MACHINE LEARNING IN PHYSICAL DESIGN FOR 2D AND 3D INTEGRATED CIRCUITS

A Dissertation
Presented to
The Academic Faculty

By

Yi-Chen Lu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2023

MACHINE LEARNING IN PHYSICAL DESIGN FOR 2D AND 3D INTEGRATED CIRCUITS

Thesis committee:

Dr. Sung Kyu Lim
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yingyan (Celine) Lin
College of Computing
Georgia Institute of Technology

Dr. Saibal Mukhopadhyay
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Siddhartha Nath
Research and Development
Intel Corporation

Dr. Shimeng Yu
Electrical and Computer Engineering
Georgia Institute of Technology

Date approved: February 24, 2023

It is not intellect that tells so much as character; not brains so much as heart; not genius so much as self-control, patience, and discipline, regulated by judgement.

Ernest Hemingway

To my parents Leng-Chien Tsao and Ching-Chung Lu,

my brother Tang-Chen Lu,

my wife Jae Youn Kim, and my daughter Hannah Lu.

ACKNOWLEDGMENTS

First and foremost, I would like to show my sincere appreciation to my advisor, Dr. Sung Kyu Lim, for his invaluable guidance, support, and encouragement throughout my Ph.D. journey. His unwavering work attitude has inspired me to devote my heart into research, and his liberal approach in pursing knowledge has given wings to my creativity, turning imagination into practice. I would also like to express my heartfelt gratitude to my proposal and dissertation committee members – Dr. Madhavan Swaminathan and Dr. Saibal Mukhopadhyay for their great advice during the proposal exam, and Dr. Shimeng Yu, Dr. Yingyan (Celine) Lin, and Dr. Siddhartha Nath for spending their valuable time and effort in serving in my dissertation committee.

I am deeply grateful to the guidance from numerous industrial members during my internships in Ph.D. study: Kambiz Samadi, Sorin Dobre in Qualcomm; Siddhartha Nath, Vishal Khandelwal, Wei-Ting (Jonas) Chan, Deyuan Guo, Sudipto Kundo in Synopsys; Tian Yang, Haoxing Ren in Nvidia. They provide me the technical resources and insightful discussions that are significantly helpful to my research. Also, my genuine thanks to my past and current colleagues at GTCAD: Dr. Bon Woong Ku, Dr. Kyungwook Chang, Dr. Sai Pentapati, Dr. Anthony Agnesina, Dr. Jinwoo Kim, Da Eun Shim, Jeehyun Lee, Lingjun Zhu, Gauthaman Murali, Chengjia Shao, Pruek Vanna-Iampikul, Hao-Hsiang (Thomas) Hsiao, Yen-Hsiang (Robert) Huang, Dongwon Jang, Ray Anderson, Sandra Shaji, Nesara Bethur, Aditya Iyer, for their great help and exciting collaboration.

I want to show my deepest gratitude to my parents Leng-Chien Tsao and Ching-Chung Lu, and my brother Tang-Chen Lu. Their unconditional love and support is the unshakable foundation of my ability to pursue my dream. Without them, I wouldn't have gone this far.

Lastly, I am particularly indebted to my wife Jae Youn Kim, and my daughter Hannah Lu, for their tremendous love and forever encouragement that give me the greatest strength to conquer any challenge that may arise in the road ahead.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Physical design (PD) automation has made significant progress in enhancing chip design productivity over the past three decades. However, with the relentless growth in design complexity driven by Moore's Law, traditional PD algorithms are now facing increasing challenges in meeting desired power, performance, and area (PPA) targets within a reasonable amount of runtime in advanced technologies. Fortunately, the advancement of machine learning (ML) theory and its applications offer a new solution to this rising challenge, where recent studies of ML-assisted PD have demonstrated great potentials in revolutionizing conventional PD processes from synthesis to signoff.

The objective of this research is to develop ML algorithms that improve the final outcomes and productivity of PD implementations for 2D and 3D integrated circuits (ICs). In particular, various supervised, unsupervised, and reinforcement learning (RL) algorithms are devised to tackle a broad spectrum of traditional PD problems, which are categorized into four major themes in this dissertation. In the first theme, unsupervised learning algorithms are developed to perform tier partitioning in monolithic 3D (M3D) ICs, and clustering-based placement optimization. In the second theme, generative adversarial learning algorithms are devised to improve global placement of open-source placers, and optimize CTS outcomes of commercial tools. In the third theme, RL frameworks are constructed to perform gate sizing for timing optimization, and drive concurrent clock and data (CCD) optimization via intelligent endpoint prioritization. In the fourth theme, supervised learning models are presented to predict threshold voltage (V_{th}) assignment for leakage power optimization, and full-flow doomed run prediction.

CHAPTER 1
INTRODUCTION

Modern physical design (PD) flows heavily rely on electronic design automation (EDA) tools to generate graph design system (GDS) layouts from synthesized netlists that can be manufactured by foundries. However, with the relentless progression of design complexity and technology scaling driven by Moore's Law, existing EDA tools are facing difficulties in achieving desired power, performance, and area (PPA) targets due to the reliance on non-generalizable heuristics. Particularly, many heuristic algorithms that were once effective are now struggling to deliver satisfactory quality-of-results (QoR) in a reasonable amount of runtime in advanced technologies. As the transistor counts of modern processors soar beyond the billion mark, the PPA gap between what is "realizable" and what "should-be-achieved-by-scaling" becomes ever wider, necessitating the need to search for more reliable and generalizable algorithms, both "within" and "around" the tools to boost chip design productivity and final outcomes [1].

