Decision Making in Synthesis cross Technologies using LSTMs and Transfer Learning

Cunxi Yu
University of Utah
Salt Lake City, UT, USA
cunxi.yu@utah.edu

Wang Zhou
IBM Thomas J. Watson Research Center
Yorktown Heights, NY, USA

ABSTRACT

We propose a general approach that precisely estimates the Quality-of-Result (QoR), such as delay and area, of unseen synthesis flows for specific designs. The main idea is leveraging LSTM-based network to forecast the QoR, where the inputs are synthesis flows represented in novel timed-flow modeling, and QoRs are ground truth. This approach is demonstrated with 1.2 million data points collected using 14nm, 7nm regular-voltage (RVT), and 7nm low-voltage (LVT) technologies with twelve IC designs. The accuracy of predicting the QoRs (delay and area) evaluated using mean absolute prediction error (MAPE). While collecting training data points in EDA can be extremely challenging, we propose to elaborate transfer learning in our approach, which enables accurate predictions cross different technologies and different IC designs. Our transfer learning approach obtains estimation MAPE ≤3.7% over ∼960,000 test points collected on 7nm technologies, with only 100 data points used for training the pre-trained LSTM network using 14nm dataset.

ACM Reference Format:
Cunxi Yu and Wang Zhou. 2020. Decision Making in Synthesis cross Technologies using LSTMs and Transfer Learning. In *2020 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD '20), November 16–20, 2020, Virtual Event, Iceland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3380446.3430638

1 INTRODUCTION

Targeted specialization of functionality in hardware has become arguably the best means for enabling improved compute performance and energy efficiency. However, as the complexity of modern hardware systems explodes, fast and effective hardware explorations are hard to achieve due to the lack of guarantee in the existing in electronic design automation (EDA) toolflow. Several major limitations prevent practical hardware explorations [6, 16, 19]. First, as the hardware design and technology advance, the design space of modern EDA tools has increased dramatically. Besides, evaluating a given design point is extremely time-consuming, such that only a very small sub-space of the large design space can be explored.

Recent years have seen increasing employment of decision intelligence in EDA, which aims to reduce the manual efforts and boost the design closure process in modern toolflows [5, 10, 12–14, 17–19]. For example, various of machine learning (ML) techniques have been used to automatically configure the tool configurations of industrial FPGA toolflow [9, 12, 15] and ASIC toolflow [5, 8, 15, 17, 19]. These works focus on end-to-end tool parameter space exploration, which are guided by ML models trained based on either offline [19] or online datasets [6, 12]. Moreover, exploring the sequence of synthesis transformations (also called synthesis flow) in EDA has been studied in an iterative training-exploration fashion through Convolutional Neural Networks (CNNs) [17] and reinforcement learning [5]. In addition, neural network based image classification and image construction techniques have been leveraged in placement and route (PnR), in order to accelerate design closure in the physical design stage [10, 13, 14, 18]. Specifically, this work focuses on optimizing synthesis flows at logic level, where the problems can be formulated as sequential decision making [5, 17].

While recent studies demonstrate that ML can improve the predictability of different EDA heuristics and toolflows, collecting dataset can be very challenging due to the expensive runtime of EDA tools. Specifically, existing works on optimizing logic synthesis flows [5, 17] either require a large amount of efforts for collecting training dataset, or need expensive reinforcement process for training, which are not less likely to be practical. Thus, this work proposes a novel approach based on a novel modeling of flows along with Long Short-Term Memories (LSTMs) networks and transfer learning, which aims to improve both forecasting performance and reduce the required number of data points for training the models. The main contributions of this paper are: **(1)** A closed formula is introduced to represent the search space for arbitrary types of flows. A *timed-model* that models flow as a discrete sequence using 2-D matrix. **(2)** An LSTM based RNN regression architecture is proposed. The inputs are flows in the timed-model matrix, and ground truth are delay and area collected after technology mapping. **(3)** We propose a transfer-learning approach that adapts the model learned from one technology node to another technology node. This offers the ability to estimate the performance for next/future technology nodes. **(4)** The approach has been demonstrated with ∼1.2 million data points with 14nm, regular-voltage (RVT) 7nm, and low-voltage (LVT) 7nm FinFET technologies, collected with 12 different IC designs. We achieve testing *mean absolute percentage error* (MAPE) ≤2.0% for specific design and technology, and testing MAPE cross technologies and designs is ≤3.7% after transfer learning. **(5)** The datasets and training code will be released publicly.

