RL-CCD: Concurrent Clock and Data Optimization using Attention-Based Self-Supervised Reinforcement Learning

Yi-Chen Lu[1], Wei-Ting Chan[2], Deyuan Guo[3], Sudipto Kundu[3], Vishal Khandelwal[2], and Sung Kyu Lim[1]
[1]School of ECE, Georgia Institute of Technology, Atlanta, GA
[2]Synopsys Inc., Hillsboro, OR; [3]Synopsys Inc., Mountain View, CA
{yclu, limsk}@gatech.edu; {wei-ting.chan, deyuan.guo, sudipto.kundu, vishal.khandelwal}@synopsys.com;

***Abstract*—Concurrent Clock and Data (CCD) optimization is a well-adopted approach in modern commercial tools that resolves timing violations using a mixture of clock skewing and delay fixing strategies. However, existing CCD algorithms are flawed. Particularly, they fail to prioritize violating endpoints for different optimization strategies correctly, leading to flow-wise globally sub-optimal results. In this paper, we overcome this issue by presenting RL-CCD, a Reinforcement Learning (RL) agent that selects endpoints for useful skew prioritization using the proposed EP-GNN, an endpoint-oriented Graph Neural Network (GNN) model, and a Transformer-based self-supervised attention mechanism. Experimental results on 19 industrial designs in $5-12nm$ technologies demonstrate that RL-CCD achieves up to 64% Total Negative Slack (TNS) reduction and 66.5% number of violating endpoints (NVE) improvement over the native implementation of a commercial tool.**

I. Introduction

Modern Physical Design (PD) tools interleave clock skewing and delay (logic) fixing strategies to perform timing optimization, which is often termed as Concurrent Clock and Data (CCD) optimization. In general, CCD aims to find an optimal balance between "clock" and "logic" optimization so as to resolve violating timing endpoints in a flow-wise globally optimized manner. However, existing CCD algorithms fail to achieve this goal, mainly because they neglect the following vital fact:

- **Not all violating endpoints are equal.** Different violating endpoints have distinct sensitivity for various optimization strategies. To truly achieve global optimal results, some of them are better to be "fixed more" by clock-path optimization (i.e., a larger portion of their slack values should be resolved by clock fixing), while others are better to be "fixed more" by data-path amendment. However, existing CCD algorithms have no intelligence on weighing the balance between different strategies in endpoint level, leading to flow-wise sub-optimal results.

In this paper, we overcome this critical issue by presenting RL-CCD, a Reinforcement Learning (RL) agent that performs intelligent endpoint prioritization. RL-CCD is built upon a customized Graph Neural Networks (GNNs) named EP-GNN for endpoint encoding, and an attention-based encoder-decoder network using self-supervised learning [9]. Prior to CCD optimization, RL-CCD selects a subset of violating endpoints that should be prioritized for clock-path optimization using useful skew (rather than data-path optimization using buffering, sizing, restructuring etc.), to achieve flow-wise globally optimal Power, Performance, and Area (PPA) metrics.

The goal of this work is to unleash the true power of CCD optimization in commercial tools using RL. Note that modern PD tools preform CCD optimization throughout the entire PD flow. To demonstrate the effectiveness of our RL-CCD framework, we specifically focus on improving the timing quality of CCD optimization at the placement stage, where the goal is to achieve better Total Negative Slack (TNS) values at the end of the entire placement optimization, which involves CCD and other optimization techniques. In this work, we take an industry-leading commercial PD tool as our reference tool, Synopsys IC Compiler II (ICC2), and demonstrate that RL-CCD significantly improves the tool's native implementation flow.

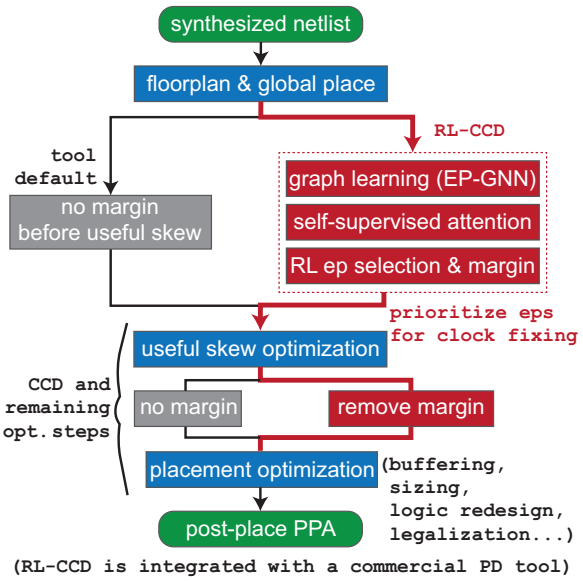

Fig. 1: Default tool flow vs. our RL-enhanced flow that performs endpoint prioritization using graph learning and self-supervised attention [9].

Figure 1 highlights the innovations of our framework and the key differences over the native implementation of the reference commercial tool. Given a globally placed netlist, unlike the default tool flow that has no intelligence on balancing CCD optimization techniques via endpoint prioritization, our RL agent selects a group of violating endpoints that should be prioritized for clock-path optimization using useful skew. Note that the total optimization steps between the left flow (default) and the right flow (ours) are exactly the same. Except for endpoint prioritization using margin (which is removed after useful skew), RL-CCD is not taking any additional optimization step.