1.1 Tackling Physical Design (PD) with Machine Learning (ML)

Given that a PD implementation involves solving various NP-hard problems such as partitioning, placement, routing, and gate-sizing, for which optimal solutions in polynomial-time remain unknown [2], machine learning (ML) has emerged as a promising approach to further advance the well-matured PD flow in the semiconductor industry. This is primarily due to ML's capability to attain a faster convergence in quality-of-results (QoR) through efficient design space exploration (DSE), surpassing the limitations of conventional heuristic methods [3]. In particular, ML algorithms make use of design and technology features to perform early prediction or optimization for effective DSE, whereas traditional methods simply rely on the same heuristics to find candidate solutions across various designs, dimin-

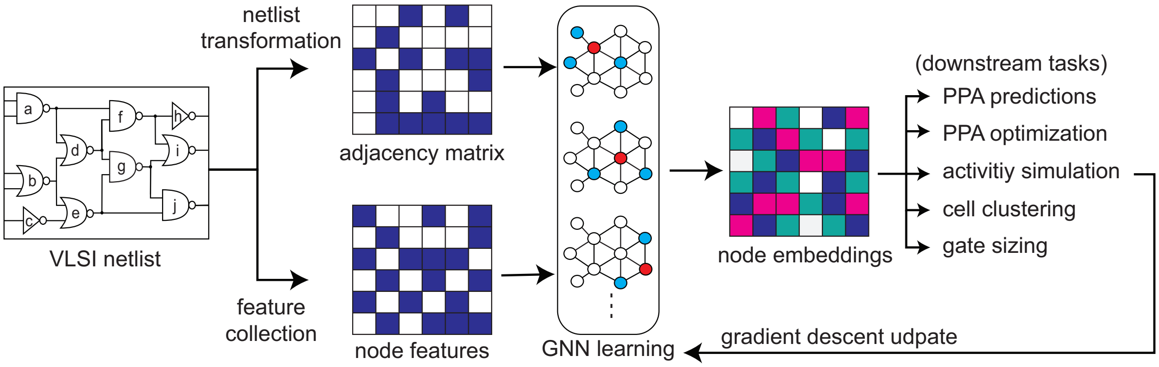


Figure 1.1: A common GNN modeling architecture in PD.

ishing the benefits of technology scaling. In this dissertation, we explore the opportunities of advancing modern PD flows using ML, where we devise supervised, unsupervised, and reinforcement learning (RL) algorithms to revisit and solve a wide range of classical PD problems in 2D and 3D ICs, including partitioning, placement, clock tree synthesis (CTS), gate sizing, prediction and optimization of PPA metrics.

1.1.1 Graph Neural Networks for Netlist Encoding

To make accurate predictions or to discover hidden netlist characteristics that can drive better PD optimization, ML algorithms deeply rely on the input vectors that are representative of underlying designs and technologies. Given the fact that very-large-scale integration (VLSI) netlists are essentially hypergraphs, where cells can be naturally considered as nodes and nets can be viewed as edges, Graph Neural Networks (GNNs) have become one of the most promising choices to encode netlist characteristics in an efficient and systematic manner [4]. Generally speaking, GNNs follow a message passing scheme, where the goal is to transform the initial features of each node into better representations by aggregating the features from neighboring nodes. A feature vector of a node can be considered as a message which is iteratively transformed and passed onto its neighboring nodes. At the end of the GNN learning process, the learned node embeddings (i.e., the transformed features) can be utilized as the inputs of downstream tasks such as link prediction, node-level or graph-level classification and clustering [5].

In the realm of PD, node embeddings learned by GNNs have been used to solve a wide range of tasks, including partitioning [6], placement [7, 8, 9, 10], gate sizing [11, 12, 13], activity simulation [14, 15], and PPA predictions [16, 17, 18, 19]. Figure 1.1 demonstrates a common GNN modeling architecture of leveraging GNNs to solve conventional PD problems. First, given an input netlist graph, an adjacency matrix denoting connectivity among cells will be constructed through netlist transformation techniques, and node features that represent design and technology characteristics will be collected. Then, node representation learning conducted by GNNs is performed to transform the initial node-level features into better representations, which can be taken as the inputs of various downstream tasks, forming an end-to-end differentiable framework. In latter chapters, we demonstrate more applications of GNNs in PD, where we show that by properly defining loss functions and reward structures, GNNs can be incorporated with self-supervised and reinforcement learning (RL) frameworks, respectively, to directly optimize key design metrics.

1.1.2 ML for Quality-of-Results (QoR) Prediction

One of the most crucial and straightforward use of ML algorithms in PD is the accurate prediction of critical QoR metrics based on pre-defined tool configurations (e.g., QoR strategy, tool parameters). This is often achieved by harnessing a large amount of historical data to train parameterizable models, which are usually powered by deep neural networks or tree-based algorithms. Ideally, these supervised models, once being trained with sufficient data, will have the ability to perform accurate inferences on new designs or technologies, thus enabling quicker and more informed decision making in new implementations, leading to an immediate improvement in chip design productivity in terms of runtime and cost (e.g., computing resources, tool licenses). For instance, a PD doomed run predictor [17, 18] can halt runs that are unlikely to meet pre-specified PPA targets in early design stages to reserve computing resources for the more promising ones.

