm 1. In each iteration, we sample a specification-circuit pair (φ, C) from training data D and evaluate model M_{i-1} from the previous iteration on specification φ. If prediction $\hat{C}$ of our model satisfies the specification (denoted as $\hat{C} \models \varphi$), we train on $\hat{C}$ instead; otherwise, we keep the training target C.

Expert Iteration and Circuit Minimization. The previous method can be generalized by performing a search over the model outputs, evaluating the top-k circuit predictions, and selecting the training target among them. Specifically, we generalize lines 4 and 5 in Algorithm 1 by first obtaining the top-k circuits $\hat{C}_1, \ldots, \hat{C}_k$, verifying them, and then selecting our training target from the circuits satisfying the specification and the circuit C from the training data. The generalized version is given as Algorithm 3 in Appendix C.

Algorithm 1 Learning from Model Checking Feedback

```
Require: $D = \{(\varphi_1, C_1), \dots, (\varphi_n, C_n)\}$
$M_0 \leftarrow \textsc{ImitationLearning}(D)$
for $i \leftarrow 1$ to $n$ do
    Sample $(\varphi, C) \sim D$
    $\hat{C} \leftarrow \textsc{Evaluate}(M_{i-1}, \varphi)$
    $C^* \leftarrow$ if $\hat{C} \models \varphi$ then $\hat{C}$ else $C$
    $M_i \leftarrow \textsc{TrainUpdate}(M_{i-1}, (\varphi, C^*))$
```

This is motivated by the observation that generating more than a single output (for example, through beam search) increases the likelihood of finding correct solutions. We note that this resembles the expert iteration method proposed in the context of reinforcement learning (Anthony et al., 2017). In our context, building an expert corresponds to automatically verifying the top-k model predictions with the fallback mechanism to the training dataset. For larger k, it becomes likely to find multiple valid circuits, therefore raising the question of how to best choose between them. In addition to selecting the first circuit that satisfies the specification, our method is also used to optimize the size of circuits. Smaller circuits are generally preferable as they are easier to understand and, in principle, less expensive to manufacture. By selecting the smallest circuit, we are not only reinforcing the semantics but also training the model to find minimal circuits, further disentangling the model from the initial dataset.

Iterating on Open Problems. Finally, we consider a version of our method in which we do not always have access to a training target for a given specification. We refer to such problems as *open problems*. This is particularly interesting for improving over a synthesis tool where the open problems correspond to those that the synthesis tool is unable to solve. Note that synthesis tools run into timeouts even for specifications for which small solutions exist if they are hard to compute. We modify our method as follows: With probability p_{open}, we sample a specification from the dataset of open problems instead of a specification-circuit pair from the dataset D. We then proceed in the same way as in expert iteration. However, if none of the model predictions satisfy the specification, we do not have a circuit for the next training update. In that case, we continue without a training update and begin sampling either an open problem with probability p_{open} or a labeled specification-circuit pair with probability $1 - p_{open}$ again.

We give the full description for iterating on open problems in Algorithm 2. In contrast to Algorithm 1 on reinforcing learned semantics, we additionally require a dataset of open problems and a probability to select from that dataset.

Algorithm 2 Iterating on Open Problems

```
Require: $D_{train}, D_{open}, p_{open}$
$M_0 \leftarrow \textsc{ImitationLearning}(D_{train})$
for $i \leftarrow 1$ to $n$ do
    if $random() > p_{open}$
        Sample $(\varphi, C) \sim D_{train}$
        $\{\hat{C}_1, \dots, \hat{C}_k\} \leftarrow \textsc{Search}(M_{i-1}, \varphi)$
        $C^* \leftarrow \textsc{VerifyAndSelect}(\varphi, \{\hat{C}_1, \dots, \hat{C}_k\} \cup \{C\})$
        $M_i \leftarrow \textsc{TrainUpdate}(M_{i-1}, (\varphi, C^*))$
    else
        Sample $\varphi \sim D_{open}$
        $\{\hat{C}_1, \dots, \hat{C}_k\} \leftarrow \textsc{Search}(M_{i-1}, \varphi)$
        if $C^* \in \{\hat{C}_1, \dots, \hat{C}_k\}$ such that $C^* \models \varphi$
            $M_i \leftarrow \textsc{TrainUpdate}(M_{i-1}, (\varphi, C^*))$
```

5 EXPERIMENTS

In this section, we report on the experimental results for our new learning approach for reactive synthesis. We begin by first describing the experimental setup, including datasets, architectures, and implementation in Section 5.1. The different variants of our method are then evaluated in Sections 5.2, 5.3, and 5.4 respectively.

5.1 EXPERIMENTAL SETUP

We closely follow the experimental setup of previous work to evaluate our method. In particular, we use the same datasets for training and testing as Schmitt et al. (2021). The training dataset comprises 200 000 specification-circuit pairs resulting from a data generation method based on specification patterns and was created with the synthesis tool Strix (Meyer et al., 2018). The trained models are evaluated on three datasets: The holdout dataset created with the same data generation method as the training data is referred to as `Testset`. The SYNTCOMP dataset is a collection of challenging benchmarks, containing 145 instances directly taken from the annual reactive synthesis competition. The `Timeouts` dataset was collected during data generation and contains specifications that the synthesis tool Strix could not solve within 120 seconds. For all datasets, we report $pass@k$ rates denoting whether one of the k circuits generated by the model satisfies the specification. Verification is performed with the nuXmv model checker (Cavada et al., 2014).

