GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization

Conference Paper · November 2019

DOI: 10.1109/ICCAD45719.2019.8942063

5 authors, including:

Jeehyun Lee
Georgia Institute of Technology
2 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Anthony D. Agnesina
Georgia Institute of Technology
10 PUBLICATIONS **14** CITATIONS
SEE PROFILE

Kambiz Samadi
Qualcomm
50 PUBLICATIONS **2,093** CITATIONS
SEE PROFILE

Sung Kyu Lim
Georgia Institute of Technology
385 PUBLICATIONS **5,707** CITATIONS
SEE PROFILE

GAN-CTS: A Generative Adversarial Framework for Clock Tree Prediction and Optimization

Yi-Chen Lu[1], Jeehyun Lee[1], Anthony Agnesina[1], Kambiz Samadi[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA
[2]Qualcomm Technologies, Inc., San Diego, CA
yclu@gatech.edu

***Abstract*—In this paper, we propose a complete framework named GAN-CTS which utilizes conditional generative adversarial network (GAN) and reinforcement learning to predict and optimize clock tree synthesis (CTS) outcomes. To precisely characterize different netlists, we leverage transfer learning to extract design features directly from placement images. Based on the proposed framework, we further quantitatively interpret the importance of each CTS input parameter subject to various design objectives. Finally, to prove the generality of our framework, we conduct experiments on the *unseen* netlists which are not utilized in the training process. Experimental results performed on industrial designs demonstrate that our framework (1) achieves an average prediction error of 3%, (2) improves the commercial tool's auto-generated clock tree by 51.5% in clock power, 18.5% in clock wirelength, 5.3% in the maximum skew, and (3) reaches an F1-score of 0.952 in the classification task of determining successful and failed CTS processes.**

I. Introduction

Clock tree synthesis (CTS) is a critical stage of physical design, since clock network often constitutes a high percentage of the overall power in the final full-chip design. An optimized clock tree helps to avoid serious design issues such as excessive power consumption, high routing congestion, elongated timing closure, etc [3]. However, due to the high complexity and run time of electronic design automation (EDA) tools, designers are struggling to synthesize high quality clock trees that optimize key desired metrics such as clock power, skew, clock wirelength, etc. To find the input parameters that achieve desired CTS outcomes, designers have to search in a wide range of candidate parameters, which is usually fulfilled in a manual and highly time-consuming calibration fashion.

To automate this task and alleviate the burden for designers, several machine learning (ML) techniques have been proposed to predict the clock network metrics. Previous work [10] utilizes data mining tools to estimate the achieved skew and insertion delay. The authors of [8] employ statistical learning and meta-modeling methods to predict more essential metrics such as clock power and clock wirelength. The authors of [9] consider the effect of non-uniform sinks and different aspect ratios. A recent work [12] utilizes artificial neural networks (ANNs) to predict the transient clock power consumption based on the estimation of clock tree components. However, all the previous works only focus on enhancing the performance of prediction. There are four highly crucial aspects that all of them neglect, namely:

- **Generality of Model:** No previous work conducts the prediction experiments on the *unseen* netlists. They utilize the same netlists during training and testing stages.
- **Interpretability of Parameters:** No previous work *interprets* the importance of each CTS input parameter subject to distinct CTS outcomes.
- **Optimization of Metrics:** No previous work demonstrates the capability to *optimize* the CTS outcomes with respect to various design objectives.
- **Suggestion of Inputs:** Given a design, no previous work exhibits the ability to *suggest* the CTS input parameters sets that achieve optimized clock trees. They merely adopt discriminative approaches to model the CTS stage.

These aspects which previous works leave under-addressed are the most crucial factors that determine whether the proposed modeling methods are practical or not. In this paper, we solve all the issues presented above. Furthermore, a unique aspect of our work is that we directly extract design features from *placement layout images* to perform practical CTS outcomes predictions in ultra-high precision. We demonstrate that the features extracted from the layouts are indeed highly useful to improve the accuracy of our models.

The goal of this work is to construct a general and practical CTS framework which owns the capability to predict CTS outcomes in high precision as well as the facility to recommend designers the input parameters sets leading to optimized clock trees even for *unseen* netlists. Our proposed framework consists of a regression model and a variation of generative adversarial network (GAN) [5] named conditional GAN [14].

Apart from the conventional GAN training process, in this work, a reinforcement learning is introduced to the generator, and a multi-task learning is presented to the discriminator. The reinforcement learning is conducted by the regression model which is trained prior to the conditional GAN. It serves as a supervisor to guide the generator to produce the CTS input parameters sets that lead to optimized clock trees. As for the discriminator, it not only predicts whether the input is generated or real as the conventional approach, but also predicts if the input results in a successful CTS run or not. In this paper, a successful CTS run indicates that the achieved clock tree outperforms the one auto-generated by the commercial tool.

The rest of the paper is organized as follows. Section II presents a solid overview of our framework. Section III

978-1-7281-2350-9/19/$31.00

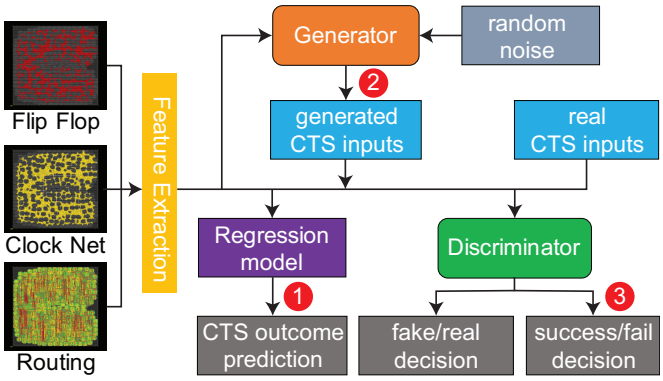


Fig. 1: A high-level view of this work and the three objectives we have achieved. The first objective is to predict the CTS outcomes in high precision. The second objective is to recommend designers good CTS input settings. The third objective is to determine whether the input settings outperform commercial tool's auto-setting.

describes the problem formulations, database generation, and identifies the difficulties of the problems. Section IV illustrates the algorithms and the detailed structures of the models. Experimental results are demonstrated in Section V, and finally we conclude our work in Section VI.