MLCAD '20, November 16–20, 2020, Virtual Event, Iceland

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3380446.3430638

2 BACKGROUND

2.1 Synthesis flows and Search Space

Synthesis flows are a set of synthesis transformations that apply iteratively to the input designs. The synthesis transformations are mainly involved in three stages of the design flow: high-level synthesis (HLS), logic synthesis (LS) and placement and route (PnR). For different types of electronic designs, the flows need to be changed accordingly. In general, there are two types of flows, *none-repetition* flows and *m-repetition* flows [17]. Given n unique transformations, a flow developed with these transformations is called none-repetition flow if each transformation appears only once. The length of none-repetition flows is n. For m-repetition flows, each transformation appears m times. The length of m-repetition flows is $m{\cdot}n$. In [17], the upper bound of the search space for both types of flows are discussed. For none-repetition flows, a closed representation $n!$ is the upper bound of its search space. An iterative formula is used to describe the search space of m-repetition. However, the upper bound was given as a range without a closed formula representation. A closed formula is introduced to describe the search space of repetition flows. The search space for m-repetition flows is a *multi-set* permutation problem. Specifically, for m-repetition flows with n unique transformations, the search space is shown in Equation 1.

$$\frac{(n \cdot m)!}{(m!)^n} \tag{1}$$

Using the multiset permutation concept, we generalize the formula to describe the search space for any type of flows. Let n be the number of unique transformation, the M-repetition flows, M={$m_1, m_2, ..., m_n$}, where m_i is the number of repetitions of the i^{th} transformation. The total number of possible flows is shown in Equation 2.

$$S(m,n) = \frac{(m_1 + m_2 + \cdots m_n)!}{(m_1!)(m_2!)\cdots(m_n!)} \tag{2}$$

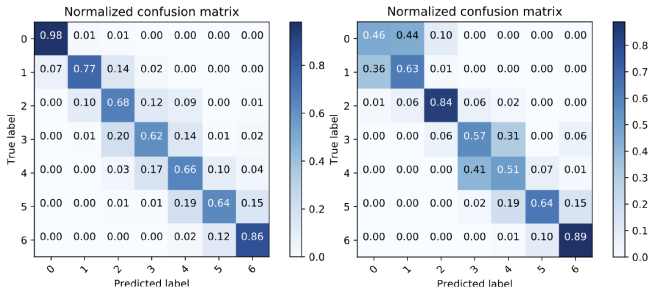
2.2 Recurrent neural network

Recurrent Neural Network (RNN) is a class of artificial neural network with a chain of units in a directed graph sequence. RNNs perform the same computations for each unit in a sequence, and output states depend on the previous states. In theory, RNNs can make use of information in arbitrarily long sequences, but in practice, they are limited to looking back only a few steps, called "Long-Term Dependencies" problem [4]. LSTMs [4] are explicitly designed to address the long-term dependency problem by adding control gates to the recurrent units. Such controlled states are referred to as a gated state or gated memory, which have been implemented as part of LSTMs or other recurrent elements. An RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for "remembering" values over arbitrary time intervals.

2.3 Related Work and Motivating Example

Yu et al. [17] presented a deep learning based approach for generating design-specific synthesis flows for ABC. The main idea of this approach is formulating the flow optimization problem as a *Multiclass Classification* problem. The authors proposed to use Convolutional Neural Network (CNN) based classifier that includes two *Convolution+MaxPool* layers and three *Dense* layers. It shows that the classifier can successfully distinguish the best and worst flows given the design objectives. However, there are two main limitations: **1)** The classifier can only classify the flows into different performance classes, however, it cannot distinguish the performance of different flows within the same class. **2)** The prediction accuracy heavily relies on the labeling rules since the labels of the flows are post-created based on the Quality-of-Result (QoR).