The outcome of our effort is a *universal* (i.e., generalize to *any* design and technology) RL-based framework, which *drastically* improves the CCD optimization quality of an industry-leading commercial tool. Our main contributions are as follows:

- We discover a new PD problem and demonstrate its importance. That is, finding a balance between clock-path and data-path optimization through endpoint prioritization in commercial tool flows so as to reach flow-wise globally optimal results.
- We present RL-CCD, the first-ever endpoint prioritization framework that focuses on improving timing quality of CCD optimization in commercial tools. RL-CCD achieves up to **64%** TNS improvements (avg. **23%**), and up to **66%** number of violating endpoints (NVE) reduction (avg.**19%**) on **19** industrial designs

979-8-3503-2348-1/23/$31.00 ©2023 IEEE

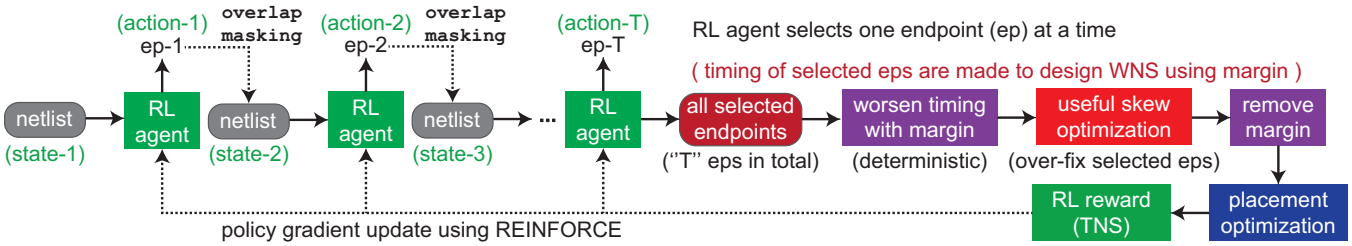

Fig. 2: High-level overview of our framework. At each RL time step (training iteration), our agent selects one endpoint at a time and mask out other endpoints based on overlapping calculation. The selection process completes when all violating endpoints are either masked or selected. Then, with the RL-selected endpoints, we apply margin to worsen their timing to design Worst Negative Slack (WNS) prior to the useful skew optimization so that they can be "over-fixed" by clock arrival adjustments. The applied margins are removed before entering the remaining placement optimization steps, which involves optimization techniques such as buffering, sizing, logic restructuring, legalization etc. Finally, the achieved TNS value is taken as the *RL reward* of the current *trajectory* to update framework parameters using a policy gradient-based algorithm named REINFORCE [12].

in advanced technologies ($5nm$,$7nm$,$12nm$).

- RL-CCD facilitates transfer learning and self-supervised learning, which makes it generalizable to any design or technology. A pre-trained RL-CCD agent can significantly improve the optimization results with only few iterations of training.

II. Related Works

A. Predictive Useful Skew

Useful skew is a well-known technique that improves design timing by adjusting clock arrival time. However, as pointed out in [11], computed skew adjustments often require redo synthesis or placement to truly realize timing benefits. To break the chicken-egg problem (i.e., iterative back annotation of skew), the authors of [1] proposed a "predictive" useful skew technique for one-pass timing optimization, where the TNS value can be improved by up to 5%. In this paper, since commercial PD tools have well-integrated useful skew techniques into CCD optimization and reach considerable success, we are not focusing on improving the native useful skew implementation in tools. Our specific focus is to balance clock-path and data-path optimization techniques offered by commercial tools via endpoint prioritization, which is an under-researched problem.

B. Learning-Driven Timing Prediction

Fast and accurate timing prediction methods are essential to improve design productivity. Previous work [7] presented a tree-based technique to predict the time-consuming post-route path-based timing analysis (PBA) results from pre-route graph-based analysis (GBA) data. Another work [2] presented a timing engine inspired GNN-based framework for slack and arrival time prediction on timing endpoints. Recently, the authors of [8] explored the use of Transformer [9] to perform gate sizing for timing optimization, where a 1400x speed up is achieved in obtaining tool-accurate sizing moves on unseen netlists. Nonetheless, in this paper, we are not comparing RL-CCD with above methods as the focus of RL is to improve optimization quality rather than prediction accuracy.

C. RL in EDA: Going Beyond Commercial Tool Quality

As the benefit of scaling saturates, leading-edge commercial tools are seeking more powerful methods for PPA optimization even at the cost of runtime. RL thus becomes a promising solution as it does not require any labeled data and has been demonstrated to achieve never-seen, high-quality optimization results in many fields [12]. In PD, the authors of [6] developed an RL agent for floorplanning, which generates superhuman floorplans that human labor is not able to achieve. Another work [5] proposed RL-Sizer to tackle the VLSI gate sizing problem, a traditional EDA optimization task. It is shown that RL-Sizer can outperform the default sizing algorithms in a commercial tool although it adopts a fundamentally different approach. With the above success stories, in this paper, we decide to continue the research of RL in PD, aiming at going beyond what state-of-the-art commercial tools are able to achieve.