II. Overview

It has been widely acknowledged that GAN is a promising model that learns the complicated real-world data through a generative approach [17]. A vanilla GAN structure contains a generator and a discriminator which are both neural networks. The goal of the generator is to generate meaningful data from a given distribution (e.g. random noise), while the objective of the discriminator is to distinguish the *generated* samples from the *real* samples in the database. Upon the vanilla GAN structure, in conditional GAN, an external conditional input is further introduced to both generator and discriminator. This conditional input enables the model to direct the data generation process, which benefits us to generate suitable CTS input parameters sets with respect to different benchmarks and even the ones which are not utilized in the training process.

A high-level view of our framework named GAN-CTS is shown in Figure 1. The framework is comprised of three training stages. The first training stage is to extract key design features from placement images which represent flip flop distribution, clock net distribution, and trial routing result. Note that trial routing is a process performed in the end of the placement stage, which provides an estimation of the routing congestion in a given placement. To extract features from images, we introduce transfer learning to a pre-trained convolutional neural network (CNN) named ResNet-50 [6], which is a 50-layer residual network.

Following from the feature extraction process, we utilize the extracted features as well as the clock trees in the database to train the regression model and to predict CTS outcomes. The regression model is a multi-output deep neural network which predicts multiple CTS metrics simultaneously. In this work, we select three essential CTS metrics that characterize a clock tree to predict and to further optimize in later processes. The three selected metrics are clock power, clock wirelength, and the achieved maximum skew. As mentioned in Section I, we do not consider our model as a black box as previous works. To further interpret the predictions made by the regression model, we leverage a gradient-based attribution method [1] to quantitatively interpret the importance of each CTS input parameter subject to different target outcomes.

The last training stage of the framework involves a reinforcement learning and a generative adversarial learning. We leverage a conditional GAN to perform the CTS optimization and classification tasks. The regression model trained prior at this stage now acts as a supervisor to guide the generator to generate CTS input parameters sets which lead to optimized clock trees. The guidance is conducted by a reinforcement learning technique named policy gradient [16].

In conditional GAN, we take the extracted placement features as the conditional input, where the original inputs of the vanilla GAN model are considered as regular inputs. The great advantage of having the conditional input is that it allows the model to control the *modes* of the generated data. In this work, we consider different benchmarks as different *modes*. Therefore, with the aid of the conditional approach, our framework has the facility to optimize *unseen* benchmarks which are not utilized during the training process.

Finally, a highlight of our framework is that a multi-task learning is conducted in the discriminator. In addition to the conventional task of distinguishing between generated and real samples, we introduce a new task of classifying successful and failed CTS runs. In this paper, we strictly define a CTS run as successful if two out of the three achieved target CTS metrics mentioned earlier are better than the ones achieved automatically by the commercial tool.

In the end of the training process, we acquire four models as follows:

- A placement feature extractor E which precisely characterizes different designs from placement images.
- A regression model R which performs high precision predictions of target CTS outcomes.
- A generator G which generates CTS input parameters sets that lead to optimized clock trees.
- A discriminator D which predicts the success and failure of CTS runs.

III. Designing Experiments

We formally define the clock tree synthesis (CTS) prediction and optimization problems as follows:

Problem 1 (CTS Outcomes Prediction). *Given a pre-CTS placement P and a CTS input parameters set X, predict the post-CTS outcomes Y without performing any actual CTS.*

Problem 2 (CTS Outcomes Optimization). *Given a pre-CTS placement P, generate a CTS input parameters set $\hat{X}$ that leads to optimized CTS outcomes $\hat{Y}$.*

TABLE I: Our benchmarks and their attributes.

Netlist Name	# Nets	# Flip Flops	# Total Cells
AES-128	90,905	10,688	113,168
Arm-Cortex-M0	13,267	1,334	12,942
NOVA	138,171	29,122	136,537
ECG	85,058	14,018	84,127
JPEG	231,934	37,642	219,064
LDPC	42,018	2,048	39,377
TATE	185,379	31,409	184,601

TABLE II: Modeling parameters we use and their values.

type	parameters	values or ranges
placement	aspect ratio	$\{0.5, 0.75, 1.0, 1.25, 1.5\}$
	utilization	$\{0.4, 0.45, 0.5, \cdots, 0.7\}$
CTS	max skew (ns)	[0.01, 0.2]
	max fanout	$[50, 250]$
	max cap trunk (pF)	[0.05, 0.3]
	max cap leaf (pF)	[0.05, 0.3]
	max slew trunk (ns)	[0.03, 0.3]
	max slew leaf (ns)	[0.03, 0.3]
	max latency (ns)	[0, 1]
	max early routing layer	$\{2, 3, 4, 5, 6\}$
	min early routing layer	$\{1, 2, 3, 4, 5\}$
	max buffer density	[0.3, 0.8]

A. Database Construction

To build the database, we utilize *Synopsys Design Compiler 2015* to synthesize the netlists and leverage *Cadence Innovus Implementation System v18.1* to perform the placement and CTS processes. The database is constructed based on *TSMC-28nm* technology node. Table I shows the netlists utilized in our work and their attributes after synthesized at *1125*MHZ.