It is obvious that the first limitation comes from the idea of flow classification. Therefore, we focus on illustrating the second limitation. The single-metric and multi-metric rules are introduced in [17] that label the synthesis flows based on single QoR or multiple QoR metrics, such as area, delay, etc. The labeling rule for *seven*-classes labeling requires *six* QoR delimiters. For example, let the six delimiters be the data point at 7%, 20%, 40%, 65%, 80%, and 93% position of training datasets (assuming the training set is sorted from best-to-worse QoR), namely *Labeling rule 1* (Figure 1a). Alternatively, let the six delimiters be the data point at 5%, 15%, 40%, 65%, 90%, and 95% position of training datasets, namely *Labeling rule 2* (Figure 1b). We compare the classification performance of two labeling rules using the CNN architecture and the 64-bit Montgomery Multiplier dataset proposed in [17]. The training and testing of CNN Classifiers are done with Keras using Tensorflow as backend. The results are shown in the confusion matrices in Figure 1.



(a) Labeling rule 1. **(b) Labeling rule 2.**

Figure 1: Confusion matrices of two classifiers that trained with two different labeling rules.

As mentioned in [17], the main tasks are searching for the best (class 0) and worst (class 6) flows for a given design objective. The results show that labeling rule 1 provides 98% accuracy for predicting class 0, and 86% accuracy for class 6; labeling rule 2 provides only 46% accuracy for class 0, and 89% for class 6. We can see that using different labeling rules, the performance of the classifiers can be very different. Note that the labeling rule input to this approach, which should be defined by the user. These two limitations offer the main motivation for our regression based approach.

As described previously, a synthesis flow is a **sequence** of transformations. Mostly in IC design, QoR of a large number of synthesized designs is a continuous variable. To achieve accurate estimation of a continuous variable that is related to sequential behaviors, we explore LSTM network regressor in this work. In the result

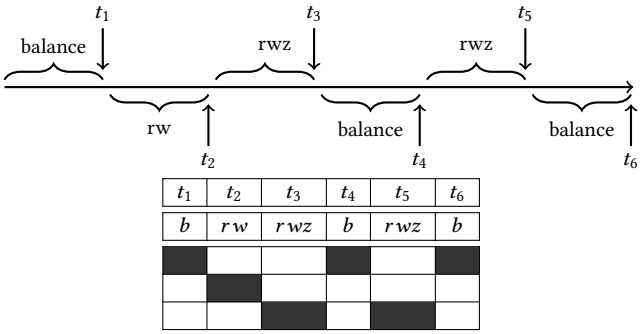
section, we also compare the performance with a regressor built using the proposed CNN model [17].

3 MODELING

This section introduces the inputs of the neural network and ground truth for the LSTM model. The inputs are the synthesis flows that are represented by a 2-D matrix using a novel *Timed-Model*. The ground truth includes delay and area results after technology mapping.

3.1 Inputs: Timed-Model of flows

Table 1: Illustration of timed-model of synthesis flows using ABC default synthesis flow *resyn*. Synthesis transformation at each time spot is shown above the visualized binary matrix.



As mentioned earlier, any flow includes a set of transformations that perform iteratively. We illustrate the concept of timed-model using one common logic synthesis flow provided in ABC [11], ***resyn***, which includes six transformations: *balance (b), rewrite (rw), rewrite -z (rwz), b, rwz, b*. The time-line of applying ***resyn*** to designs is shown in Table 1. Each transformation in this flow is applied to the design at each time frame. For example, for time in range (0,t_1), the transformation *balance* is applied and it finishes at t_1. Then, the second transformation *rewrite* starts and finishes at t_2. The whole flow finishes at t_6. Note that the runtime of different transformations can be very different; and the runtime of the same transformation at different stage could be different as well. In this work, the runtime of each transformation is not included in the modeling. This means that the timed-model of the flows is considered as a discrete sequence. Using one-hot encoding for the three transformations in *resyn*, let *balance*=[1 0 0], *rw* = [0 1 0], and *rwz* = [0 0 1]. The resulting timed-model of *resyn* is shown in Table 1.