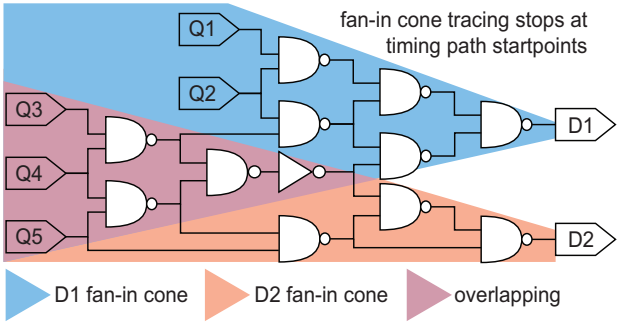

Fig. 3: Illustration of endpoint fan-in cone overlapping. Note that the fan-in cone tracing of an endpoint stops at its previous startpoints. The overlapping ratio is calculated as dividing the number of overlapped cells by the total number of fan-in cone cells.

III. RL-CCD Framework

Given a globally placed netlist $G = (V,E)$, the ultimate goal of RL-CCD is to select a group of violating endpoints $V' \in V$ to be prioritized for useful skew optimization, such that the TNS value after the proceeding placement optimization steps can be optimized. Note that V' is an empty set in the native implementation of the reference commercial tool. In this paper, we demonstrate that by selecting proper V', the achieved TNS value can be drastically improved.

A. Overview and Reinforcement Learning Formulation

Figure 2 shows a high-level overview of our RL endpoint selection process. As aforementioned, the goal of our RL agent, RL-CCD, is to select the endpoints (colored in red) that should be prioritized for clock-path optimization, The key idea behind is to let the useful skew engine "over-fix" the timing of the RL-selected endpoints so that the proceeding delay (logic) optimization techniques can spend less effort on them, which together result in flow-wise optimal solutions. We understand another route may also work (i.e., useful skew "under-fix"), however, we empirically observe that the proposed method (i.e., useful skew "over-fix") works significantly better.

Our endpoint prioritization problem is a combinatorial optimization problem. In this paper, we choose to formulate it as a Markov

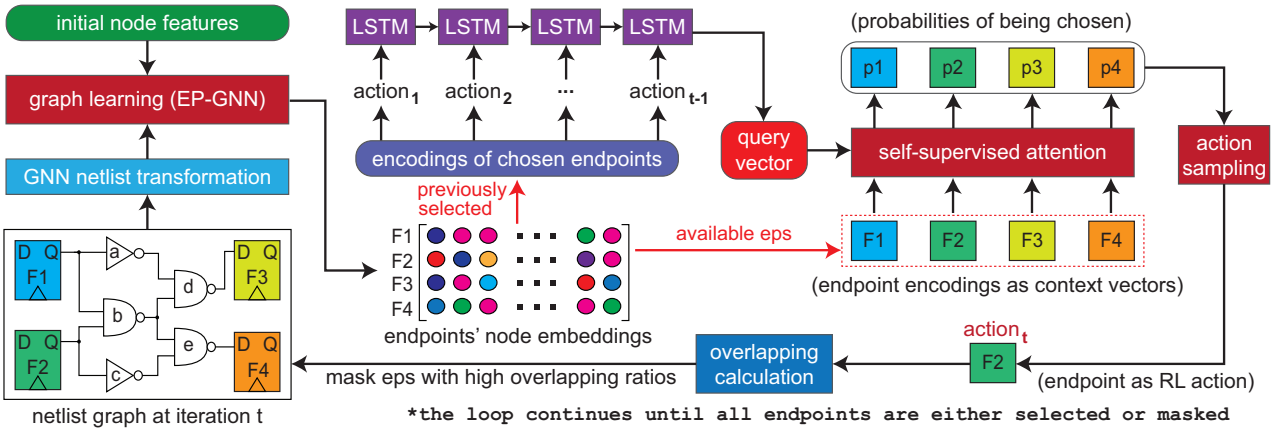

Fig. 4: Illustration of RL-CCD endpoint selection process. At each time step t, RL-CCD first leverages the proposed EP-GNN model to obtain endpoint embeddings $F_{EP}^{(t)} = \{F_e, \forall e \in EP\}$, which is considered as the *RL state* s_t. Then, based on the embeddings, a LSTM network is utilized as an encoder to encode past actions $\{a_{t-1}\}$ sequentially. Its final hidden vector h_t is taken as the query vector q_t for the downstream attention-based decoder network. Using a self-supervised attention mechanism [9], the decoder takes the query vector q_t and current endpoint embeddings $F_{EP}^{(t)}$ as inputs and outputs a probability vector $P_t \in R^{|EP|}$ which is used to sample one endpoint (i.e., action a_t) at current iteration. An overlapping calculation is followed to mask out other endpoints whose fan-in cones have an overlapping ratio higher than a pre-defined threshold ρ with the selected endpoint (Figure 3). The features (Table I) are updated accordingly and the loop continues until all endpoints are either selected or masked.