Table II defines the modeling features and their ranges of values that we utilize in our work. The ranges of the CTS related parameters are determined by reasonably widening the commercial tool's auto-setting values. Among the modeling features, aspect ratio and utilization rate represent physical structures. The min and max early routing layers (min $\leq$ max) indicate the metal layers utilized in early global routing (EGR), which is a procedure to reserve space for future detailed signal routing during the CTS stage.

The combinations of the two placement related parameters give us 35 different placements per netlist. By running CTS with randomly sampled CTS input parameters, we generate 100 clock trees per placement. Therefore, in total, we have $24.5k$ datum across 7 different netlists in our database. To substantiate the generality of our framework, in the experiments, we only utilize 5 netlists (ARM, ECG, JPEG, LDPC, TATE) in the training process and perform the validations on the rest 2 unseen netlists (AES, NOVA).

B. Database Analysis

Figure 2 shows the total power distribution of all $24.5k$ clock trees in the database. It demonstrates the variety of our database as well as the difficulty to model the CTS process across different benchmarks. Table III demonstrates the complicated impact of different input slew constraints on essential CTS metrics. We observe that if a single tighter slew constraint is merely given on leaf cells (from design A to design B), the total number of inserted buffers drastically decreases. However, if tighter slew constraints are given on both trunk cells and leaf cells (from design A to design C), more buffers are inserted compared with the previous approach. In summary, the above analyses show that the behavior of CTS is very sophisticated, which implies great benefits to employ ML methods in modeling the CTS process.

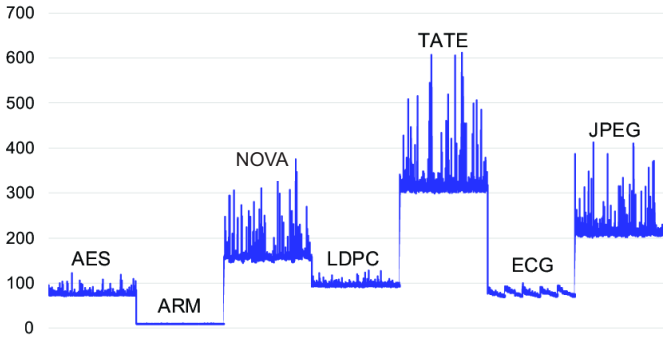


Fig. 2: Total power (mW) distribution among 24.5k full-chip designs of all 7 benchmarks in our database.

TABLE III: Clock trees with different input slew constraints for Nova benchmark.

	tree A	**tree B**	**tree C**
max trunk slew (ns)	0.1	0.1	0.05
max leaf slew (ns)	0.1	0.05	0.05
# inserted buffers	2,556	600	1,124
total power (mW)	164.7	154.5	158.8
clock power (mW)	27.9	22.6	24.9
clock wirelength (mm)	118.7	97.3	101.6
maximum skew (ns)	0.06	0.1	0.07

In this paper, we consider the clock tree auto-generated by the commercial tool as a baseline to evaluate the trees generated by our framework. As mentioned in Section II, we define a CTS run as successful if two out of the three achieved target metrics, which are clock power, clock wirelength, and clock skew, are better than the ones of the auto-generated tree. In Section V, we demonstrate the CTS metrics and layout comparisons between the optimized clock trees generated by our framework and the one auto-generated by the commercial tool.

IV. GAN-CTS Algorithms

In this section, we first describe the process of feature extraction. Then, we present our methodologies to solve the CTS outcomes prediction and optimization problems. In the meantime, we illustrate the detailed structures of the models. In the end, we summary the overall training process.

A. Placement Image Feature Extraction

One of the innovations of this work is that we directly utilize placement images as inputs to predict and optimize CTS outcomes. The key rationale is that placement images contain important information of designs. Previous work [19] has demonstrated the efficiency of using placement images to predict routability and number of design rule violations (DRVs). In this work, we leverage the extracted features

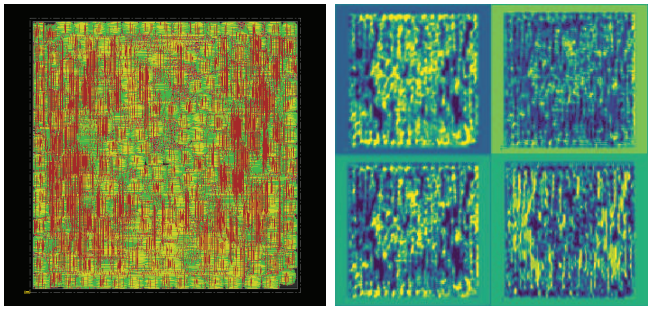


Fig. 3: Visualization of trial routing image in convolutional filters.

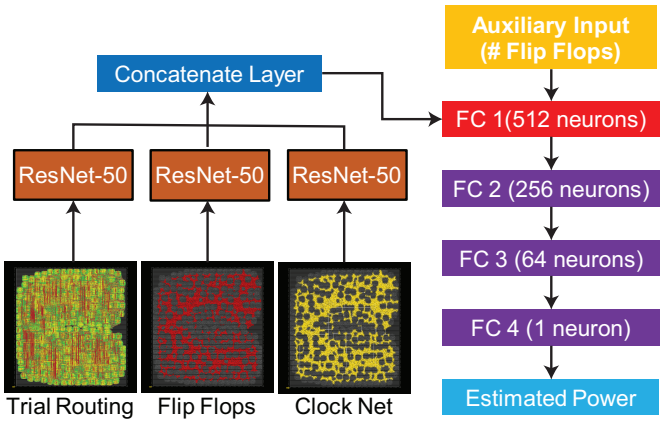


Fig. 4: Our image feature extraction flow. The extracted features are colored in red.