1.1.3 From QoR Prediction to Optimization

Although prediction techniques achieved by supervised models provide instant productivity boost, these models often require a large amount of pre-generated training labels (i.e., ground-truths), making their applications prohibitive in the realm of PD as the generation of each label is extremely time-consuming. Furthermore, predictive models can never fundamentally solve the pressing issue that existing heuristic algorithms in EDA tools can no longer meet satisfactory QoR in advanced technologies such as $7nm$ and beyond. While supervised models can provide "tool-accurate" predictions, achieving "better-than-tool" results remains elusive. Hence, to truly reap the benefits of technology scaling, more powerful QoR optimization techniques are needed to bridge the PPA gap between what is available and what can be achieved, even at the cost of runtime.

Fortunately, ML offers a wide spectrum of optimization techniques that have demonstrated their abilities to reach never-seen, super-human results across various domains [20]. For example, generative adversarial networks (GANs) [21] and diffusion models [22] generate hyper-realistic images purely from random noises, earning recognition in art competitions; RL algorithms [23] have triumphed the best-in-class GO players in the world, making moves that challenge conventional thinking of human. These success stories motivate us to explore the powerful applications of ML in PD for QoR optimization. In this work, we show that although using a fundamentally different approach from commercial tools, "better-than-tool" optimization results can indeed be achieved by ML, especially with RL and self-supervised learning.

1.2 Contribution and Organization

The main contributions of this dissertation encompass four different themes across 8 chapters. The first theme corresponds to the development of *unsupervised learning* algorithms for tier partitioning in monolithic 3D (M3D) ICs, and clustering-based VLSI placement optimization using GNNs. The second theme corresponds to the design of *generative adversarial learning* frameworks for the improvements of DREAMPlace [24], a renowned open-source placer, and the clock tree synthesis (CTS) process in a commercial tool flow. The third theme corresponds to the construction of *RL* algorithms to solve gate sizing for timing optimization, and concurrent clock and data (CCD) optimization via endpoint prioritization. Finally, the last theme corresponds to the demonstration of *supervised learning* methods for the classification of threshold voltage (V_{th}) assignment in signoff power optimization, and the prediction of PD doomed runs.

Each part of the research is organized into a self-contained chapter as follows:

- In chapter 2, we present an unsupervised GNN-based framework named TP-GNN for tier partitioning in M3D ICs, which is motivated by the severe drawbacks of the state-of-the-art bin-based min-cut algorithm that introduce severe PPA degradation. Instead of simply relying on cutsize to determine the tier assignment of each cell, TP-GNN leverages design and technology features to solve the partitioning problem by finding the assignments that minimize an unsupervised loss. Experimental results on industrial designs demonstrate that TP-GNN significantly improves the QoR of the state-of-the-art 3D implementation flows. Specifically, in OpenPiton, a RISC-V-based multi-core system, we observe 27.4%, 7.7% and 20.3% improvements in performance, wirelength, and energy-per-cycle, respectively.

- In chapter 3, we present the first PPA-directed, unsupervised, end-to-end placement optimization framework that provides cell clustering constraints as placement guidance to advance commercial placers. Specifically, we directly formulate traditional PPA metrics as ML loss functions, and use graph clustering techniques to optimize them by improving cell clustering assignments. Experimental results on commercial GPU/CPU blocks under a commercial $5nm$ technology node and OpenCore benchmarks under a foundry $28nm$ technology node demonstrate that our framework improves the default design flow by up to 88% in post-route total negative slack (TNS) and delivers consistent improvements in wirelength and power.

- In chapter 4, we present DREAM-GAN, the first-ever placement optimization framework that transfers the placement quality of an industry-leading commercial placer, Synopsys ICC2, to a renowned open-source placer, DREAMPlace, using generative adversarial learning. Without knowing the algorithms used by the tool, DREAM-GAN facilitates transfer learning and directly improves DREAMPlace by optimizing a differentiable loss that denotes the "similarity" between DREAMPlace-generated placements and those in commercial databases. Experimental results on 6 industrial designs not only show the our DREAM-GAN immediately improves the Power, Performance, and Area (PPA) metrics at the placement stage, but also demonstrate that these improvements last firmly to the post-route stage, where we observe improvements by up to 8.3% in wirelength, 7.4% in power, and 37.6% in TNS on a commercial CPU benchmark.

- In chapter 5, we propose a novel framework named GAN-CTS, which utilizes conditional GAN to predict and optimize CTS outcomes. Our framework is comprised of three sequential learning stages. To precisely characterize distinct designs, we leverage transfer learning to extract netlist features directly from placement images, and with the extracted features along with the CTS input parameters, we adopt and analyze different regression methods to predict the target CTS outcomes. Finally, with the regression model, generative adversarial learning is leveraged to optimize the target metrics. Experimental results on real-world designs demonstrate that our framework (1) achieves an average prediction error of 3%, (2) improves the commercial tool's auto-generated clock tree by 20.7% in clock power, 21.5% in clock wirelength, 36.1% in the worst skew, and (3) reaches an F1-score of 0.93 in the classification task of determining successful and failed CTS runs.

- In chapter 6, we present a novel framework named RL-Sizer to perform gate sizing for timing optimization the deep deterministic policy gradient (DDPG) [25] algorithm. Unlike conventional sizing algorithms inherent in the tool using various pseudo-heuristics that can not generalize globally, RL-Sizer performs the optimization in a global and systematic manner. Experimental results demonstrate that RL-Sizer outperforms the native sizing algorithm in an industry-leading commercial tool, Synopsys ICC2, in terms of TNS and number of violating endpoints (NVE) on 4 out of 6 commercial designs with negligible power overhead, while achieving parity on the others.