Decision Process (MDP) and use RL algorithms to solve it. Below, we formally describe the formulation in key MDP terminologies:

- *States (s)*: A state s_t represents the status of the endpoint set $EP \in V$ in the netlist graph $G = (V, E)$, which is encoded by the proposed EP-GNN framework via fan-in cone aggregation.
- *Actions (a)*: An action a_t refers to the endpoint being selected.
- *State Transition*: By taking an action a_t in a state s_t, the probability distribution over the next state s_{t+1}.
- *Reward (r)*: In our settings, the rewards are zero for intermediate actions $\{a_1..a_{T-1}\}$ except for the last action a_T, which represents the final achieved TNS value after the entire placement optimization, including CCD and other optimization techniques.
- *Trajectory (τ)*: A trajectory τ refers to a complete selection process from time step $t = 1$ to $t = T$, where at each time step t, there is a corresponding state s_t, action a_t, and reward r_t pair denoted as (s_t, a_t, r_t).

The ultimate goal of our RL agent, RL-CCD, is to obtain an optimal policy π that maximizes the expected return J at the end of a trajectory $R(\tau)$, which can be denoted as:

$$\max_{\theta} J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)], \tag{1}$$

where π_θ denotes the policy parameterized by θ which represents all trainable parameters of the RL-CCD framework.

B. Detailed Architecture

RL-CCD consists of three main components: (1) a GNN module that generates endpoint embeddings, (2) a Long Short-Term Memory (LSTM) [3] network for past actions encoding, and (3) a self-supervised attention module to decode actions from probabilities. Note that each of them is hardly independent of each other. Figure 4 depicts the details of how they function together in one RL time step t, forming a circular selection loop. Below, we describe each main component in detail:

*1) **Netlist Encoding using GNNs**:* GNNs have shown promising results in advancing many traditional PD tasks thanks to their ability to perform effective graph representation learning [4]. In this paper, we present EP-GNN, an endpoint-oriented GNN framework that focuses on generating node embeddings of timing path endpoints through iterative neighborhood and fan-in cone aggregation.

TABLE I: Initial node features for EP-GNN endpoint encoding. Note that the first attribute "RL masked" will be updated in each RL training iteration based on the selection of new endpoint and overlapping calculation.

name	# dim.	description
RL masked	1	is selected or masked by RL-CCD
locations	2	cell (x,y) location in global placement
outNet cap	1	output net capacitance
load cap	1	sum of driving load capacitance
cell cap	1	cell input capacitance
cell power	2	cell internal power and leakage power
net power	1	output net switching power
max toggle	1	maximum toggle rate at output pin
wst slack	1	worst slack of paths through cell
wst output slew	1	worst output transition
wst input slew	1	worst input transition

Prior to the actual graph learning, we first construct GNN message passing edges using the netlist transformation technique proposed in [4]. Then, for each node in the transformed graph, we hand-craft a comprehensive list of features as shown in Table I, which include timing, power, and physical attributes. With the message passing edges and the initial features defined, we leverage the proposed EP-GNN framework to obtain endpoint embeddings.

Our EP-GNN framework has three graph convolution layers and one fully-connected (FC) layer. All graph convolution layers have the same hidden dimension, and each of them transforms the node features $\{f_v, \forall v \in V\}$ from layer $l-1$ to layer l as follows:

$$f_v^l = \sigma\left(\gamma f_v^{l-1} \cdot \Theta_{proj} + (1-\gamma) \cdot \Theta_{agg}\left(\frac{1}{|N(v)|}\sum_{j \in N(v)} f_j^{l-1}\right)\right), \tag{2}$$

where σ denotes sigmoid function, $N(v)$ denotes the local neighborhood of node v, γ denotes the trainable parameter that weighs the importance between the self-projection and neighborhood-aggregation operations that are parameterized by Θ_{proj} and Θ_{agg} respectively, which are both realized by neural networks. After completing the graph convolution, a FC layer Θ_{FC} is followed to compute the final

representations of each endpoint e among the endpoint set EP as:

$$f_e = \Theta_{FC}\left(f_e^{l=3} + \sum_{j \in cone(e)} f_j^{l=3}\right), \tag{3}$$

where $cone(e)$ denotes the fan-in cone of the endpoint e. In our implementation, the graph convolution layer has a dimension of 32, and the final FC layer has a dimension of 16. Hence, the generated endpoint embeddings are in 16 dimensions.

Since the masking mechanism based on overlapping calculation change some node features (i.e., "RL masked" in Table I) after each selection, the graph learning by EP-GNN is conducted in every RL time step t, where the computed endpoint embeddings $F_{EP}^{(t)} = \{f_e^{(t)}, \forall e \in EP\}$ are considered as the *RL state* s_t. These endpoint embeddings are taken as the inputs to the downstream LSTM-based encoder network and the self-supervised attention module to decide the next endpoint to select (i.e., *RL action* a_t).

*2) **Past Actions Encoding using LSTM**:* Our encoder-decoder structure as shown in Figure 4 is inspired by the renowned Transformer architecture proposed in [9]. In this paper, we customize the renowned architecture to solve our specific problem by replacing the encoder with a LSTM network to encode the past *RL actions*, and by simplifying the attention mechanism to focus on generating the probability distribution of RL actions. Our effort significantly reduces the number of parameters required for training, making the framework fully applicable to industrial designs with millions of instances.