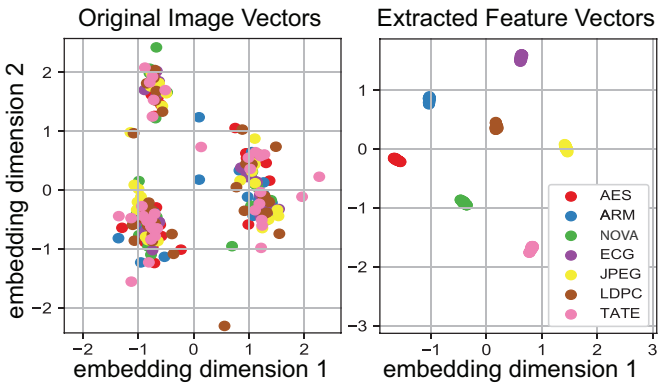


Fig. 5: t-SNE visualizations of the original placement image vectors and the extracted feature vectors.

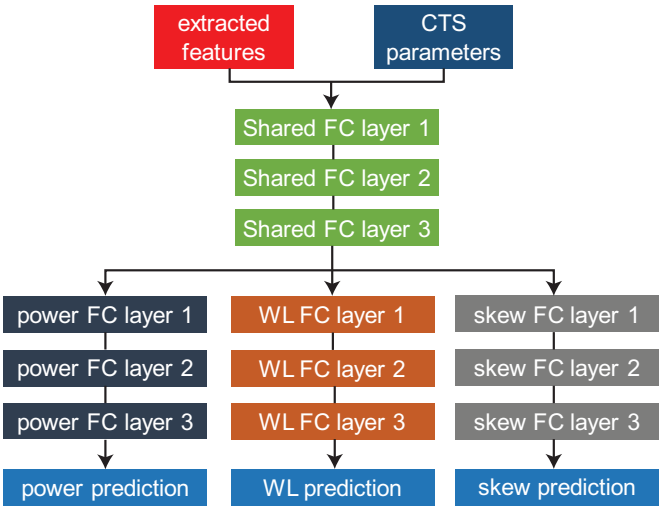


Fig. 6: Our multi-output regression model.

from placement images to solve CTS outcomes prediction and optimization problems. Our approach is built upon convolutional neural network (CNN) and transfer learning, where we devise our own fully connected (FC) layers upon the pre-trained model named ResNet-50 [6], which is a CNN-based model pre-trained on the well-known ImageNet [4] dataset with millions of images.

Figure 3 shows the visualization of a trial routing image passing through four convolutional filters inside the pre-trained ResNet-50 model. In the figure, we observe that important information such as usage of metal layers is well captured. Despite that ResNet-50 is powerful on many image datasets, it is not devised specifically for physical design problems. Therefore, to extract key design features from the high dimensional vectors, we devise a feature extraction flow with transfer learning as shown in Figure 4. Four self-designed FC layers are appended on top of the ResNet-50 model to predict the commercial tool's estimation of total power right after the placement stage. As shown in the figure, since placement images cannot precisely reflect the actual size of a design, we introduce an auxiliary input with one dimension to our FC layers which represents the total number of flip flops. In the training process, we fix the parameters of the pre-trained ResNet-50 model and only update the parameters of the FC layers. After training, for each pre-CTS placement, we take the output vector of the first FC layer with 512 dimensions denoted in red in the figure as the final extracted feature vector.

In the implementation, all the images are in a dimension of 700 x 717 x 3. The achieved mean absolute percentage error of the unseen validation netlists to predict the estimated power is only 1%. However, since a low prediction error does not guarantee a good feature representation, we leverage a dimension reduction technique named t-distributed stochastic neighbor embedding (t-SNE) [13] to visualize the extracted features ($\in R^{512}$) in R^2 as shown in Figure 5. In the figure, we observe that different placements belonging to the same netlist are clustered together and those belonging to different netlists are well separated. Therefore, we conclude that the extracted features well capture the characteristics of a design.

B. CTS Outcomes Prediction

Constructing a precise regression model is the key step to solve Problem 1. Following the feature extraction process, we train the regression model with the extracted features to predict the target CTS outcomes. As mentioned in Section II, in this paper, we target at predicting and optimizing three

target CTS outcomes: clock power, clock wirelength, and the maximum skew. The regression model is a multi-output deep neural network as shown in Figure 6. The model takes the extracted features along with the CTS parameters as inputs and outputs three predictions simultaneously. Note that a multi-task learning is introduced in here. The reason we leverage a single multi-output model rather than multiple single-output models as previous work [12] is that different CTS outcomes are not independent of each other. Therefore, shared layers enable different objectives to own mutual information, which reduces the training time as well as enhances the performance of each prediction. In the training process, we utilize mean squared error as the loss function, and leverage dropout layers inside the model to prevent it from overfitting. The validation results of this experiment are shown in Section V.

In this work, we do not treat the regression model as a black box as previous works. Instead, we leverage a gradient-based attribution algorithm named DeepLIFT [15] to interpret the predictions. The algorithm aims to determine the attribution (relevance) value of each input neuron subject to different outputs. Assume our regression model R takes an input vector $x = [x_1, ..., x_N] \in R^N$ and produces an output vector $S = [S_1, ..., S_k]$. DeepLIFT proceeds a_i^l, the attribution of neuron i at layer l, in a backward fashion by calculating the activation difference of the neuron between the target input x and reference input $\hat{x}$. The procedure is derived as

$$a_i^L = \begin{cases} S_i(x) - S_i(\hat{x}) & \text{if neuron i is the output of interest} \\ 0 & \text{otherwise,} \end{cases} \tag{1}$$

$$a_i^l = \sum_j \frac{z_{ji} - \hat{z}_{ji}}{\sum_i z_{ji} - \sum_i \hat{z}_{ji}} \cdot a_j^{l+1}, \tag{2}$$

where L denotes the output layer, and z_{ji} is the weighted activation of neuron i onto neuron j in the next layer. In the implementation, we take the reference input $\hat{x}$ as the input parameters of the auto-generated clock tree.