We evaluate the two architectures that have been previously employed for learning the reactive synthesis problems, hierarchical transformers (Li et al., 2021) and CodeT5 (Wang et al., 2021). A hierarchical transformer architecture was chosen in previous work for its permutation invariance with respect to different orderings of assumption and guarantee properties in the temporal logic specification. We chose the same hyperparameters for the architecture as in previous work, training models with 8 layers, 4 attention heads, an embedding dimension of 256, and a feed-forward neural network dimension of 1024. Token positions of LTL properties are encoded with a tree positional encoding (Shiv & Quirk, 2019). All models are trained with the Adam optimizer (Kingma & Ba, 2015) with betas set to $\beta_1 = 0.9$ and $\beta_2 = 0.98$. We implemented the standard transformer learning rate schedule proposed by Vaswani et al. (2017). The batch size is set to 256. From the CodeT5 model family, we chose the small version, which has about 60 million trainable parameters. We train CodeT5 models with the AdamW optimizer (Loshchilov & Hutter, 2019) with a start learning rate of 0.0005 and a linear learning rate decay. The weight decay is set to 0.1 and the batch size is set to 128.

We implemented all experiments in PyTorch (Paszke et al., 2019) and included the HuggingFace (Wolf et al., 2020) transformers library for the CodeT5 experiments. We train on a NVIDIA DGX A100 system. All training runs are between 8 and 96 hours. Further training details and parameters are provided in the respective sections.

5.2 REINFORCING LEARNED SEMANTICS

We begin by evaluating the semantic reinforcement for both hierarchical transformer models trained from scratch and fine-tuned CodeT5 models. In both cases, we obtain the top prediction of the model using greedy decoding. All results are summarized in Table 1 and further discussed below.

Hierarchical Transformer (HT). In Table 1, we directly compare the supervised learning of the hierarchical transformer with our semantic reinforcement approach in the rows specifying model HT. Specifically, we compare training for 30 000 steps of supervised learning with only 10 000 steps of supervised learning followed by 2 000 steps of semantic reinforcement. We see the pass rates increase for all datasets, with the difference being most pronounced for the $pass@1$ rates.

The training takes about 1 hour and 45 minutes for 10 000 steps of supervised learning and 8 hours and 20 minutes for 2 000 steps of semantic reinforcement when running 4 model checker instances in parallel. While a semantic reinforcement step is more expensive, a relatively small number of steps toward the end of training are sufficient to achieve the performance gains. In Figure 3, we visualize the diminishing returns of changing from supervised learning to semantic reinforcement early in the training process. Further results for different combinations of supervised learning steps and semantic reinforcement steps are shown in Table 5 in Appendix A.

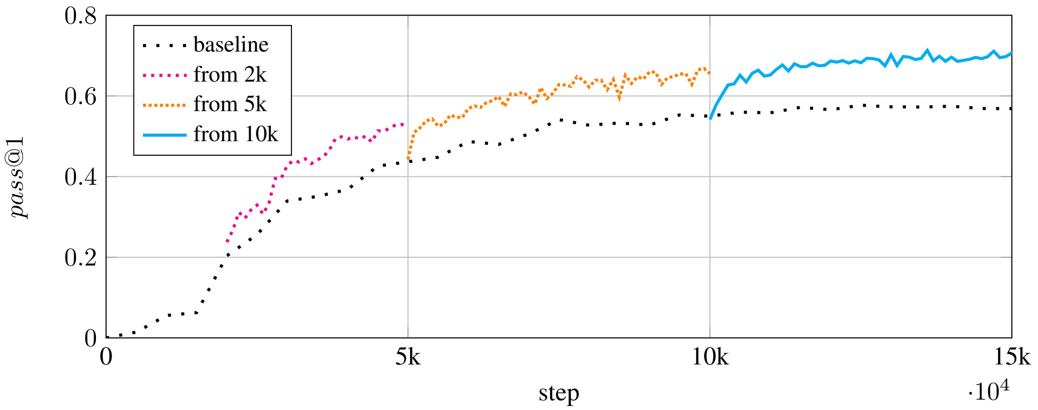


Figure 3: The $pass@1$ rate on validation data over the course of the training comparing supervised learning (baseline), and starting reinforcing learned semantics from step 2 000, 5 000, and 10 000.

Table 1: Pass rates on evaluation datasets for both hierarchical transformer (HT) and CodeT5 showing results for 2 000 steps of reinforcing learned semantics.

Dataset	Model	Method	pass@1	pass@4	pass@8	pass@16
Testset	HT	Supervised Learning	53.6	70.4	75.8	79.9
		+ Semantic Reinforcement	**70.4**	**80.0**	**82.6**	**85.3**
	CodeT5	Fine-tuning	61.0	76.3	81.2	84.5
		+ Semantic Reinforcement	**70.1**	**81.7**	**85.2**	**87.4**
SYNTCOMP	HT	Supervised Learning	51.9	60.0	63.6	66.8
		+ Semantic Reinforcement	**53.8**	**62.1**	**65.5**	**66.9**
	CodeT5	Fine-tuning	**55.2**	63.2	65.7	68.3
		+ Semantic Reinforcement	53.1	**63.7**	**67.4**	**71.9**
Timeouts	HT	Supervised Learning	11.7	21.1	25.9	30.1
		+ Semantic Reinforcement	**24.0**	**31.7**	**34.2**	**36.5**
	CodeT5	Fine-tuning	13.8	24.1	30.2	35.2
		+ Semantic Reinforcement	**27.7**	**36.1**	**39.0**	**42.4**

We performed an ablation study to investigate how much performance improvement is due to reinforcing correct circuits. In the ablation study, we compare our setting with skipping over correct circuits and only training on specification-circuit pairs that are not solved yet. Note that this resembles hard negative mining techniques employed, for example, in computer vision domains (Shrivastava et al., 2016). The ablation study can be found in Appendix A.2 and concludes that simply not training on already correct circuits behaves similarly to the regular supervised learning setting. We can therefore attribute performance improvements to the reinforcement of correct circuits.