At each time step t, the goal of our LSTM-based encoder is to generate a query vector q_t for the proceeding decoder network by sequentially encoding the past actions taken in all previous time steps (i.e., a_1 to a_{t-1}). The rationale behind using LSTM [3], a renowned sequence encoding network, for past actions encoding is that the decision of each selection is made sequentially, and each of them should not be independent of each other. Hence, at each training iteration, our LSTM network takes the EP-GNN node embeddings of the previously selected endpoints $\{f_{a_{t-1}}\}$ and the previous hidden vector h_{t-1} as inputs, and outputs a new hidden vector h_t that is taken as the query vector q_t to the decoder as:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, a_{t-1}] + b_i), \quad f_t = \sigma(W_f \cdot [h_{t-1}, a_{t-1}] + b_f), \\ o_t &= \sigma(W_o \cdot [h_{t-1}, a_{t-1}] + b_o), \quad \tilde{c}_t = \tanh(W_c [h_{t-1}, x_t] + b_c), \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t, \quad h_t = o_t \odot \tanh(c_t), \quad q_t = h_t, \end{aligned} \tag{4}$$

where i, f, o, c the input gate, forget gate, output gate, cell gate, respectively, $\{W\}$ denotes the trainable weights. Note that the hidden vector h_t is passed to both the decoder network at the current time step t and the LSTM-encoder itself in the next time step $t+1$.

*3) **Current Action Decoding using Self-Supervised Attention**:* The goal of the decoder network is to generate a probability vector $P_t \in R^{|EP|}$, where each element $P_t^{(i)}$ represents the probability of an endpoint i being selected at the current time step t. To efficiently consider all endpoints at once for the selection, in this paper, we leverage a self-supervised attention mechanism [9] to build the decoder network. Inspired from from pointer networks [10], given a query vector q_t and GNN embeddings $F_{EP}^{(t)} \in R^{|EP| \text{x} 16}$ of all endpoints in the design $\{f_e, \forall e \in EP\}$, each element $A_t^{(i)}$ in the final attention vector $A_t \in R^{|EP|}$ is computed as:

$$A_t^{(i)} = \begin{cases} v^T \tanh\left(W_1 \cdot F_{EP}^{(t)} + W_2 \cdot q_t\right) & \text{if ep-}i\text{ is valid} \\ -\infty & \text{otherwise,} \end{cases} \tag{5}$$

where v, W_1, and W_2 are the learning parameters of the self-supervised attention module, and the condition "valid" denotes not being previously selected or masked. As aforementioned, the query vector q_t is the hidden vector h_t of the LSTM encoder network. Basically, Equation 5 aims to find the weight matrices that jointly quantify the importance of all endpoints EP in the design. With a higher attention score $A_t^{(i)}$, an endpoint i will have a higher probability of being chosen at the current time step t.

To compute the probability $P_t^{(i)}$ of each endpoint i being selected at time step t, we use softmax to transform attention scores into probabilities as:

$$P_t^{(i)} = softmax(A_t^{(i)}) = \frac{e^{A_t^{(i)}}}{\sum_k A_t^{(k)}}, \forall i \in EP. \tag{6}$$

Note that for the endpoints that are not valid in current iteration, their probabilities of being selected will be zero as they all have an attention score equal to $-\infty$ from Equation 5. Finally, based on the distribution $P_t \in R^{|EP|}$, at each iteration t, we perform sampling to select one endpoint for useful skew prioritization. Note that the entire attention-based action decoding process is preformed in a self-supervised manner. That is, we are not using any pre-defined label or guidance as many other supervised frameworks. Hence, RL-CCD is generalizable to any design or technology as the entire selection process is purely based on design characteristics.

Algorithm 1 RL-CCD training methodology. We use $\rho = 0.3$ as default.

Input: Netlist $G = (V,E)$, Initial EP-GNN parameters θ_{gnn}, Initial LSTM encoder parameters θ_{LSTM}, Initial attention-based decoder parameters θ_{attn}, Overlapping threshold ρ, Violating endpoints EP
Output: RL-CCD parameters $\{\theta_{gnn}, \theta_{LSTM}, \theta_{attn}\}$

1: $t \leftarrow 0$ ▷ RL time step
2: Randomly initialize all training parameters $\{\theta_{gnn}, \theta_{LSTM}, \theta_{attn}\}$
3: $h_0 \leftarrow \mathbf{0}$, $F_{a_0} \leftarrow \mathbf{0}$ ▷ initialize LSTM inputs with zero vectors
4: $selected\_endpoints \leftarrow \{\}$
5: **while** not all violating endpoints are masked or selected **do**
6: $\{F_{EP}\} \leftarrow$ EP-GNN_encoding$(G, EP | \theta_{gnn})$ ▷ Equations 2, 3
7: $h_t \leftarrow LSTM\left(F_{a_{t-1}}, h_{t-1} \middle| \theta_{LSTM}\right)$ ▷ a_{t-1} is prior chosen ep
8: $q_t \leftarrow h_t$ ▷ take LSTM hidden vector as attention query vector
9: $P_t \leftarrow Attention(F_{EP}, q_t | \theta_{attn})$ ▷ Equations 5, 6
10: $a_t \leftarrow$ sample one endpoint from P_t ▷ selected ep
11: $G, EP \leftarrow$ overlap_masking(G, EP, a_t, ρ) ▷ fan-in cone
12: $selected\_endpoints \leftarrow$ add a_t to selection set
13: $t \leftarrow t+1$ ▷ total time steps will vary by design
14: Use margin to worsen timing of all selected endpoints to WNS
15: Run clock-path optimization using useful skew
16: Remove all added margins in Line 12, continue remaining place opt.
17: $R \leftarrow$ final TNS after completing entire placement optimization
18: REINFORCE update $\nabla_{\theta_\pi} \sum_t R \cdot \log \pi(a_t | \{\theta_{gnn}, \theta_{LSTM}, \theta_{attn}\})$
19: **Repeat** from Line 2 until TNS is optimized