C. CTS Optimization

To solve Problem 2, we introduce a reinforcement learning to the conditional GAN by taking the pre-trained regression model as a supervisor of the generator. Before illustrating the objectives of the optimization process, we first describe the structure of the generator. The generator G is a neural network parameterized by θ_g which samples a regular input z with 100 dimensions from a $N(0, 1)$ Gaussian distribution p_z and samples the extracted placement features f from the database p_d as the conditional input.

Figure 7 shows the detailed structure of the generator. The leaky ReLU [20] layers are employed as activation functions of the input and hidden layers to project latent variables onto a wider domain, which eliminates the bearing of vanishing gradients. Batch normalization [7] layers are utilized to normalize the inputs of each hidden layer to zero-mean and unit-variance, which accelerates the training process since the oscillation of gradient descent is reduced. Finally, for the output layer, the number of neurons D denotes the number

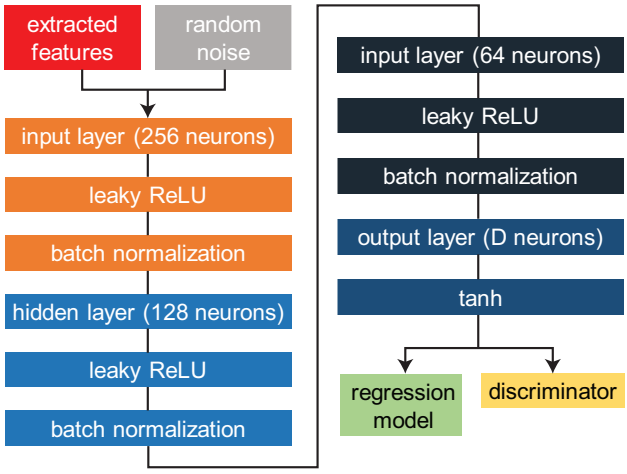


Fig. 7: Detailed structure of our GAN generator.

of CTS modeling parameters, and a hyperbolic tangent layer is chosen as the activation function to match the domain of the normalized samples drawn from the database. Since when training the discriminator, we normalize the real samples x from the database to $\hat{x} \in [-1, 1]$ as

$$\hat{x} = \frac{x}{\max_{x \in supp(x)}(x)} \times 2 - 1. \tag{3}$$

In GAN-CTS, the generator has two important objectives. The first is to generate realistic samples that deceive the discriminator, where the corresponding objective function is

$$\mathcal{L}_{G_D} = \mathbb{E}_{z,f}[log(D(G(z, f))]. \tag{4}$$

The second objective is to generate the samples that lead to optimized clock trees by maximizing reward r from the pre-trained regression model R. The reward r is defined as

$$r := R(G(z, f)) = -\prod_{i=1}^{N} \frac{R_i(G(z, f))}{\text{auto-setting result of target } i}, \tag{5}$$

where N denotes the number of target CTS outcomes and R_i denotes the corresponding prediction of the regression model.

To optimize the reward defined in Equation 5, we leverage a policy gradient algorithm named REINFORCE [18]. The algorithm considers the generator as an agent that strives to maximize the accumulative reward by producing wise actions $G(z, f)$ from a given environment created by the discriminator. The corresponding objective is derived as

$$\mathcal{L}_{G_P} = \mathbb{E}_{z,f}\ [r \cdot ln(\mu(G(z, f)))], \tag{6}$$

where μ is a softmax function that maps $G(z, f)$ to $[0, 1]$. With the above objective function, the generator is expected to generate the CTS input parameters sets $G(z, f)$ that maximize the rewards r, which results in optimized clock trees.

Finally, by combining the two objective functions illustrated, we formulate the training process of the generator as

$$\max_G \mathbb{E}_{\substack{z \sim p_z \\ f \sim p_d}}[log(D(G(z, f))) + r \cdot ln(\mu(G(z, f)))]. \tag{7}$$

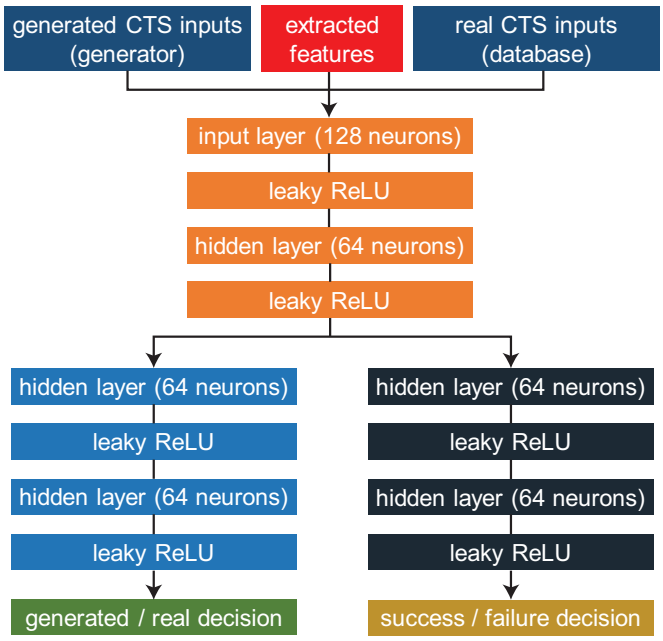


Fig. 8: Detailed structure of our GAN discriminator.