Pre-trained Code Generation Models (CodeT5). In previous work, it was reported that instead of training hierarchical transformers from scratch, fine-tuning pre-trained code generation models such as CodeT5 (Wang et al., 2021) yields comparable or better results for reactive synthesis. We repeat the experiments for hierarchical transformers with CodeT5. The rows specifying model

CodeT5 in Table 1 compare CodeT5 fine-tuned for 30 000 steps with CodeT5 fine-tuned for 20 000 steps and performing an additional 2 000 steps of reinforcing learned semantics. The training takes about 7 hours and 20 minutes for 20 000 steps of fine-tuning and 4 hours and 10 minutes for 2 000 steps of semantic reinforcement. Similar to training transformers from scratch, reinforcing learned semantics improves fine-tuned code models for almost all pass rates on all evaluation datasets. We note that CodeT5's general advantage over hierarchical transformers carries over to the setup with reinforcing learned semantics. More results for different combinations of training steps can be found in Table 12 in Appendix B.

5.3 Expert Iteration

In this section, we evaluate the generalization of our method to expert iteration. We obtain the top-k circuit predictions from our model with a beam search (Sutskever et al., 2014) and distinguish two criteria to choose our training target among them. First, we present results for selecting the first circuit that satisfies the specification, and second, the results for choosing the smallest circuit among the ones satisfying the specification. In both cases, we fall back to the circuit from our dataset if no circuit satisfies the specification. We only report results for CodeT5 since CodeT5 outperformed hierarchical Transformers in most evaluations. The results for hierarchical Transformers can be found in Table 7 in Appendix A.3.

Table 2: Pass rates for CodeT5 on evaluation datasets after 20 000 steps of supervised learning and 2 000 steps of expert iteration compared for different beam sizes. Results are averaged over 3 runs.

Dataset	Beam Size	pass@1	pass@4	pass@8	pass@16
`Testset`	1	70.1	81.7	85.2	87.4
	4	**74.8**	**84.4**	**87.2**	**89.3**
`SYNTCOMP`	1	53.1	63.7	67.4	71.9
	4	**56.1**	**67.8**	**69.9**	**72.4**
`Timeouts`	1	27.7	36.1	39.0	42.4
	4	**32.1**	**41.3**	**43.9**	**46.5**

We summarize the results for the first criterion in Table 2, comparing a beam size of 1 and a beam size of 4. Note that a beam size of 1 corresponds to greedy decoding and therefore to the results of the method presented in Section 5.2. We can see through all $pass@k$ rates a clear improvement when moving from Beam Size 1 to 4. For the `Testset`, the result for $pass@16$ constitutes the best performance in the paper and establishes new state-of-the-art results on the dataset. We note that the improvements come at a higher computational cost, as more model checking calls are made each training step. The number of model checking calls scales linearly in the beam size.

In a second step, we adapt our expert iteration method to optimize for smaller circuits. We optimize our method by selecting the smallest correct circuit from the set of the top-k predictions as the next training target. Notably, the syntactic accuracy during expert iteration with selecting smaller circuits drops from 36% to 0%, while the semantic accuracy is still improving (see Figure 4 in the Appendix). This shows that the model's predictions shift strongly away from the original training data, while still being correct.

We show that optimizing for size does not impede performance (see A.4) while generating smaller circuits. We evaluate on the `SYNTCOMP` dataset, and compare circuit sizes to Strix, the algorithmic state-of-the-art tool in Reactive Synthesis. As shown in Table 3, our circuits are 54% smaller on average, while fine-tuning without Expert Iteration creates circuits that are 46% smaller than Strix's circuits. Expert Iteration improves the circuit size over fine-tuning by 12.5%. Absolute results, including for the hierarchical Transformer, can be found in Appendix A.4.

Table 3: Improvement in circuit size over different baselines (percent). Evaluated on SYNTCOMP.

Improvement of	**eval@1**	**eval@4**	**eval@8**	**eval@16**
Fine-tuning over Strix	46.0	46.1	46.4	48.7
Expert Iteration over Strix	54.6	55.2	56.5	55.0
Expert Iteration over Fine-tuning	12.5	9.1	5.7	4.7

5.4 Iterating on Open Synthesis Problems

By applying deep learning to reactive synthesis, we aim to enable compute solutions for synthesis problems that existing synthesis tools cannot solve. The `Timeouts` dataset that we evaluated on in previous sections provides exactly these kind of specifications. It contains specifications that the Strix could not solve within 120 seconds. Note that increasing this timeout does not increase the number of solved instances substantially because of the high complexity of the problem. In the following, we use the dataset to evaluate our method to iterate on open problems. We follow the same setup as in Section 5.3, fine-tuning CodeT5 models for 20 000 steps and then performing 2 000 steps of expert iteration with a beam size of 4. Within the 2 000 steps of expert iteration, we pick a problem of the `Timeouts` dataset with probability $p_{timeout}$ instead of a problem from the regular dataset. In Table 4 we report the results of the experiment comparing $p_{timeout}$ values of 0.0, 0.25, and 0.5. The results show that we can effectively bootstrap on open synthesis problems and solve more than half of the problems in the dataset that the symbolic synthesis tool was unable to solve. The results for $pass@16$ establish a new state-of-the-art result on the dataset. On other datasets, iterating on timeouts does not clearly improve or decrease performance as shown in the full results Table 14 in Appendix B.3.