C. Fan-in Cone Overlap Masking and the Rationale Behind

As shown in Figure 4, RL-CCD selects endpoints sequentially and the selection process completes when all endpoints in the design are either masked or selected. Our masking strategy is as follows: at each time step t, we mask out the endpoints whose fan-in cones have an overlapping ratio higher than a pre-defined threshold ρ with the selected endpoint a_t. The ratio calculation is described in Figure 3. The rationale behind is two-fold: (1) from design knowledge, successive endpoints are better not to be prioritized at the same time, otherwise it may cause ping-pong effect on clock arrival adjustments [7], and (2) for different designs, our strategy allows the RL agent to decide the total number of endpoints to select by either aggressively selecting highly-overlapped endpoints to mask out the rest faster or vice versa based on design characteristics.

D. Training Methodology

In this paper, we leverage REINFORCE [12], a renowned policy gradient algorithm, to train our RL-CCD framework. The objective of our RL agent is defined in Equation 1, which is to maximize the expected return $J(\pi_\theta)$ of each trajectory τ, where π represents the entire RL-CCD framework and θ denotes all parameters involved. To maximize the objective J, we perform gradient descent on the parameters of the entire framework $\{\theta_{gnn},\theta_{LSTM},\theta_{attn}\}$, and it is shown in [12] that the gradient of the objective J can be derived as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} R(\tau)\nabla_\theta \log \pi_\theta(a_t|s_t)\right], \tag{7}$$

where in our settings, $R(\tau)$ is the achieved TNS value after completing the entire placement optimization. Equation 7 denotes that the gradient of the objective is equivalent to the expected sum of the gradients of the log probabilities of the taken actions, weighted by the achieved reward at the end of the trajectory.

Algorithm 1 illustrates the end-to-end training process of our RL-CCD framework. We first initialize the inputs of the LSTM encoder to zero vectors in Line 3. Then, in Lines 5–13, we sequentially determine an action a_t, which denotes the endpoint to be selected at each RL time step t. Note that an overlap masking is performed in Line 11 to mask out the endpoints whose fan-in cones have an overlapping ratio greater than $\rho = 0.3$ with the selected endpoint. When the selection process completes, in Line 14, we worsen the timing of all selected endpoints to design WNS before entering the useful skew optimization (Line 15). These added margins are removed after the clock-path optimization (Line 16). Finally, we run through the remaining placement optimization steps and obtain the final achieved TNS value in Line 17, which is taken as the *RL reward*. With the reward, all parameters are jointly updated in Line 18, and the whole process repeats until the reward (TNS) is optimized.

IV. Experimental Results

In the experiments, we validate RL-CCD on **19** commercial designs (renamed due to confidentiality) in advanced technologies $5-12nm$. RL-CCD is integrated with an industry-leading commercial PD tool (name will be disclosed upon acceptance). The goal of RL-CCD is to find an optimal balance between clock-path and data-path (i.e., CCD) optimization through endpoint prioritization, so as to optimize design timing in terms of TNS. To perform fair and apple-to-apple comparison as shown in Figure 1, we use the same seed in each run to completely remove non-deterministic run-to-run variation. Also, RL-CCD does not leverage any additional optimization step other than the original ones used in the default tool flow (i.e., exact same recipe is used for both RL-CCD and the tool's native implementation). Finally, the runtime of both RL-CCD and the commercial tool is measured on the same farm machine without GPU support. RL-CCD is implemented using Python and TCL (no internal C++ code needed). Below, we clearly demonstrate that RL-CCD significantly improves the optimization quality of the reference commercial tool.

A. Single-Design Optimization Results

Table II demonstrates the optimization results achieved by training Algorithm 1 from scratch. Both reference tool and RL-CCD take the same global placements as inputs, where their attributes are reported in the left-most column. In the middle and right-most columns, it is shown that RL-CCD consistently outperforms the default tool flow (without endpoint prioritization) across all benchmarks, where we observe significant timing improvements in TNS by up to 64.4% (with an average improvement of 24%), and in NVE by up to 66.5% (with an average improvement of 19.4%). Figure 5 further demonstrates the prioritization impact of RL-CCD in terms of clock arrival adjustments on clock pins. It is shown that by intelligently prioritizing 74 critical endpoints out of the entire design (block11 with 180K cells), RL-CCD is able to efficiently affect the behaviour of the underlying useful skew engine to perform better optimization.

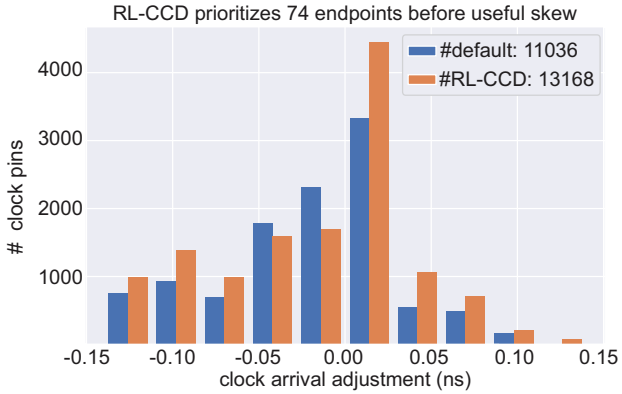

Fig. 5: Histogram of clock arrival adjustments on block11 (180K cells). Each pair of juxtaposed color bars has the same range of arrival values.