A more complex example is shown in Table 2. The input synthesis flow is an ABC synthesis flow including six transformations, *balance (b), restructure (rs), rewrite (rw), refactor (rf), rewrite -z (rwz), refactor -z (rfz)*, and these transformations are repeated four times. The length of this synthesis flow is 24 such that it requires 24 timeframes to complete. With total six transformations, the final model of this synthesis flow is a matrix of shape (6,24) (Figure 2). This type of matrices will be the input to the neural network for training and inference.

3.2 Ground truth

In the context of machine learning, the ground truth is a measurement of the target variable(s) for the training and testing data points. In other words, the ground truth defines the objective(s) of the learning model. In the scenario of training a regressor for synthesis, taking synthesis flows as inputs, the ground truth could be *synthesis runtime*, *critical path delay*, *total logic area*, *XOR counts*, etc. Similarly, this can be extended to other flow performance estimation problem such as placement and route, with the ground truth being *worst negative slacks*, *total negative slacks*, *routing length*, etc. In the result of this paper, the demonstration and evaluation of the proposed approach specifically target on synthesis flows of the open source logic synthesis framework ABC [11]. The ground truth includes *critical path delay* (delay in short) and *logic area* (area in short).

4 APPROACH

This section presents the implementation of LSTM based RNN regressor, training setup and the summary of datasets.

4.1 LSTM network architecture

The RNN regressor architecture is presented in Table 3. The regressor is designed with LSTM×2, Batch Normalization (BN)×4, Dropout×1, and Dense layers×3. The first column shows the layers and its type in a top-down order. The second column presents the output shape of the current layer, and the last column shows the number of parameters in each layer. The activation function of the Dense1 and Dense2 layers is *ReLu*. The output layer is implemented with a dense layer where the number of units equals to the ground truth dimension, *dim*. In this work, the ground truth dimension is one, i.e., either area or delay. The activation function for the last layer is *Linear*.

4.2 Datasets

The datasets are generated by logic synthesis tool ABC [11], with 100,000 random flows generated. All 100,000 flows are applied to three different designs, 64-bit Montgomery Multiplier, 64-bit ALU and 128-bit AES core, using 14nm, 7nm RVT and 7nm LVT technologies. For exploring the transferability cross designs and technologies, we apply the first 20,000 random flows to nine more designs with different IPs (intellectual property) (Table 4), including cryptographic hash *SHA*, RISC (Reduced Instruction Set Computer) architecture Open RISC 1200 *OR1200*, etc. These designs are obtained from OpenCore [1]. Note that some of the random flows fail because of the internal ABC crashes (segment fault reported). There are three failure cases observed while applying to the Montgomery multiplier, and 263 failure cases for the AES core. There are ~300,000 data points generated using 14nm technology with 3 designs, and ~960,000 data points using 7nm technologies with 12 designs. The summary of the datasets is shown in Table 4.

Random flows: Each random flow includes six different transformations, and each transformation can repeat four times (example shown in Table 2), resulting in totally twenty-four transformations in each flow. These 100,000 random flows are generated by randomly permuting these twenty-four transformations.

Table 2: Example of a 24 transformations long 4-repetition synthesis flow using the presented timed-model, with six unique transformations, {*rw, rwz, b, rs, rfz, rf*}. The complete synthesis flow is shown in the second row and synthesis transformation at each time spot is shown above the visualized binary matrix.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}
b	rf	rwz	rw	rs	rfz	b	rw	rwz	rf	rs	rfz	b	rw	rwz	rw	rs	rfz	b	rf	rwz	rfz	rs	rf
			■				■						■	■	■								
		■						■						■						■			
■						■						■						■					
				■						■						■						■	
					■						■						■				■		
	■								■											■			■

Table 3: LSTM based RNN model architecture, including the output shape and number of parameters of each layer.