D. Success vs. Failure Classification

The classification of successful and failed CTS runs are performed in the discriminator. To describe how the classification task works, we first illustrate the structure of the discriminator as shown in Figure 8. The discriminator takes a regular input which is either the generated samples $G(z, f; \theta_g)$ or the real samples x from the database p_d, and a conditional input which denotes the features extracted from placement images. Note that when the regular input represents the real samples x, the conditional input f must be aligned to x. The reason we introduce a conditional input to the discriminator is that it helps the discriminator to distinguish better between the generated and the real samples under different *modes* (benchmarks). As shown in Equation 3, since the unit of each CTS input parameter is different, we normalize real samples x from the database to $\hat{x} \in [-1, 1]$ to improve the stability of the GAN training process.

In GAN-CTS, the discriminator has two objectives. One is to distinguish the generated samples from the real samples, which is derived as

$$\begin{aligned}\mathcal{L}_{D_g} =& \mathbb{E}_{(x,f)\sim p_d}[log(D_g(x, f))] \\ &+ \mathbb{E}_{\substack{z\sim p_z \\ f\sim p_d}}[log(1 - D_g(G(z, f)))].\end{aligned} \tag{8}$$

The other is to classify whether a CTS run is successful or not, which is formulated as

$$\mathcal{L}_{D_s} = \mathbb{E}_{x\sim p_d}[log(D_s(x, f))]. \tag{9}$$

Note that the definition of successful and failed CTS runs are defined in Section II. The reason we introduce a multi-task learning to the discriminator is that the attributes of the two objectives are similar, where both of them are performing binary classification. Therefore, some latent features can be shared in the early network as shown in Figure 8. Finally, the training process of the discriminator is summarized as

$$\begin{aligned}\max_D \ & \mathbb{E}_{(x,f)\sim p_d}[log(D_g(x, f)) + log(D_s(x, f))] \\ &+ \mathbb{E}_{\substack{z\sim p_z \\ f\sim p_d}}[log(1 - D_g(G(z, f)))].\end{aligned} \tag{10}$$

Algorithm 1 GAN-CTS training methodology.
We use default values of $\alpha_r = 0.0001$, $\alpha_{GAN} = 0.0001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $m = 128$.

Input: $\{f\}$: extracted placement features, $\{x\}$: training data, $\{Y\}$: target CTS metrics for prediction and optimization.
Input: α_r: learning rate of regression model, α_{GAN}: learning rate of GAN, m: batch size, $\{\theta_r\}_0$: initial parameters of regression model, θ_{g_0}: initial parameters of generator, $\{\theta_d\}_0$: initial parameters of discriminator, $\{\beta_1, \beta_2\}$: Adam parameters.
Output: R: regression model, G: generator, D: discriminator.

1: $N \leftarrow length(y)$
2: **while** $\{\theta_r\}$ do not converge **do**
3: Sample a batch of training data $\{x^{(i)}\}_{i=1}^m \sim p_d$
4: Take features $\{f^{(i)}\}_{i=1}^m$ corresponding to $\{x^{(i)}\}_{i=1}^m$
5: **for** $k \leftarrow 1$ to N **do**
6: $g_{r_k} \leftarrow \nabla_r[\frac{1}{m}\sum_{i=1}^m (R_k(f^{(i)}, x^{(i)}) - Y_k^{(i)})^2]$
7: $\theta_{r_k} \leftarrow Adam(\alpha_R, \theta_{r_k}, g_{r_k}, \beta_1, \beta_2)$
8: **end for**
9: **end while**
10: **while** θ_g and θ_d do not converge **do**
11: Sample a batch of training data $\{x^{(i)}\}_{i=1}^m \sim p_d$
12: Take features $\{f^{(i)}\}_{i=1}^m$ corresponding to $\{x^{(i)}\}_{i=1}^m$
13: Sample a batch of random vectors $\{z^{(i)}\}_{i=1}^m \sim p_z$
14: $g_{d_1} \leftarrow \nabla_{\theta_{d_1}}[\frac{1}{m}\sum_{i=1}^m log(D_{\theta_{d_1}}(x^{(i)}, f^{(i)})) + \frac{1}{m}\sum_{i=1}^m log(1 - D_{\theta_{d_1}}(G_{\theta_g}(z^{(i)}, f^{(i)})))]$
15: $\theta_{d_1} \leftarrow Adam(\alpha_{GAN}, \theta_{d_1}, g_{d_1}, \beta_1, \beta_2)$
16: $g_{d_2} \leftarrow \nabla_{\theta_{d_2}}[\frac{1}{m}\sum_{i=1}^m log(D_{\theta_{d_2}}(x^{(i)}, f^{(i)}))]$
17: $\theta_{d_2} \leftarrow Adam(\alpha_{GAN}, \theta_{d_2}, g_{d_2}, \beta_1, \beta_2)$
18: Sample a batch of random vectors $\{z^{(i)}\}_{i=1}^m \sim p_z$
19: Sample a batch of features $\{f^{(i)}\}_{i=1}^m$
20: $g_{gen} \leftarrow -\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^m log(D_{\theta_{d_1}}(G_{\theta_g}(z^{(i)}, f^{(i)})))$
21: $\theta_g \leftarrow Adam(\alpha_{GAN}, \theta_g, g_{gen}, \beta_1, \beta_2)$
22: $r \leftarrow \prod_{k=1}^N \frac{R_k(G(\{z^{(i)}\}_{i=1}^m, \{f^{(i)}\}_{i=1}^m))}{\text{auto-setting result of outcome } k}$
23: $g_\theta \leftarrow \nabla_\theta \frac{1}{m}\sum_{i=1}^m r \cdot ln(\mu(G(z^{(i)}, f^{(i)})))$
24: $\theta_g \leftarrow Adam(\alpha_{GAN}, \theta_g, g_{gen}, \beta_1, \beta_2)$
25: **end while**