- In chapter 7, we present RL-CCD, an RL agent that performs CCD optimization through endpoint prioritization. RL-CCD is motivated by the fact that existing CCD algorithms in commercial tools fail to prioritize violating endpoints for different optimization strategies correctly, leading to flow-wise globally sub-optimal results. Particularly, they ignore the fact that different endpoints have distinct sensitivity to various optimization techniques, where they always use the same recipe of strategies to fix all violating endpoints. RL-CCD overcomes this issue by selecting endpoints for useful skew optimization 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% TNS reduction and 66.5% NVE improvement over the native implementation of an industry-leading commercial tool.

- In chapter 8, we present ECO-GNN, a transferable graph-learning-based framework, which harnesses GNNs to perform commercial-quality signoff power optimization through discrete V_{th}-assignment. The development of ECO-GNN is motivated by the fact that the signoff engineering change order (ECO) optimization is highly time-consuming in modern PD flows, and the power improvement is hard to predict in advance. ECO-GNN strives to alleviate this issue by generating tool-accurate optimization results *instantly* without going through the entire ECO process. Furthermore, subgraph approximation technique is proposed to improve training and inferencing time of ECO-GNN. We show that design instances with non-overlapping subgraphs can be optimized in parallel so as to improve the inference time of the learning-based model. Experimental results on 14 industrial designs, demonstrate that our framework achieves up to 14X runtime improvement with similar signoff power optimization quality compared with *Synopsys PrimeTime*, an industry-leading signoff tool.

- In chapter 9, we propose PD-LSTM, a framework that leverages graph neural networks (GNNs) and long short-term memory (LSTM) networks to perform end-of-flow power predictions sequentially from early PD stages. This work is motivated by the fact that leading-edge designs on advanced nodes are pushing PD flow runtime from days to weeks, and stringent time-to-market constraint necessitates efficient PPA exploration by developing accurate models to evaluate netlist quality in early design stages. Experimental results on two commercial CPU designs and five OpenCore netlists demonstrate that PD-LSTM achieves high-fidelity total power prediction results within 4% normalized root-mean-squared error (NRMSE) on unseen netlists and a correlation coefficient score as high as 0.98.
- Finally, in chapter 10, we revisit and present the conclusion of each chapter.

CHAPTER 2
A MACHINE LEARNING POWERED TIER PARTITIONING FRAMEWORK FOR MONOLITHIC 3D ICS

2.1 Background and Motivation

2.1.1 Monolithic 3D (M3D) ICs

The 3D integration technology offers a promising path to continue technology scaling beyond Moore's Law. It improves the Power, Performance, and Area (PPA) metrics of 2D Integrated Circuits (ICs) by stacking multiple dies one on top of another using inter-tier vias. Based on the die stacking method, 3D integrated circuits can be classified into three categories: Through-Silicon Via (TSV) based, Monolithic Inter-tier Via (MIV) based, and Face-to-Face (F2F) bonded [26]. Although TSV-based 3D ICs are developed the first, their low integration densities due to large pitches and high parasitics hinder them from fully realizing the benefits of 3D integration. In contrast, Monolithic 3D (M3D) Integration has emerged as the most promising approach among the three, with its nano-scale MIVs enabling more cost-effective inter-tier connections and finer physical design, resulting in a significantly higher device density [27, 28]. Hence, in this chapter, we concentrate on enhancing M3D implementation flows using Machine Learning (ML) [6, 29].

2.1.2 Limitations of Current Tier Partitioning Methods in M3D Flows

The greatest challenge in building high-quality 3D ICs is the placement process [30], as there are currently no commercial Electronic Design Automation (EDA) tools available that can perform 3D placement directly from 2D synthesized netlists. Hence, to address this challenge and produce commercial-grade 3D ICs using industry-leading EDA tools, state-of-the-art M3D implementation flows, such as Shrunk2D [31], Compact2D [32], and

Snap3D [33], utilize commercial 2D placers to simulate 3D placement by performing tier partitioning in ”projected 2D designs” rather than directly performing ”true 3D placement” from 2D netlists.

Tier partitioning refers to the practice of allocating each design instance (i.e., cell or macro) to a specific tier. This is a crucial step, as it determines the placement of standard cells and inter-tier vias (such as MIVs), which significantly affects the Quality of Results (QoR) of full-chip designs. State-of-the-art M3D flows [31, 32, 33] all adopt the bin-based min-cut algorithm proposed in [31] to perform tier partitioning, which partitions netlists by minimizing cutsize of pre-defined bins. Particularly, this algorithm first divides the 2D design into several rectangular regions, known as ”bins,” on the x-y plane. Afterwards, it uses an area-balanced min-cut partitioning algorithm to divide the cells within each bin into different tiers (i.e., along the z-direction) while minimizing the cutsize of the partial netlist. However, there are several significant drawbacks of this approach that lead to sub-optimal 3D full-chip designs, namely:

- Timing Degradation. The bin-based partitioning algorithm fails to consider the global connections among bins. It only iteratively partitions the sub-netlist within a single bin, which inevitably leads to a severe timing degradation.

- Low 3D Integration Density. Min-cut partition is not necessarily good for 3D integration as it might not realize the full potential of the high integration density that monolithic 3D (M3D) integration provides.

- Placement Quality Degradation. Hierarchy information from RTL is completely ignored in the existing bin-based algorithm. Therefore, extra cutsize will be introduced and inter-tier vias will be inserted in sub-optimal locations, which results in a placement quality degradation.