Layer : Type	Output Shape	# Param
1:LSTM1	(None, 24, 128)	68608
2:BN1	(None, 24, 128)	512
3:LSTM2	(None, 128)	131584
4:BN2	(None, 128)	512
5:Dense1	(None, 30)	3870
6:BN3	(None, 30)	120
7:Dense2	(None, 30)	930
8:BN4	(None, 30)	120
9:Dropout	(None, 30)	0
10:Dense3	(None, dim)	31×dim

Inputs and Labels: The inputs of the neural network are the flows using the timed-model matrix representation with shape (6,24). The labels include the delay/area results by applying the random flow following by ABC technology mapping (command: *map -v*).

Table 4: Summary of Datasets. Data points are generated with 100,000 random flows using three different technology libraries using the first three designs. For the rest of the designs, the data points are collected with 20,000 random flows using the 7nm RVT and LVT FinFET technologies. The ground truth are the QoRs (delay or area) that are collected after technology mapping. **RVT = Regular Voltage Transistor; *LVT = Low Voltage Transistor.*

Design	14nm	7nm RVT	7nm LVT
64-bit Montgomery	99,997	99,997	99,997
64-bit ALU	100,000	100,000	100,000
128-bit AES core	99,737	99,737	99,737
LU8PEng	-	20,000	20,000
Stereovison0	-	20,000	20,000
Stereovison1	-	20,000	20,000
SHA	-	20,000	20,000
raygentop	-	20,000	20,000
OR1200	-	20,000	20,000
Boundtop	-	20,000	20,000
blob_merge	-	20,000	20,000
bgm	-	20,000	20,000

Inputs		Ground truth	
Flow	(6, 24)	Delay/Area	(1,1)

4.3 Training setups and pre-processing

Training setups: The loss function is the *mean squared error* (MSE) and is optimized with *Adam* optimizer [7] with *learning rate*=0.001, β_1=0.9, β_2=0.999. The *batch size* used in this work is 256 and models are trained for 1000 epochs.

Pre-processing (Data normalization): The training data points are normalized before model training. Specifically, the label vectors are normalized by subtracting its *mean* and dividing its *range*. The *mean* and *range* are used to reconstruct the ground truth at testing.

5 TRANSFERABILITY CROSS DESIGNS AND TECHNOLOGIES

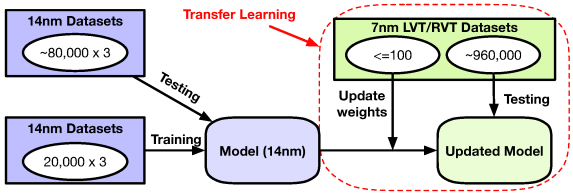


Figure 2: Overview of transfer-learning cross different technologies. Initial model is trained using 20% of the 14nm datasets. The pre-trained model is updated using ≤100 new data points, which are produced using unseen 7nm technologies and IC designs.

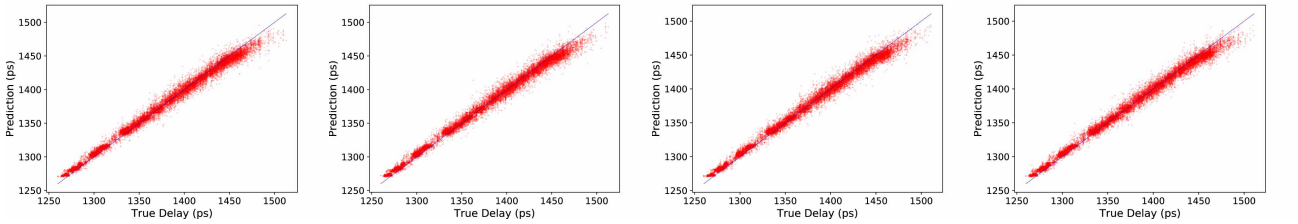
We explore the transferability over different technologies and designs using the approach shown in Figure 2. The main idea is to utilize the model pre-trained with 14nm data points and update the model with little data points to predict for unseen 7nm technologies and designs shown in Table 4. In this work, we restrict the number of data points for transfer learning to be ≤100. Specifically, the results of transfer learning using {10, 25, 50, 100} data points are included in the result section. The evaluations are made using the rest of the 7nm datasets.