E. Training Methodology

Based on the structures and the objective functions presented, we now illustrate the training process of our framework in Algorithm 1, where a gradient descent optimizer Adam [11] is utilized across different training stages. First, we train the regression model (lines 2-9) which serves as a supervisor in the training process of conditional GAN. Then, we train the generator and discriminator alternatively (lines 10-25), since the two networks have antagonistic objectives. The parameters of the discriminator are split into θ_{d_1} and θ_{d_2}

TABLE IV: Mean absolute percentage error (MAPE) (%) and correlation coefficient (CC) of our predictor tested using *unseen netlist*.

unseen	clock power		clock wirelength		achieved skew	
netlist	MAPE	CC	MAPE	CC	MAPE	CC
AES	1.26	0.978	1.64	0.986	3.57	0.965
NOVA	1.19	0.982	1.24	0.981	2.66	0.979

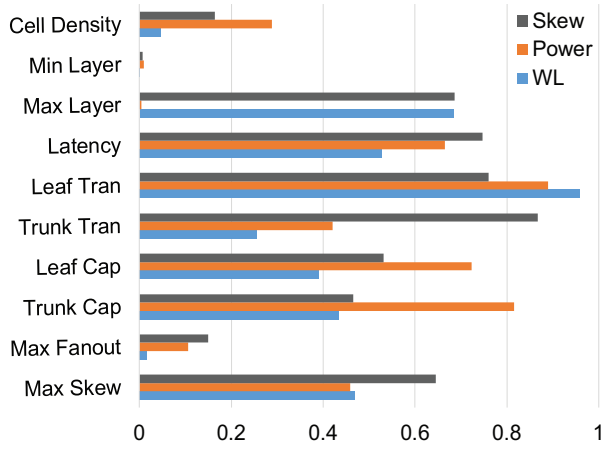


Fig. 9: Relative importance of CTS input parameters on skew, power, and wirelength for AES benchmark.

($\theta_{d_1} \cap \theta_{d_2} \neq \phi$) to represent different tasks, since a multi-task learning is conducted. The overall training process is done when the losses of the generator and the discriminator reach an equilibrium, which takes about 6 hours on a machine with 2.40 GHz CPU and a NVIDIA RTX 2070 graphics card.

V. Experimental Results

In this section, we describe several experiments that demonstrate the achievements of GAN-CTS framework. The framework is implemented in *Python3* with *Keras* [2] library. As mentioned in Section III, we utilize *Cadence Innovus* to generate a database containing $24.5k$ clock trees with 245 different placements across 7 netlists under *TSMC-28nm* technology node. All the netlists utilized are from *OpenCores.org*. To prove the generality of our framework, we only use 5 netlists during the training process, and leverage the remaining two (AES, NOVA) to perform the validation.

A. CTS Prediction and Interpretation Results

In this experiment, we evaluate the regression model on three target CTS outcomes with two evaluation metrics: the mean absolute percentage error (MAPE) and the correlation coefficient (CC). Table IV shows the evaluation results. It is shown that the regression model achieves very low evaluation errors among all target metrics on the *unseen* netlists, which substantiates that the regression model predicts in high precision and earns a good generality.

However, accurate prediction results are not sufficient for designers to trust a model. Understanding the reasons behind the predictions is crucial. As shown in Figure 9, we evaluate the importance of each CTS input parameter based on the predictions presented in Table IV. As mentioned in Section IV-B, we define the importance through a gradient-based attribution method named DeepLIFT [15]. The algorithm quantifies the relevance of input parameters with respect to different outputs. Since the input of the regression model contains the CTS input parameters and the extracted features, in this interpretation experiment, we focus on determining the relative importance among CTS input parameters by further normalizing the relevance scores to $[0, 1]$. Note that the normalization is performed within the CTS input parameters. Below, we explain two important phenomenons observed from Figure 9.

1) The slew constraint for leaf cells has great impacts on clock power and clock wirelength. Indeed, with a tight slew constraint, more buffers need to be inserted to meet the timing target, which ultimately results in higher clock power and clock wirelength.
2) The max EGR layer has high impacts on maximum skew and clock wirelength. The reason is that signal nets are often routed in top metal layers (e.g. M5, M6). If signal nets are forced to route in low metal layers (e.g. M1. M2) that are reserved to route clock nets, there will be many detours in clock routing, which results in long clock paths and hence a large maximum skew.

B. CTS Optimization Results

In this experiment, we demonstrate the optimization results achieved by our GAN-CTS framework, where a joint optimization is performed on three target CTS metrics: clock wirelength, clock power, and the maximum skew. Figure 10 shows the optimization result of AES benchmark, where the blue dots denote the original clock trees in the database, and the red stars represent the clock trees generated by GAN-CTS. To plot the figure, we first take the extracted features of the pre-CTS placements as conditional inputs, and then utilize the generator to suggest 200 sets of CTS input parameters. With these suggested parameters sets, we further leverage the commercial tool to perform actual CTS processes. Finally, according to the target optimization metrics, we plot the scatter distributions of the clock trees suggested by GAN-CTS together with the ones originally generated in the database. Note that the input parameters of the clock trees in the database are randomly sampled from the ranges shown in Table II.

Table V further shows the detailed comparisons between the commercial tool auto-generated clock tree and the selected GAN-CTS generated clock trees, where the selection is conducted by taking the tree with minimum clock power. It is shown that compared with the auto-generated clock tree, the clock trees generated by GAN-CTS significantly reduce many crucial CTS metrics. Figure 11 further exhibits the layout comparison of AES benchmark, where the clock wirelength of the GAN-CTS optimized tree is much shorter than the one auto-generated by the commercial engine.