Table 4: Pass rates for CodeT5 on holdout portion of `Timeouts` dataset after 20 000 steps of fine-tuning and 2 000 steps of expert iteration with beam size 4 comparing different probabilities $p_{timeout}$ of including samples from `Timeouts` during training. Results are averaged over 3 runs.

Dataset	$p_{timeout}$	**pass@1**	**pass@4**	**pass@8**	**pass@16**
	0.0	32.1	41.3	43.9	46.5
`Timeouts`	0.25	38.0	45.6	48.4	50.7
	0.5	**40.1**	**47.4**	**49.9**	**51.9**

6 Conclusion

We presented a combination of supervised learning and reinforcement learning for learning reactive synthesis. By rectifying the training objective to synthesizing correct circuits rather than imitating a synthesis tool, we showed substantial performance gains over pure supervised learning approaches on both in-distribution and out-of-distribution benchmarks. The tight integration with a model checker makes our approach flexible and allows to include search and other optimization criteria, such as size, in the learning process. Indeed, we showed the ability to discover smaller circuits than algorithmic synthesis tools.

The results of the paper show the potential of deep learning solutions to push the barriers of reactive synthesis. By bootstrapping on open synthesis problems, our approach is capable of solving more than half of the problems that modern synthesis tools cannot solve. The ability to self-improve and progress on a distribution of open problems is particularly interesting synthesis-style problems, and will be an essential step for solving reactive synthesis problems in practice.

7 Reproducibility Statement

All datasets and libraries used in this work are open source. In Section 5.1 we describe the hyperparameters for all architectures and training algorithms needed to reproduce the results. We will make our implementation publicly available after the double-blind review period ends. We averaged experimental results over 3 runs to ensure reproducibility.

8 LLM Use Statement

In this paper, large language models were used for small refinements when writing the paper and as an assistant tool when implementing the experiments.

References

Saeed Amizadeh, Sergiy Matusevych, and Markus Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=BJxgz2R9t7.

Thomas Anthony, Zheng Tian, and David Barber. Thinking fast and slow with deep learning and tree search. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5360–5370, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d8e1344e27a5b08cdfd5d027d9b8d6de-Abstract.html.

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *CoRR*, abs/2108.07732, 2021. URL https://arxiv.org/abs/2108.07732.

Mislav Balunovic, Pavol Bielik, and Martin T. Vechev. Learning to solve SMT formulas. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pp. 10338–10349, 2018. URL https://proceedings.neurips.cc/paper/2018/hash/68331ff0427b551b68e911eebe35233b-Abstract.html.

R. Brummayer, A. Cimatti, K. Claessen, N. Een, M. Herbstritt, H. Kim, T. Jussila, K. McMillan, A. Mishchenko, F. Somenzi, et al. The aiger and-inverter graph (aig) format version 20070427, 2007. Available at http://fmv.jku.at/aiger/.

J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. URL https://api.semanticscholar.org/CorpusID:4568478.

Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In Armin Biere and Roderick Bloem (eds.), *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pp. 334–342. Springer, 2014. doi: 10.1007/978-3-319-08867-9_22. URL https://doi.org/10.1007/978-3-319-08867-9_22.

Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. NL2TL: transforming natural languages to temporal logics using large language models. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pp. 15880–15903. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.EMNLP-MAIN.985. URL https://doi.org/10.18653/v1/2023.emnlp-main.985.

Animesh Basak Chowdhury, Marco Romanelli, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Retrieval-guided reinforcement learning for boolean circuit minimization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=0t1O8ziRZp.

Alonzo Church. Application of recursive arithmetic to the problem of circuit synthesis. *Journal of Symbolic Logic*, 28(4), 1963.

Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. nl2spec: Interactively translating unstructured natural language to temporal logics with large language models. In Constantin Enea and Akash Lal (eds.), *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II*, volume 13965 of *Lecture Notes in Computer Science*, pp. 383–396. Springer, 2023a. doi: 10.1007/978-3-031-37703-7_18. URL https://doi.org/10.1007/978-3-031-37703-7_18.

Matthias Cosler, Frederik Schmitt, Christopher Hahn, and Bernd Finkbeiner. Iterative circuit repair against formal specifications. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/forum?id=SEcSahl0Ql.

Matthias Cosler, Christopher Hahn, Ayham Omar, and Frederik Schmitt. Neurosynt: A neuro-symbolic portfolio solver for reactive synthesis. In Bernd Finkbeiner and Laura Kovács (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part III*, volume 14572 of *Lecture Notes in Computer Science*, pp. 45–67. Springer, 2024. doi: 10.1007/978-3-031-57256-2_3. URL https://doi.org/10.1007/978-3-031-57256-2_3.