In this chapter, we address all the limitations outlined above. Our solution is TP-GNN, a novel unsupervised learning-driven framework that uses Graph Neural Networks (GNNs)

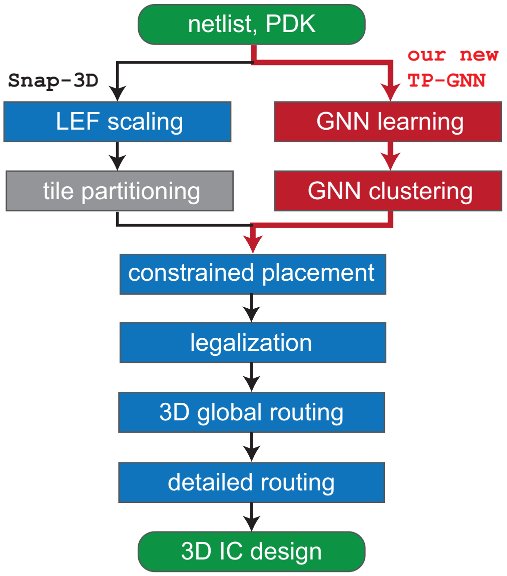


Figure 2.1: Partitioning-first M3D design flow.

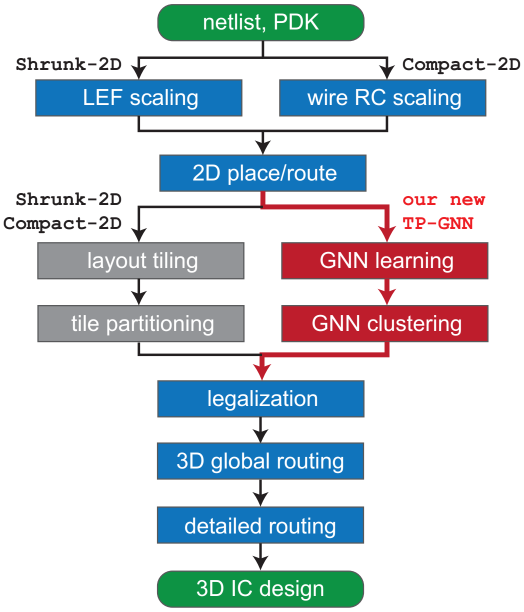


Figure 2.2: Partitioning-last M3D design flow.

and the weighted k-means clustering algorithm to perform tier partitioning. Unlike the previous bin-based min-cut method that disregards important design and technology parameters, our TP-GNN takes into account crucial information such as timing, hierarchy, and technology information of the underlying designs. Our goal is to offer an innovative tier partitioning framework that outperforms the current state-of-the-art M3D implementation flows in terms of full-chip PPA metrics.

In this chapter, we address all the drawbacks raised above. We present TP-GNN, an unsupervised graph-learning-based framework that performs tier partitioning using graph neural networks (GNNs) and the weighted k-means clustering algorithm [34]. Unlike previous works that neglect design-related and technology-related parameters, we consider timing, hierarchy, and library information in our algorithm. The goal of this work is to present a novel tier partitioning framework that advances the state-of-the-art M3D implementation flows in terms of the full-chip PPA metrics.

2.1.3 Different Styles of M3D Flows: Partitioning-First and Partitioning-Last

Active research in M3D ICs has been conducted extensively over the years, resulting in the creation of numerous M3D implementation methods. Given that all M3D approaches must separate logics onto different tiers during tier partitioning, which can occur either early (partitioning-first) or late (partitioning-last) with respect to the placement stage. As a result, M3D design flows can be divided into two categories: partitioning-first and partitioning-last, as shown in Figure 2.1 and Figure 2.2. Below, we delve into the specifics of the two types of M3D flows:

Partitioning-first M3D Flows:

The authors of [35] introduce the first partitioning-first M3D implementation flow, named Cascade-2D, which seeks to enhance memory-intensive commercial CPU designs through a 3D implementation. However, as Cascade-2D is limited in its scope, where the approach does not generalize to a wider range of design styles, in this work, we take a more recent flow, Snap-3D [33], as our baseline of partitioning-first M3D design flow. Snap-3D, which achieves state-of-the-art results on popular benchmarks, is depicted in Figure Figure 2.1. The central concept behind Snap-3D is the division of the row structure of a 2D placement into even and odd sites, representing two distinct 3D dies. By carefully specifying placement constraints in 2D commercial placers, standard cells can be placed in different dies in a simultaneous, co-optimization manner. The original Snap-3D flow employs tiling methods, akin to the min-cut partitioning used in Shrunk-2D and Compact-2D, to generate such constraints. In this work, we leverage the proposed tier partitioning framework to generate the constraints and demonstrate that our design-aware partitioning approach can lead to better PPA results.

Partitioning-last M3D Flows:

Shrunk-2D [31] and Compact-2D [32] are recognized as the leading partitioning-last M3D implementation flows. These flows employ commercial tools for physical design implementations and differ in the manner of emulating 3D designs during the 2D stage. In Shrunk-2D, standard cells are reduced in size by half for 2D placement and routing (P&R), while in Compact-2D, the RC parasitics are scaled by a factor of $1/\sqrt{2}$ without altering cell size. Following the 2D P&R, cells are expanded in Shrunk-2D or projected in Compact-2D onto a 2D die with half of the original footprint before tier partitioning, where as aforementioned, both flows adopt the bin-based partitioning method, which leads to a significant degradation in the quality of the final full-chip 3D design. The remaining stages after the 2D P&R stage for both flows are similar, starting from the legalization for both tiers to the timing closure for tape-out.