5.1 Transfer Learning Strategies

Two transfer learning approaches are implemented.

Updating Dense Layers: This approach takes the pre-trained model and turns the LSTM layers to be non-trainable, i.e., layers 1-4 shown in Table 3. The main intuitions of this approach are that 1) the sequential behavior of the synthesis flows could be similar over different designs and technologies, and 2) the sequential features have been learned mostly in the LSTM layers. In this case, there are only about 5,000 parameters in the pre-trained model that need to be updated during transfer learning.

Updating All Layers: However, if the sequential behaviors of the synthesis flows are different over different designs and technologies, the model could fail to converge without updating the LSTM layers. Hence, this approach updates the parameters of all layers in the pre-trained model.



(a) Sub-set 1: MAPE=0.37% **(b) Sub-set 2: MAPE=0.37%** **(c) Sub-set 3: MAPE=0.36%** **(d) Sub-set 4: MAPE=0.37%**

Figure 3: Visualization of delay prediction with the remaining ~80,000 data points of 14nm Montgomery dataset as testing inputs. Each sub-set includes 20,000 test results (last sub-set includes 19,997). Overall prediction accuracy over 79,997 test points is 99.6%.

6 RESULT

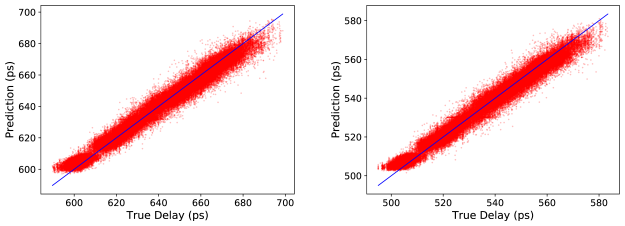
First, the pre-trained model is evaluated with 14nm datasets by evaluating the delay and area prediction accuracy, with 20,000 data points for training and ~80,000 for evaluation for each design (Table 4). Secondly, we evaluate our transfer learning approaches on the delay and area of the 7nm RVT/LVT datasets. The training and testing are conducted on a server with 28 Intel Xeon CPU E5-2690 v4 processors and 256 GB memory. The experiments are implemented in Python3 using Tensorflow-Keras [2].

6.1 Evaluation within 14nm datasets

The results in this section use the training setups provided in Section Approach, with epochs=1000, and 20% training data for validation. The training results using the 14nm datasets (delay) of 64-bit Montgomery multiplier, in which the training data points fit perfectly after 1000 epochs with MAPE 0.25 (N-MAPE will be 99.75). The model is then tested using the remaining 80,000 14nm data points. The testing results are shown in Figure 3. The testing dataset is randomly split into four subsets, with each sub-set including ~20,000 points. **Unlike most machine learning tasks, we intentionally use small portion of the dataset to train the network and the remaining dataset for testing in order to demonstrate the generalizability of the proposed approach. This is particularly important for EDA tasks since collecting data points is very challenging.** This is used to demonstrate that the prediction accuracy does not differ much while choosing different inputs. The MAPEs for delay prediction are 0.37, 0.37, 0.36, and 0.37, with an overall MAPE 0.37. Similarly, we evaluate our approach using the 14nm datasets of all three designs, with average prediction MAPE≤ 2.0% with ~80,000 data points for each QoR objective (see Table 4).

6.2 Transfer Learning: cross designs and technologies

This section presents the evaluation results of transfer learning using the approaches shown in Figure 2. The testing datasets are the 7nm datasets shown in Table 4. We first show the complete delay prediction results using 64-bit GF multiplier design. The initial model is pre-trained with 20,000 14nm data points. Then, we update the weights of all layers of this model with 100 7nm data points from the same design. The rest data points for each technology are used for testing. The prediction accuracy is ≥99.5% for both 7nm technologies. Note that these show the transferability cross the technologies only.