Mohamed Ghanem, Frederik Schmitt, Julian Siber, and Bernd Finkbeiner. Learning better representations from less data for propositional satisfiability. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/56a225639da77e8f7c0409f6d5ba996b-Abstract-Conference.html.

Mirco Giacobbe, Daniel Kroening, Abhinandan Pal, and Michael Tautschnig. Neural model checking. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/9d0947107ea92d6ce369dce7749180dd-Abstract-Conference.html.

Christopher Hahn, Frederik Schmitt, Jens U. Kreber, Markus Norman Rabe, and Bernd Finkbeiner. Teaching temporal logics to neural networks. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=dOcQK-f4byz.

IEEE-Commission. IEEE standard for property specification language (PSL). *IEEE Std 1850-2005*, 2005.

Ilker Isik, Ramazan Gokberk Cinbis, and Ebru Aydin Gol. Interchangeable token embeddings for extendable vocabulary and alpha-equivalence. *CoRR*, abs/2410.17161, 2024. doi: 10.48550/ARXIV.2410.17161. URL https://doi.org/10.48550/arXiv.2410.17161.

Swen Jacobs, Guillermo A. Pérez, Remco Abraham, Véronique Bruyère, Michaël Cadilhac, Maximilien Colange, Charly Delfosse, Tom van Dijk, Alexandre Duret-Lutz, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Klara J. Meyer, Thibaud Michaud, Adrien Pommellet, Florian Renkin, Philipp Schlehuber-Caissier, Mouhammad Sakr, Salomon Sickert, Gaëtan Staquet, Clément Tamines, Leander Tentrup, and Adam Walker. The reactive synthesis competition (SYNTCOMP): 2018-2021. *CoRR*, abs/2206.00251, 2022. doi: 10.48550/ARXIV.2206.00251. URL https://doi.org/10.48550/arXiv.2206.00251.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Jan Kretínský, Tobias Meggendorfer, Maximilian Prokop, and Ashkan Zarkhah. Semml: Enhancing automata-theoretic LTL synthesis with machine learning. In Arie Gurfinkel and Marijn Heule (eds.), *Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part I*, volume 15696 of *Lecture Notes in Computer Science*, pp. 233–253. Springer, 2025. doi: 10.1007/978-3-031-90643-5_12. URL https://doi.org/10.1007/978-3-031-90643-5_12.

Markus Alexander Kuppe, Leslie Lamport, and Daniel Ricketts. The TLA+ toolbox. In Rosemary Monahan, Virgile Prevosto, and José Proença (eds.), *Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM 2019, Porto, Portugal, 7th October 2019*, volume 310 of *EPTCS*, pp. 50–62, 2019. doi: 10.4204/EPTCS.310.6. URL https://doi.org/10.4204/EPTCS.310.6.

Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJluxREKDB.

Wenda Li, Lei Yu, Yuhuai Wu, and Lawrence C. Paulson. Isarstep: a benchmark for high-level mathematical reasoning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=Pzj6fzU6wkj.

Zhaoyu Li, Jinpei Guo, and Xujie Si. G4satbench: Benchmarking and advancing SAT solving with graph neural networks. *Trans. Mach. Learn. Res.*, 2024, 2024. URL https://openreview.net/forum?id=7VB5db72lr.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Weilin Luo, Hai Wan, Jianfeng Du, Xiaoda Li, Yuze Fu, Rongzhen Ye, and Delong Zhang. Teaching ltlf satisfiability checking to neural networks. In Luc De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pp. 3292–3298. ijcai.org, 2022. doi: 10.24963/IJCAI.2022/457. URL https://doi.org/10.24963/ijcai.2022/457.

Daniel Mendoza, Christopher Hahn, and Caroline Trippel. Translating natural language to temporal logics with large language models and model checkers. In Nina Narodytska and Philipp Rümmer (eds.), *Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024*, pp. 1–11. IEEE, 2024. doi: 10.34727/2024/ISBN.978-3-85448-065-5_17. URL https://doi.org/10.34727/2024/isbn.978-3-85448-065-5_17.

Philipp J. Meyer, Salomon Sickert, and Michael Luttenberger. Strix: Explicit reactive synthesis strikes back! In Hana Chockler and Georg Weissenbacher (eds.), *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pp. 578–586. Springer, 2018. doi: 10.1007/978-3-319-96145-3_31. URL https://doi.org/10.1007/978-3-319-96145-3_31.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim M. Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nat.*, 594(7862): 207–212, 2021. doi: 10.1038/S41586-021-03544-W. URL https://doi.org/10.1038/s41586-021-03544-w.

Walter Lau Neto, Matheus Trevisan Moreira, Luca G. Amarù, Cunxi Yu, and Pierre-Emmanuel Gaillardon. Read your circuit: Leveraging word embedding to guide logic optimization. In *ASPDAC '21: 26th Asia and South Pacific Design Automation Conference, Tokyo, Japan, January 18-21, 2021*, pp. 530–535. ACM, 2021. doi: 10.1145/3394885.3431560. URL https://doi.org/10.1145/3394885.3431560.

Emils Ozolins, Karlis Freivalds, Andis Draguns, Eliza Gaile, Ronalds Zakovskis, and Sergejs Kozlovics. Goal-aware neural SAT solver. In *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*, pp. 1–8. IEEE, 2022. doi: 10.1109/IJCNN55064.2022.9892733. URL https://doi.org/10.1109/IJCNN55064.2022.9892733.

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

Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pp. 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32. URL https://doi.org/10.1109/SFCS.1977.32.

Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pp. 179–190. ACM Press, 1989. doi: 10.1145/75277.75293. URL https://doi.org/10.1145/75277.75293.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393, 2020. URL https://arxiv.org/abs/2009.03393.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=-P7G-8dmSh4.

Michael Oser Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, USA, 1972. ISBN 0821816632.

Florian Renkin, Philipp Schlehuber-Caissier, Alexandre Duret-Lutz, and Adrien Pommellet. Dissecting ltlsynt. *Formal Methods Syst. Des.*, 61(2):248–289, 2022. doi: 10.1007/S10703-022-00407-6. URL https://doi.org/10.1007/s10703-022-00407-6.

Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In Yee Whye Teh and D. Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, volume 9 of *JMLR Proceedings*, pp. 661–668. JMLR.org, 2010. URL http://proceedings.mlr.press/v9/ross10a.html.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950, 2023. doi: 10.48550/ARXIV.2308.12950. URL https://doi.org/10.48550/arXiv.2308.12950.

Stefan Schaal. Learning from demonstration. In Michael Mozer, Michael I. Jordan, and Thomas Petsche (eds.), *Advances in Neural Information Processing Systems 9, NIPS, Denver, CO, USA, December 2-5, 1996*, pp. 1040–1046. MIT Press, 1996. URL http://papers.nips.cc/paper/1224-learning-from-demonstration.

Sven Schewe and Bernd Finkbeiner. Bounded synthesis. In Kedar S. Namjoshi, Tomohiro Yoneda, Teruo Higashino, and Yoshio Okamura (eds.), *Automated Technology for Verification and Analysis, 5th International Symposium, ATVA 2007, Tokyo, Japan, October 22-25, 2007, Proceedings*, volume 4762 of *Lecture Notes in Computer Science*, pp. 474–488. Springer, 2007. doi: 10.1007/978-3-540-75596-8_33. URL https://doi.org/10.1007/978-3-540-75596-8_33.

Frederik Schmitt, Christopher Hahn, Markus N. Rabe, and Bernd Finkbeiner. Neural circuit synthesis from specification patterns. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 15408–15420, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/8230bea7d54bcdf99cdfe85cb07313d5-Abstract.html.

Frederik Schmitt, Matthias Cosler, and Bernd Finkbeiner. Neural circuit synthesis with pre-trained language models. In *First International Workshop on Deep Learning-aided Verification*, 2023. URL https://openreview.net/forum?id=Q2171WADNTT.

Daniel Selsam and Nikolaj S. Bjørner. Guiding high-performance SAT solvers with unsat-core predictions. In Mikolás Janota and Inês Lynce (eds.), *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pp. 336–353. Springer, 2019. doi: 10.1007/978-3-030-24258-9_24. URL https://doi.org/10.1007/978-3-030-24258-9_24.

Daniel Selsam, Matthew Lamm, Benedikt Bünz, Percy Liang, Leonardo de Moura, and David L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=HJMC_iA5tm.

Vighnesh Leonardo Shiv and Chris Quirk. Novel positional encodings to enable tree-based transformers. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pp. 12058–12068, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/6e0917469214d8fbd8c517dcdc6b8dcf-Abstract.html.

Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 761–769. IEEE Computer Society, 2016. doi: 10.1109/CVPR.2016.89. URL https://doi.org/10.1109/CVPR.2016.89.

Christophe Stammet, Prisca Dotti, Ulrich Ultes-Nitsche, and Andreas Fischer. Analyzing büchi automata with graph neural networks. *CoRR*, abs/2206.09619, 2022. doi: 10.48550/ARXIV.2206.09619. URL https://doi.org/10.48550/arXiv.2206.09619.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pp. 3104–3112, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. Verigen: A large language model for verilog code generation. *ACM Trans. Design Autom. Electr. Syst.*, 29(3):46:1–46:31, 2024. doi: 10.1145/3643681. URL https://doi.org/10.1145/3643681.

Shobha Vasudevan, Wenjie Jiang, David Bieber, Rishabh Singh, Hamid Shojaei, Richard Ho, and Charles Sutton. Learning semantic representations to verify hardware designs. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pp. 23491–23504, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/c5aa65949d20f6b20e1a922c13d974e7-Abstract.html.

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

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pp. 8696–8708. Association for Computational Linguistics, 2021. doi: 10.18653/V1/2021.EMNLP-MAIN.685. URL https://doi.org/10.18653/v1/2021.emnlp-main.685.

Zhihai Wang, Jie Wang, Qingyue Yang, Yinqi Bai, Xing Li, Lei Chen, Jianye Hao, Mingxuan Yuan, Bin Li, Yongdong Zhang, and Feng Wu. Towards next-generation logic synthesis: A scalable neural circuit generation framework. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/b3ac808c09f98444090a8f6c2d4bd1dc-Abstract-Conference.html.

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. 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, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale LEAN problems. *CoRR*, abs/2410.15700, 2024. doi: 10.48550/ARXIV.2410.15700. URL https://doi.org/10.48550/arXiv.2410.15700.

Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Haowei Zhang, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=I4YAIwrsXa.

Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook: A large-scale lean problem set formalized from natural language math problems. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and

Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/bf236666a2cc5f3ae05d2e08485efc4c-Abstract-Datasets_and_Benchmarks_Track.html.

Ziyang Zheng, Shan Huang, Jianyuan Zhong, Zhengyuan Shi, Guohao Dai, Ningyi Xu, and Qiang Xu. Deepgate4: Efficient and effective representation learning for circuit design at scale. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=b10lRabU9W.