Note that independent of the design style, our TP-GNN framework can significantly advance all M3D flows in terms of the final full-chip PPA metrics by replacing the widely-adopted bin-based min-cut tier partitioning algorithm with our ML-powered approach as shown in Figure 2.1 and Figure 2.2. Moreover, we validate our framework's ability to handle heterogeneous 3D designs, where different technologies can be employed in each tier. This is demonstrated through the use of Pin3D [36], a heterogeneous M3D design flow based on a commercial multi-core CPU design. Our findings demonstrate that the TP-GNN framework has the capability to understand technology features and carry out effective tier partitioning.

Details on MIV planning (insertion):

For both partitioning-first and partitioning-last design flows, MIV insertions are performed after obtaining the partitioning results at the 3D global routing stage as shown in Figure 2.1 and Figure 2.2. Specifically, in this stage, we stack the metal layers from top die and bottom die together, and advise the router to perform global routing on the stacked metal layers, where MIVs are the vias that the router inserts between the top metal layer of the bottom die and the bottom metal layer of the top die. For example, assume a two-tier M3D design and each die has 6 metal layers (M1–M6), which results in 12 metal layers (M1–M12) when stacked together during the 3D global routing phase, the MIVs are the vias that the router inserts between the M6 and M7 layers. Note that the main purpose of MIVs is to connect the cells located in different dies. In the above example, MIVs are leveraged to connect the pins of the bottom die cells that are located in M1 with the pins of the top die cells located in M7.

2.1.4 Heterogeneous 3D ICs Design Flow

Heterogeneous 3D ICs refer to the 3D chips that adopt more than one technologies within. A common practice is to use different technologies for various dies (tiers). The main benefit is that by using heterogeneous 3D stacking, 3D ICs in old technologies can reap the performance gain of the 2D chips equipped with new technologies which are prohibitively expensive to be developed. In other words, heterogeneous 3D integration may produce competitive products as 2D technology scaling but at a lower cost. However, The design flows (Shrunk2D, Compact2D, Snap3D) introduced in the previous sub-section do not support building heterogeneous 3D designs. To validate the proposed tier partitioning framework in a broader scale, in this paper, we take Pin3D [36], a novel heterogeneous 3D design flow, as our reference flow, and demonstrate that the proposed partitioning strategy can achieve better full-chip PPA than the original partitioning strategy in Pin3D [36].

2.2 TP-GNN Algorithms

2.2.1 Overview

Figure 2.1 and Figure 2.2 demonstrate the integration of our proposed tier partitioning framework TP-GNN with the state-of-the-art 3D design flows. As shown in the figures, the input to the TP-GNN framework is a projected 2D design, where all the cells are placed,

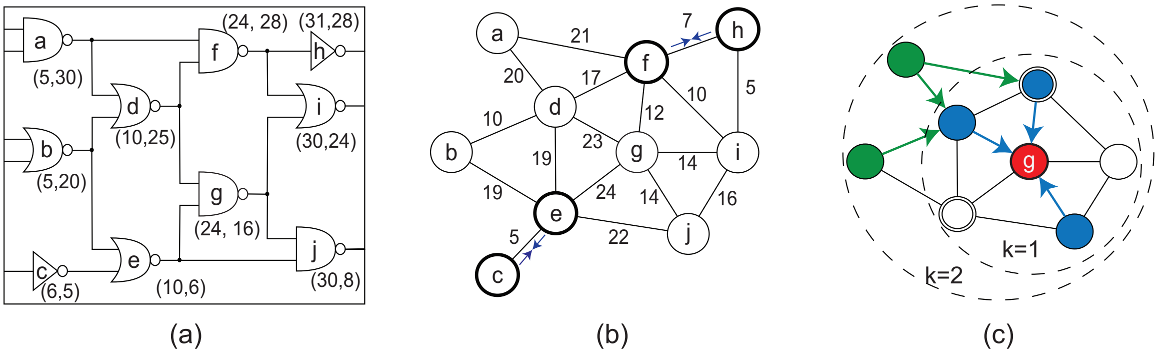


Figure 2.3: TP-GNN visualization. (a) Input netlist with two design hierarchies: $\{a, b, d, f, h\}$ and $\{c, e, g, i, j\}$. Numbers represent cell locations. (b) Hierarchy-aware edge contractions on the transformed clique-based graph. Edge weights represent the Manhattan distance. (c) For target node g, sampling and aggregating features from its k-hop neighbors.

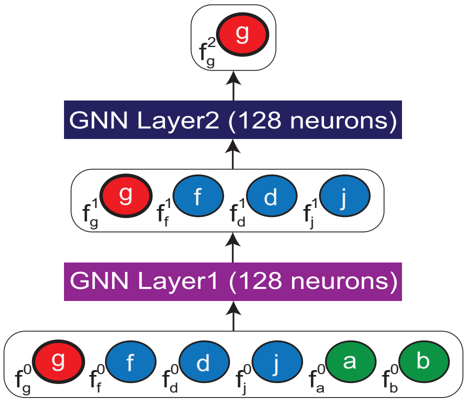


Figure 2.4: Graph learning for target node g. Following from Figure 8.1(c) we demonstrate the detailed learning process, where $\{f^0\}$ represent the initial features and f_g^2 represents the learned representations.

routed, and projected onto a 2D die with half of the 2D counterpart's footprint. The output of the framework is a partitioned design, where each cell is assigned to a unique tier.