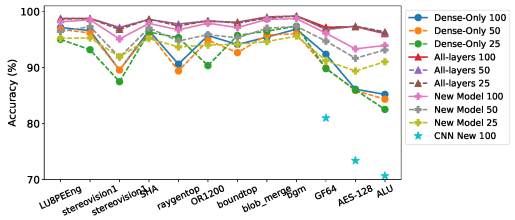
(a) 100k 7nm RVT predictions. MAPE = 0.5 (N-MAPE = 99.5) **(b) 100k 7nm LVT predictions. MAPE = 0.4 (N-MAPE = 99.6)**

Figure 4: Visualization of 7nm delay predictions using transfer learning with prediction accuracy ≥99.5%. Initial model is trained with 64-bit GF multiplier 14nm datasets, and is updated with 100 7nm data points.

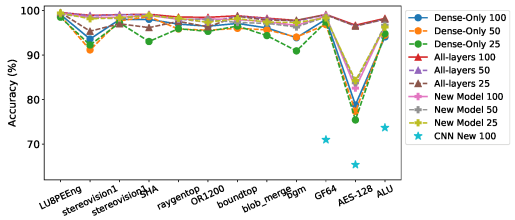
To explore the transferability cross both designs and technologies, we apply transfer learning to all 12 designs with the initial model pre-trained with 14nm GF delay dataset. Note that industrial studies indicate that machine learning based electronic design systems require a minimum of 95% accuracy for performance estimation [3]. More importantly, these systems are required to be stable (i.e., have similar accuracy) for different types of designs. The results of two approaches, updating dense layers only and updating all layers, are included in Figure 5. To demonstrate the advantages of transfer learning, the results of training a new model from scratch using the same amount of data points, *without* pre-training on 14nm data first, are included as a baseline. It shows that transfer learning by updating all layers provides the best results over all designs, for delay and area estimations. Our approach obtains MAPE≤3.7% (N-MAPE≥96.3%) for delay and area over all designs, while model is trained with only 100 data points collected on 7nm. In comparison to transfer learning approaches, training a new model without transfer learning yields much worse accuracy due to insufficient training data. This indicates that transfer learning is helpful when the size of available training data is limited.

We also compare our LSTM network with the CNN based approach. We modify the model released in [17] by 1) changing the the

output layer from multi-channel *softmax* output to single-channel *linear* output and 2) adding BN following the Convolutional layers. For fair comparison, we only compare the delay/area estimation accurate for GF, AES, and ALU designs that were used in that work. It shows that with 100 training data points, CNN regressor performs much worse than the LSTM regressor (Figure 5). The transfer-learning results using CNN approach are not included since they perform worse than training new CNN model.



(a) N-MAPE of Delay Prediction.



(b) N-MAPE of Area Prediction.

Figure 5: Evaluation of two transfer learning approaches using 25/50/100 data points.

Finally, we try to find the minimum number of data points for transfer learning to achieve reasonable accuracy. Specifically, we choose the approach of updating all layers, and set the number of training data points to 5/10/25. The results are shown in Figure 6. For both delay and area, the estimation accuracy significantly decreases with $\leq$10 data points. This suggests that at least 25 data points are needed for the proposed transfer learning approach to achieve stable estimation accuracy cross different designs and technologies.

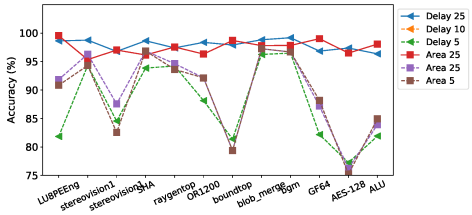


Figure 6: Evaluation of transfer learning (updating all layers) for delay and area estimation using 5/10/25 data points.