Keren Zhu, Hao Chen, Walker J. Turner, George F. Kokai, Po-Hsuan Wei, David Z. Pan, and Haoxing Ren. TAG: learning circuit spatial embedding from layouts. In Tulika Mitra, Evangeline F. Y. Young, and Jinjun Xiong (eds.), *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2022, San Diego, California, USA, 30 October 2022 - 3 November 2022*, pp. 66:1–66:9. ACM, 2022. doi: 10.1145/3508352.3549384. URL https://doi.org/10.1145/3508352.3549384.

A Full Hierarchical Transformer Results

A.1 Semantic Reinforcement

Table 5: Pass rates across datasets, highlighting the effect of varying the number of Expert Iteration (EI) steps and Supervised Learning (SL) steps.

Dataset	# SL Steps	# EI Steps	pass@1	pass@4	pass@8	pass@16
Testset	10k	2k	70.4	80.0	82.6	85.3
	10k	5k	68.7	79.3	82.2	85.6
	15k	2k	67.3	78.6	83.2	86.6
	15k	5k	69.1	79.9	82.9	85.5
SYNTCOMP	10k	2k	53.8	62.1	65.5	66.9
	10k	5k	53.8	64.8	67.6	70.3
	15k	2k	57.2	63.4	68.3	68.3
	15k	5k	57.9	68.3	71.0	71.7
Timeouts	10k	2k	24.0	31.7	34.2	36.5
	10k	5k	25.2	32.1	34.4	36.9
	15k	2k	22.3	29.0	31.5	34.1
	15k	5k	24.5	31.5	33.8	35.6

A.2 Hard Negative Mining

Table 6: Results for a hierarchical transformer trained for 10 000 steps on the full dataset and for 2 000 steps on the subset of specification–circuit pairs it could not solve up to that point.

Dataset	pass@1	pass@4	pass@8	pass@16
Testset	55.2	72.9	78.1	82.5
SYNTCOMP	51.7	63.4	67.6	70.3
Timeouts	12.7	22.6	26.7	32.3

A.3 Expert Iteration

Table 7: Pass rates across datasets, showing the effect of varying the beam search size alongside the number of Expert Iteration (EI) steps and Supervised Learning (SL) steps.

Dataset	# SL Steps	# EI Steps	Beam Size	pass@1	pass@4	pass@8	pass@16
Testset	10k	2k	1	70.4	80.0	82.6	85.3
	5k	2k	4	66.1	78.5	82.0	85.0
	10k	2k	4	75.0	84.4	87.2	89.2
SYNTCOMP	10k	2k	1	53.8	62.1	65.5	66.9
	5k	2k	4	41.1	56.6	63.4	65.5
	10k	2k	4	57.2	66.9	69.0	71.7
Timeouts	10k	2k	1	24.0	31.7	34.2	36.5
	5k	2k	4	24.5	35.4	38.8	41.4
	10k	2k	4	29.9	39.4	42.1	44.9

A.4 Circuit Minimization Results

Table 8: Evaluation of the hierarchical transformer models optimized for size vs. optimized for correctness

Dataset	Criterion	pass@1	pass@4	pass@8	pass@16
Testset	Correctness	76.4	84.0	86.7	88.8
	Size	76.5	83.8	86.9	89.1
SYNTCOMP	Correctness	59.8	66.9	69.9	72.6
	Size	57.2	62.8	68.0	71.0
Timeouts	Correctness	31.5	39.9	42.9	45.5
	Size	31.9	39.8	43.1	45.7

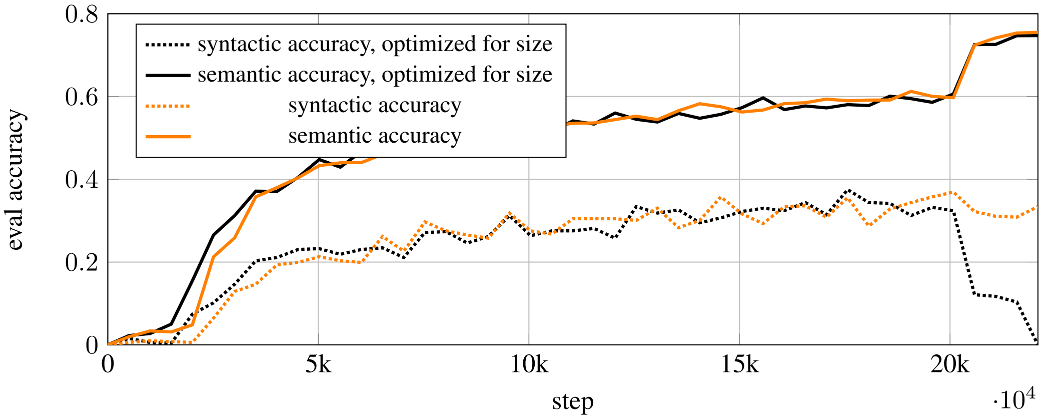


Figure 4: Comparison of semantic and syntactic accuracy during training for Expert Iteration, with and without size optimization.