Figure 8.1 shows the visualization of our framework. Given a projected 2D design as shown in Figure 8.1(a), we transform the netlist hypergraph into an edge-contracted clique-based graph as shown in Figure 8.1(b) by devising a hierarchy-aware edge contraction algorithm. After the contraction, we leverage GNNs to perform instance-based graph representation learning as shown in Figure 8.1(c), where features within K-hop neighbors ($K = 2$) of the target node are sampled and aggregated to learn accurate representations for the downstream clustering stage.

Algorithm 1 Hierarchy-aware edge contraction algorithm.

```
Input: $G(V, E)$: original clique-based graph.
Output: $G'(V', E')$: edge-contracted clique-based graph.
$E \leftarrow$ SortEdgesByWeight($E$) (in ascending order)
for $e = (u, v) \in E$ do
    if $u, v$ not contracted and $u, v$ in the same hierarchy then
        contract $(u, v)$ to form a new vertex $v'$                ▷ in-place
        $v'_x \leftarrow \frac{u_x+v_x}{2}$, $v'_y \leftarrow \frac{u_y+v_y}{2}$                ▷ update location
        for $n \in \{$neighbors($u$) $\cup$ neighbors($v$)$\}$ do
            edgeWeight($v'$, $n$) = $|v'_x - n_x| + |v'_y - n_y|$
$G'(V', E') \leftarrow G(V, E)$
```

Finally, our tier partitioning framework TP-GNN is generalizable to *every* design, since it learns the feature representations by optimizing an unsupervised loss function (unsupervised learning). Also, it does not assume anything regarding the netlist structure or design characteristics. Instead, it learns and adapts to various netlists using graph embedding techniques. TP-GNN can be easily integrated with existing 3D implementation flows to significantly improve the quality of the final full-chip design. Note that ideally, our method can be extended to support multi-tier partitioning by clustering the nodes into $k > 2$ groups. However, the transition will not be that smooth because it will depend on the ways that pseudo-placements are generated and MIVs are inserted into multi-tiers. Furthermore, currently state-of-the-art M3D design flows (Shrunk2D [31], Compact2D [32], Snap3D [33], and Pin3D [36]) only support two-tier M3D designs, in this work, we focus on improving the full-chip PPA metrics of two-tier 3D designs. The detailed algorithms of our framework are described in the following sub-sections.

2.2.2 Hierarchy-Aware Edge Contraction

Starting from a projected 2D design, we first transform the original netlist (a directed hypergraph) into an undirected clique-based graph G, where a net that originally contains k cells forms a k-clique in G, and each edge $e = (u, v)$ is assigned a weight representing the Manhattan distance between cell u and cell v in the projected 2D placement. Then, we apply a hierarchy-aware edge contraction algorithm (Algorithm 1) on this graph G, where

Table 2.1: Initial node features for partitioning last design flows in edge-contracted graph G'. Note that a node may represent multiple cells in the design.

features	**descriptions**
hierarchy	"module" defined in the synthesized netlist
sum_slack	sum of worst slacks of all cells
sum_slew	sum of maximum pin slew of all cells
sum_delay	sum of worst delay of all cells
dist2source	length of shortest path to clock source on G'
1-hop degree	number of 1-hop neighbors on G'
2-hop degree	number of 2-hop neighbors on G'

pairs of nodes within the same hierarchy are contracted into supernodes based on the ascending order of edge weights (lines 1-4). When a supernode v' is obtained, we update the edge weights between its neighbors and its center of gravity (lines 5-7). Note that the term "hierarchy" refers to the "module" defined in the synthesized netlist (RTL).

The goal of Algorithm 1 is to prevent the severe placement quality degradation occurred in Shrunk2D and Compact2D, which can be accounted by two reasons. First, cells within the same hierarchy are highly connected with each other. If the hierarchy information is ignored in the partitioning algorithm, inter-tier vias will be inserted in sub-optimal locations that introduce redundant cuts. Second, previous works fail to consider the actual cell distance in the 2D placement while performing partitioning. Cells that are nearby and connected should have a higher chance to remain in the same tier compared with other distant cells; otherwise, designs will suffer from severe 3D routing overhead. Finally, Algorithm 1 can be performed recursively to condense the graph and to benefit from the run time and memory requirement of the later graph learning. However, a denser graph does not always achieve better PPA. In the experiments, we perform 2 runs of Algorithm 1 for each design implemented by our framework.

2.2.3 GNN as Feature Aggregator

After obtaining the edge-contracted clique-based graph G' from Algorithm 1, we leverage GNN to perform graph learning. The goal of this stage is to learn accurate node representations that capture the characteristics of the design regarding tier partitioning. These learned representations are further utilized to determine the tier assignment in the later clustering stage.

Before the actual learning process, we determine an initial feature vector for each node as shown in Table 9.1. Note that features in Table 9.1 are designed for partitioning-last M3D flows where the tier partitioning stage happens after 2D physical implementation stage as shown in Figure 2.2. For partitioning-first design flow, the features are extracted right after the synthesis stage. Due to the lack of physical implementations (e.g. placement and routing), we drop the slack and slew features presented in Table 9.1 while remaining others.