7 CONCLUSION

This paper presents an RNN regression based approach that precisely estimates the delay and area of synthesis flows. The proposed RNN regressor is constructed using LSTM network with batch normalization and dense layers. To enable accurate predictions for future technologies and different designs, we propose a transfer-learning approach that utilizes the pre-trained model and requires much less training data. The demonstrations are made with logic synthesis tool ABC using 14nm and 7nm FinFET technologies, and models are tested over 1.2 million data points. The results show the prediction accuracy of delay and area is $\geq$98.0% for single technology, and the prediction accuracy after transfer-learning cross designs and technologies is $\geq$96.3% with only 100 new data points. This demonstrates that the proposed transfer learning approach can effectively learn to estimate QoR for unseen technologies and designs. Future work will focus on performance estimations at physical layout level (e.g., silicon routing congestion), and 5nm technologies.

REFERENCES

[1] [n.d.]. OpenCores. *URL https://opencores.org* ([n. d.]).

[2] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467* (2016).

[3] Jeff Dyck. 2018. Mentor, A Siemens Business: Production Ready Machine Learning for EDA. In *Design Automation Conference (DAC'18)*.

[4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[5] Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 2019. DRiLLS: Deep Reinforcement Learning for Logic Synthesis. *arXiv preprint arXiv:1911.04021* (2019).

[6] Nachiket Kapre, Harnhua Ng, Kirvy Teo, and Jaco Naude. 2015. InTime: A Machine Learning Approach for Efficient Selection of FPGA CAD Tool Parameters. (Feb. 2015).

[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] Dandan Li, Shuzhen Yao, Yu-Hang Liu, Senzhang Wang, and Xian-He Sun. 2016. Efficient Design Space Exploration via Statistical Sampling and AdaBoost Learning. *DAC* (2016).

[9] Shuangnan Liu, Francis CM Lau, and Benjamin Carrion Schafer. 2019. Accelerating fpga prototyping through predictive model-based hls design space exploration. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[10] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, et al. 2020. Chip Placement with Deep Reinforcement Learning. *arXiv preprint arXiv:2004.10746* (2020).

[11] Alan Mishchenko et al. [n.d.]. ABC: A System for Sequential Synthesis and Verification. *URL http://www. eecs. berkeley. edu/alanmi/abc* ([n. d.]).

[12] Benjamin Carrion Schafer and Zi Wang. 2019. High-level synthesis design space exploration: Past, present and future. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2019).

[13] Haoyu Yang, Shuhe Li, Yuzhe Ma, Bei Yu, and Evangeline FY Young. 2018. GAN-OPC: Mask optimization with lithography-guided generative adversarial nets. In *DAC*.

[14] Haoyu Yang, Piyush Pathak, Frank Gennari, Ya-Chieh Lai, and Bei Yu. 2019. DeePattern: Layout pattern generation with transforming convolutional auto-encoder. In *DAC'2019*. 1–6.

[15] Cunxi Yu. 2020. FlowTune: Practical Multi-armed Bandits in Boolean Optimization. In *International Conference On Computer Aided Design (ICCAD'2020)*. IEEE/ACM, 234–241.

[16] Cunxi Yu, Chau-Chin Huang, Gi-Joon Nam, Mihir Choudhury, Victor N Kravets, Andrew Sullivan, Maciej Ciesielski, and Giovanni De Micheli. 2018. End-to-end industrial study of retiming. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 203–208.

[17] Cunxi Yu, Houping Xiao, and Giovanni De Micheli. 2018. Developing synthesis flows without human knowledge. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*. 50:1–50:6.

[18] Cunxi Yu and Zhiru Zhang. 2019. Painting on placement: Forecasting routing congestion using conditional generative adversarial nets. In *Proceedings of the 56th Annual Design Automation Conference 2019*. 1–6.

[19] Matthew M. Ziegler, Ramon Bertran Monfort, Alper Buyuktosunoglu, and Pradip Bose. 2017. Machine Learning Techniques for Taming the Complexity of Modern Hardware Design. *IBM Journal of Research and Development* 61, 4 (2017), 13.