Table 9: Evaluation of the CodeT5 models optimized for size vs. optimized for correctness

Dataset	Criterion	pass@1	pass@4	pass@8	pass@16
`Testset`	Correctness	74.8	84.4	87.2	89.3
	Size	75.0	82.7	85.9	88.4
`SYNTCOMP`	Correctness	56.1	67.8	69.1	72.4
	Size	55.4	59.3	64.4	68.3
`Timeouts`	Correctness	32.1	41.3	43.9	46.5
	Size	31.6	38.3	41.8	44.9

Table 10: Comparison of circuit sizes (gates and latches) on CodeT5 for the `SYNTCOMP` benchmark, restricted to samples solved by both methods

Method	eval@1	eval@4	eval@8	eval@16
CodeT5 Fine-tuning	5.13	4.53	4.4	4.29
+ Expert Iteration	4.49	4.12	4.15	4.09
HT Training	4.4	4.48	4.53	4.27
+ Expert Iteration	3.98	4.2	4.11	4.14

Table 11: Circuit sizes (gates and latches) on CodeT5 for the SYNTCOMP benchmark, restricted to samples solved by both our method and Strix.

Method	**eval@1**	**eval@4**	**eval@8**	**eval@16**
CodeT5 Fine-tuning	5.36	4.99	5.2	4.83
Strix	9.93	9.25	9.7	9.41
Expert Iteration on CodeT5	4.63	4.39	4.16	4.12
Strix	10.2	9.81	9.57	9.15
HT Training	4.71	4.76	4.86	4.53
Strix	9.45	9.51	9.32	9.12
Expert Iteration on HT	4.53	4.4	4.64	4.59
Strix	9.16	9.59	9.65	9.2

B Full CodeT5 Results

B.1 Semantic Reinforcement

Table 12: Accuracy of pre-trained CodeT5 across varying numbers of Supervised Learning (SL), Expert Iteration (EI) steps and beam sizes.

Dataset	SL Steps	EI Steps	pass@1	pass@4	pass@8	pass@16
Testset	30k	0	61.2	75.5	80.7	83.9
	50k	0	65.9	74.6	81.5	84.8
	10k	2k	66.4	78.7	83.0	85.9
	10k	5k	70.1	80.0	83.4	86.7
	20k	2k	67.2	82.3	85.5	87.3
	20k	5k	74.2	84.3	87.5	89.4
	30k	2k	70.4	82.8	86.2	88.4
	30k	5k	71.7	81.4	86.4	88.3
SYNTCOMP	30k	0	59.3	62.8	66.2	68.3
	50k	0	57.9	64.1	67.6	70.3
	10k	2k	43.4	58.6	62.1	66.2
	10k	5k	50.3	62.1	65.5	66.2
	20k	2k	51.7	64.1	67.6	71.0
	20k	5k	57.2	64.1	65.5	69.0
	30k	2k	56.5	64.8	70.3	75.9
	30k	5k	56.5	63.4	65.5	71.7
Timeouts	30k	0	13.7	24.2	30.4	36.3
	10k	2k	25.8	35.4	38.6	41.0
	10k	5k	29.7	38.6	40.5	43.1
	20k	2k	27.3	35.6	38.4	42.0
	20k	5k	30.2	37.7	40.4	43.7
	30k	2k	25.3	34.6	38.3	41.3
	30k	5k	29.0	37.2	40.1	42.4

B.2 Expert Iteration

Table 13: CodeT5 circuit minimization.

Dataset	Criterion	pass@1	pass@4	pass@8	pass@16
Testset	Correctness	76.0	85.1	87.6	89.6
	Size	75.0	82.7	85.9	88.4
SYNTCOMP	Correctness	55.6	67.1	70.8	73.8
	Size	55.4	59.3	64.4	68.3
Timeouts	Correctness	32.8	41.4	44.0	46.5
	Size	31.6	38.3	41.8	44.9

B.3 ITERATING ON OPEN PROBLEMS

Table 14: Pass rates for CodeT5 on evaluation datasets after 20 000 steps of fine-tuning and 2 000 steps of expert iteration with beam size 4 comparing different probabilities $p_{timeout}$ of including timeouts. Results are averaged over 3 runs.

Dataset	$p_{timeout}$	**pass@1**	**pass@4**	**pass@8**	**pass@16**
`Testset`	0.0	74.8	84.4	87.2	89.3
	0.25	**75.2**	84.3	**87.6**	**89.6**
	0.5	74.5	**84.8**	87.4	89.3
`SYNTCOMP`	0.0	**56.1**	67.8	69.9	72.4
	0.25	55.0	68.1	**71.2**	**73.8**
	0.5	53.5	**68.3**	70.8	71.5
`Timeouts`	0.0	32.1	41.3	43.9	46.5
	0.25	38.0	45.6	48.4	50.7
	0.5	**40.1**	**47.4**	**49.9**	**51.9**

C EXPERT ITERATION ALGORITHM

Algorithm 3 Expert Iteration

```
Require: $D = \{(\varphi_1, C_1), \ldots, (\varphi_n, C_n)\}$
$M_0 \leftarrow \textsc{ImitationLearning}(D)$
for $i \leftarrow 1$ to $n$ do
    Sample $(\varphi, C) \sim D$
    $\{\hat{C}_1, \ldots, \hat{C}_k\} \leftarrow \textsc{Search}(M_{i-1}, \varphi)$
    $C^* \leftarrow \textsc{VerifyAndSelect}(\varphi, \{\hat{C}_1, \ldots, \hat{C}_k\} \cup \{C\})$
    $M_i \leftarrow \textsc{TrainUpdate}(M_{i-1}, (\varphi, C^*))$
```