We believe the significant timing improvements achieved are not coming from the sacrifice of power, because RL-CCD is not degrading the power quality in general as reported by the sophisticated reference tool. In fact, although power is not an explicit objective, RL-CCD still achieves an average of 0.2% improvement via intelligent endpoint prioritization. Nonetheless, as different skewing solutions may impact downstream clock networks, we agree that the most accurate approach to justify power impact is to run through the entire PD flow, which, however, would easily take weeks to accomplish with our commercial benchmarks. Hence, in this paper, we specifically focus on improving the CCD optimization quality at the placement stage to demonstrate the effectiveness of the proposed RL framework.

Finally, the training of RL-CCD is achieved using multi-processing on CPU-only farm machines. Particularly, for each design, we launch 8 parallel processes to train the framework parameters. The training is terminated when the TNS value no longer improves in 3 consecutive iterations. For both reference tool and RL-CCD, we enable 16 threads to perform the entire placement optimization, including CCD and other optimization techniques. We understand that the runtime of RL-CCD may be prohibitive for other industrial designs in the real-world. Hence, we leverage transfer learning to further improve it as follows.

B. Transfer Learning on Unseen Designs

The key idea of transfer learning is to reuse pre-trained parameters in unseen domains, so that a framework that is trained upon certain samples can reach faster convergence in the unseen ones. In this paper, RL-CCD facilitates transfer learning by reusing the proposed EP-GNN model which is responsible for generating endpoint embeddings. Particularly, we first use the same EP-GNN model to perform RL training on different designs in the same technology (note that the encoder-decoder frameworks are distinct as the number of available endpoints varies by design). Then, after the training is completed, we load the weights and biases of the pre-trained EP-GNN model (with a new encoder-decoder framework) to perform RL training (Algorithm 1) on "unseen" designs. Figure 6 shows the transfer learning results on block19 (922K cells). It is shown that with transfer learning, RL-CCD can quickly converge to comparable optimization

TABLE II: Optimization results comparison between RL-CCD and the native implementation of an industry-leading commercial tool. RL-CCD is trained to minimize design TNS by selectively prioritizing critical endpoints. The unit for timing is ns and for power is mW. Runtime is normalized by default tool flow. Note that we use the same seed across all experiments to completely remove non-deterministic run-to-run variation.

design (# cells)	begin (post global place)				default tool flow (16 threads)					RL-CCD enhanced (ours)				
	WNS	TNS	#vio. EPs	total power	WNS	TNS (goal)	#vio. EPs	total power	run-time	WNS	TNS (goal)	#vio. EPs	total power	run-time
block1 (577K)	-0.24	-2009.98	33785	482.92	-0.16	-97.2	4296	1114.33	1.00	-0.16	-84.0 (-14.1%)	3603	1116.48	16
block2 (1.3M)	-0.18	-1104.03	40091	761.41	-0.05	-2.93	540	764.13	1.00	-0.07	-2.56 (-12.6%)	443	763.98	36
block3 (353K)	-0.26	-2966.04	36265	468.06	-0.17	-149.28	4119	474.72	1.00	-0.18	-87.45 (-41.42%)	1942	473.80	29
block4 (370K)	-0.46	-4590.85	38943	297.19	-0.11	-20.78	1258	322.48	1.00	-0.12	-7.40 (-64.4%)	421	321.97	31
block5 (194K)	-0.27	-1165.33	9708	199.45	-0.14	-162.45	4271	205.50	1.00	-0.14	-59.99 (-63.1%)	2081	204.95	39
block6 (195K)	-0.30	-1382.51	8704	102.03	-0.16	-69.90	1424	120.03	1.00	-0.16	-50.31 (-28.03%)	1146	119.50	20
block7 (416K)	-0.34	-2108.89	14086	121.56	-0.15	-41.47	1149	134.25	1.00	-0.16	-39.98 (-3.6%)	1009	134.35	21
block8 (135K)	-0.15	-1186.14	21272	348.10	-0.10	-72.18	2796	349.427	1.00	-0.10	-61.32 (-15.0%)	2314	349.56	42
block9 (162K)	-0.11	-50.90	1784	113.35	-0.02	-0.28	75	114.61	1.00	-0.01	-0.11 (-60.7%)	44	114.55	8
block10 (84K)	-0.43	-4428.41	29951	90.60	-0.26	-205.47	3669	90.70	1.00	-0.25	-189.92 (-7.6%)	3603	90.69	45
block11 (180K)	-0.29	-793.53	10658	266.72	-0.12	-5.67	149	276.96	1.00	-0.09	-4.04 (-28.8%)	135	276.79	32
block12 (243K)	-0.32	-1720.92	18465	78.72	-0.19	-102.90	2223	27.83	1.00	-0.18	-79.9 (-22.4%)	1794	27.83	46
block13 (507K)	-0.12	-375.08	12987	63.48	-0.06	-39.37	3779	64.95	1.00	-0.06	-33.72 (-14.4%)	3291	64.80	10
block14 (816K)	-0.16	-1913.75	44044	333.60	-0.06	-51.43	4260	340.07	1.00	-0.06	-48.89 (-4.9%)	3915	340.00	7
block15 (821K)	-0.18	-331.51	11002	66.17	-0.11	-40.55	2116	66.72	1.00	-0.11	-37.78 (-6.83%)	1861	66.71	20
block16 (432K)	-0.18	-374.15	9228	27.18	-0.07	-32.24	2586	28.09	1.00	-0.05	-24.89 (-22.8%)	2149	28.09	16
block17 (507K)	-0.14	-226.09	8860	407.69	-0.07	-46.22	2472	412.26	1.00	-0.06	-33.05 (-28.5%)	2361	412.21	35
block18 (412K)	-0.41	-2787.22	51675	583.88	-0.10	-6.14	123	1183.46	1.00	-0.10	-5.81 (-5.4%)	124	1182.23	26
block19 (922K)	-0.16	-383.69	8009	98.66	-0.09	-19.01	667	218.38	1.00	-0.06	-13.71 (-27.9%)	626	218.33	47
											avg. -24%	avg. -19%	avg. -0.2%	