C. Success vs. Failure Classification Results

In the final experiment, we demonstrate the classification results achieved by the discriminator of determining successful and failed CTS runs. As mentioned in Section II, success and

TABLE V: Commercial auto-setting vs. GAN-CTS comparison.

netlist	CTS Metrics	Auto-Setting	GAN-CTS
aes	# inserted buffers	695	83 (-88.1%)
	clock power (mW)	20.34	**9.86 (-51.5%)**
	clock wirelength (mm)	34.09	27.78 (-18.5%)
	maximum skew (ns)	0.019	0.018 (-5.3 %)
nova	# inserted buffers	2617	524 (-79.9%)
	clock power (mW)	43.59	**19.86 (-54.4%)**
	clock wirelength (mm)	118.24	97.92 (-17.2%)
	maximum skew (ns)	0.031	0.033 (+6.4 %)

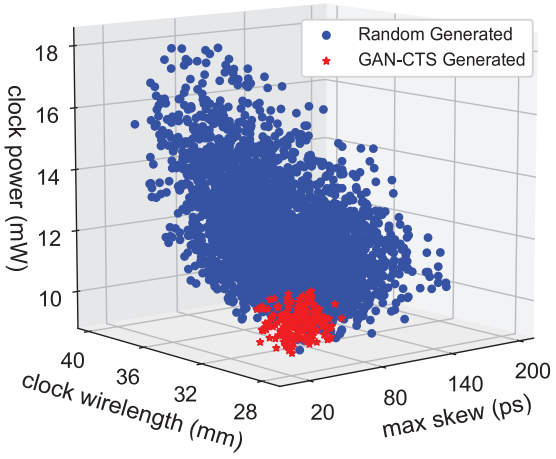


Fig. 10: Distributions of random generated vs. GAN-CTS generated clock trees for AES benchmark.

failure are defined by comparing the CTS metrics of the clock trees generated by GAN-CTS to the one auto-generated by the commercial tool. Table VI summarizes the classification results in a confusion matrix with NOVA benchmark. The accuracy and the F1-score are 0.947 and 0.952, respectively.

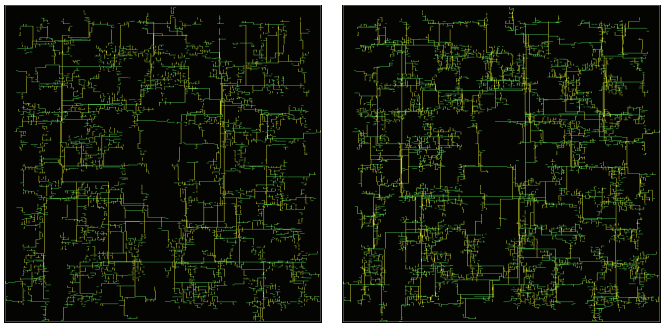
VI. Conclusion and Future Work

In this paper, we have proposed a complete framework named GAN-CTS to solve the complicated CTS outcomes optimization and input generation problems which all previous works leave under-addressed. The fact that the proposed framework demonstrates promising results on the *unseen* benchmarks proves that it is general and practical.

In spite of the superior performance achieved, the proposed framework is currently dependent on the technology node. In the future, we aim to overcome this dependence.

VII. Acknowledgment

This work is supported by the National Science Foundation under Grant No. CNS 16-24731 and the industry members of the Center for Advanced Electronics in Machine Learning.



(a) GAN-CTS optimized (b) commercial auto-setting

Fig. 11: Clock tree layout comparison with AES benchmark.

TABLE VI: Confusion matrix of success vs. failure classification in NOVA benchmark. Failure means worse than auto-setting.

		Predictions		
		Success	Failure	Total
Ground Truths	Success	1848	111	1959
	Failure	74	1453	1527
	Total	1922	1564	3486

References

[1] M. Ancona et al. Towards better understanding of gradient-based attribution methods for deep neural networks. In *International Conference on Learning Representations*, 2018.

[2] F. Chollet et al. Keras, 2015.

[3] A. Datli, U. Eksi, and G. Isik. A clock tree synthesis flow tailored for low power. In *https://www.design-reuse.com/articles/33873/clock-tree-synthesis-flow-tailored-for-low-power.html*, 2013.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, 2009.

[5] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv:1502.03167*, 2015.

[8] A. B. Kahng, B. Lin, and S. Nath. Enhanced metamodeling techniques for high-dimensional ic design estimation problems. In *Proceedings of the Conference on Design, Automation and Test in Europe*, 2013.

[9] A. B. Kahng, B. Lin, and S. Nath. High-dimensional metamodeling for prediction of clock tree synthesis outcomes. In *System Level Interconnect Prediction (SLIP), ACM/IEEE International Workshop on*. IEEE, 2013.

[10] A. B. Kahng and S. Mantik. A system for automatic recording and prediction of design quality metrics. In *isqed*. IEEE, 2001.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Y. Kwon, J. Jung, I. Han, and Y. Shin. Transient clock power estimation of pre-cts netlist. In *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*. IEEE, 2018.

[13] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov), 2008.

[14] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[15] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje. Not just a black box: Learning important features through propagating activation differences. *arXiv preprint arXiv:1605.01713*, 2016.

[16] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 2000.

[17] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F.-Y. Wang. Generative adversarial networks: introduction and outlook. *IEEE/CAA Journal of Automatica Sinica*, 4(4):588–598, 2017.

[18] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4), 1992.

[19] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu. Routenet: routability prediction for mixed-size designs using convolutional neural network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018.

[20] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv:1505.00853*, 2015.