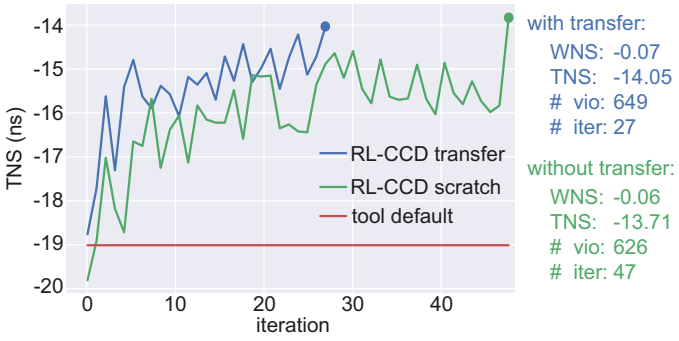

Fig. 6: Transfer learning on block19 (922K cells) by using a pre-trained EP-GNN model, where comparable optimization results is achieved in a much faster convergence rate.

results compared with training the entire framework (i.e., EP-GNN + encoder-decoder) from scratch as in Table II. The key rationale behind our transfer learning approach is that GNN netlist encoding should be universal (at least in the same technology). Hence, starting from more accurate embeddings, RL-CCD should be able to reach optimized solutions in faster convergence, which is proved in Figure 6.

C. Discussion: Why Does RL-CCD Work?

The development of RL-CCD is strongly motivated by the fact that commercial tools are ignoring endpoint sensitivity "across different optimization strategies". They adopt the same sequence of optimization steps to fix timing, however, they fail to make use of the fact that different endpoints react to various strategies distinctly (e.g., some are easier fixed from clock-path, while others, datapath). This is the key information that RL-CCD is learning, which eventually brings tremendous success. Finally, we attribute part of our success to the proposed fan-in cone overlap masking technique, which efficiently prunes out the action space, while allowing the RL agent to decide the total number of endpoints to pick subject to design characteristics.

V. Conclusion and Future Work

In this paper, we discover a new problem of balancing clock-path and data-path optimization in commercial tool flows, and solve it by using endpoint prioritization with RL. The proposed framework, RL-CCD, significantly improves the timing optimization quality of an industry-leading commercial tool across 19 commercial benchmarks in advanced technologies $5-12nm$. This work shall demonstrate the importance of the observed problem, and the strength of using RL algorithms to solve it. In the future, we aim to expand RL-CCD for full-flow optimization, so as to improve the overlap masking technique and quantify its impact on the achieved PPA values.

References

[1] T.-B. Chan, A. B. Kahng, and J. Li. Nolo: A no-loop, predictive useful skew methodology for improved timing in ic implementation. In Fifteenth International Symposium on Quality Electronic Design, pages 504–509. IEEE, 2014.

[2] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In 2022 59th ACM/IEEE Design Automation Conference (DAC).

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.

[4] Y.-C. Lu, , and S. K. Lim. On advancing physical design using graph neural networks. In IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2022.

[5] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim. Rl-sizer: Vlsi gate sizing for timing optimization using deep reinforcement learning. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 733–738. IEEE, 2021.

[6] A. Mirhoseini, A. Goldie, M. Yazgan, J. W. Jiang, E. Songhori, S. Wang, Y.-J. Lee, E. Johnson, O. Pathak, A. Nazi, et al. A graph placement methodology for fast chip design. Nature, 594(7862):207–212, 2021.

[7] S. Nath and V. Khandelwal. Machine learning-enabled high-frequency low-power digital design implementation at advanced process nodes. In Proceedings of the 2021 International Symposium on Physical Design.

[8] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren. Transsizer: A novel transformer-based fast gate sizer. In IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2022.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. Advances in neural information processing systems, 30, 2017.

[10] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. Advances in neural information processing systems, 28, 2015.

[11] K. Wang, L. Duan, and X. Cheng. Extensiveslackbalance: an approach to make front-end tools aware of clock skew scheduling. In 2006 43rd ACM/IEEE Design Automation Conference (DAC).

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