The features in the table span from a node's structural information and its design attributes. Unlike previous works that ignore timing information during tier partitioning, we prevent the severe timing degradation by considering four timing related features as shown in Table 9.1. Note that these initial node representations are insufficient to perform tier partitioning. To learn better representations, we train GNNs to sample and aggregate the neighboring features for each node. The GNN model will capture the local structural information as well as the node attributes that are related to tier partitioning. Inspired by [37], our feature aggregator aggregates the k-hop neighborhood features of a node v as follows:

$$f_v^k = \sigma \left(f_v^{k-1} + \theta_k \cdot \frac{1}{s_k} \sum_{u \in SN_k(v)} f_u^{k-1} \right), \tag{2.1}$$

where σ is the sigmoid function, f_v^k denotes the representation vector of node v at level k, $SN_k(v)$ denotes the neighbors sampled at k-hop, s_k denotes the corresponding sampling size, and θ_k represents the parameters of the neural network (NN) at k-hop (each hop has its own NN). Note that the concept of "level" is corresponding to the concept of "hop", where f_v^0 is the initial features defined in Table 9.1 for node v, and $f_v^{k=K}$ is the final representation after aggregation the information within the K-hop neighborhood of v. The aggregator (Equation Equation 2.1) can be considered as a "graph filter", since it performs instance-based learning that aggregates a node's neighboring information iteratively. In the experiments, we set $K = 2$ and each neural network (θ_1, θ_2) has an output dimension of 128. Finally, Figure 2.4 further demonstrates the feature aggregation process based on Figure 8.1(c), where our goal is to construct the node representation for the target node g. The learning process happens as follows. First, we sample a fixed amount of neighbors from its 1-hop (denoted in blue) and 2-hop (denoted in green) neighbors. Then, starting from the initial features $\{f^0\}$, we leverage a two-layer GNN to perform iterative feature aggregation in order to construct the final representation f_g^2.

2.2.4 Unsupervised GNN Learning

In this work, we leverage unsupervised learning to train the TP-GNN framework. Therefore, our framework is generalizable, since it does not require any pre-training before using. Here, we introduce an unsupervised instance-based loss function $\mathcal{L}(y_v)$, which takes $y_v = f_v^K$, the final representation vector of node v, as the input and calculates the cross-entropy between v and its neighboring nodes $N(v)$ (not necessary in K-hop) as:

$$\begin{aligned} \mathcal{L}(y_v) = &- \sum_{u \in N(v)} \log(\sigma(y_v^\top y_u)) \\ &- \sum_{i=1}^{M} \mathbb{E}_{n_i \sim Neg(v)} \log(\sigma(-y_v^\top y_{n_i})), \end{aligned} \tag{2.2}$$

where $Neg(v)$ denotes the negative sampling distribution of node v, and M denotes the negative sampling size. In practice, rather than taking $N(v)$ as the full k-hop neighborhood of node v, which causes overfitting and damages computational efficiency, we perform a random walk starting from node v to generate $N(v)$ that represents the passed by nodes. Also, in Equation Equation 7.1, the negative sampling technique improves the efficiency of GNN learning, where an underlying idea is that the GNN model should not only improve the similarity between a node v and its true contexts $N(v)$, but also enhance the disparity of v to the false samples $Neg(v)$ (nodes that are not occurred in the random walk).

Algorithm 2 TP-GNN training methodology.
We use default values of $\alpha = 0.001, K = 2, NRW = 5, LRW = 5, M = 30, s_1 = 30, s_2 = 20, \beta_1 = 0.9, \beta_2 = 0.999$.

Input: $G'(V', E')$: edge-contracted clique-based graph. $\{f^0\}$: initial features. α: learning rate, K: depth of neighborhood, NRW: # random walks starting from a node, LRW: length of a walk, M: negative sampling size, $\{s_k, \forall k \in \{1, ..., K\}\}$: k-hop neighborhood sampling size, σ: sigmoid function, $\{\theta_k, \forall k \in \{1, ..., K\}\}$: parameters of NN at hop k, $\{\beta_1, \beta_2\}$: Adam parameters.

Output: $\{y\}$: learned node representations.

```
 1: for $v \in V'$ do                                          ▷ random walks on each node
 2:     $N(v) \leftarrow \{\}$                                  ▷ initialization of neighboring nodes
 3:     for $n \leftarrow 1$ to $NRW$ do
 4:         $cur\_v \leftarrow v$
 5:         for $l \leftarrow 1$ to $LRW$ do
 6:             $next\_v \leftarrow$ Sample a 1-hop neighbor of $cur\_v$
 7:             if $next\_v$ is not $v$ then
 8:                 add $next\_v$ to $N(v)$
 9:             $cur\_v \leftarrow next\_v$
10: while $\{\theta_k\}$ do not converge do                     ▷ train to converge
11:     $f_v^0 \leftarrow \frac{f_v^0}{\|f_v^0\|_2}, \forall v \in V'$
12:     for $k \leftarrow 1$ to $K$ do                          ▷ aggregate neighborhood features
13:         for $v \in V'$ do
14:             $S_k \leftarrow$ Sample $s_k$ neighbors at $k$-hop neighborhood
15:             $f_v^k = \sigma\left(f_v^{k-1} + \theta_k \cdot \frac{1}{s_k} \sum_{u \in S_k} f_u^{k-1}\right)$
16:         $f_v^k \leftarrow \frac{f_v^k}{\|f_v^k\|_2}, \forall v \in V'$
17:     $y_v \leftarrow f_v^K, \forall v \in V'$
18:     for $v \in V'$ do                                       ▷ minimize unsupervised loss
19:         for $u \in N(v)$ do
20:             $Neg(v) \leftarrow$ Sample $M$ samples from $\{V' - N(v)\} \setminus v$
21:             $neg\_loss \leftarrow \sum_{n_i \in Neg(v)} \log(\sigma(-y_v^\top y_{n_i}))$
22:             $g_v \leftarrow \nabla_\theta[\log(\sigma(y_v^\top y_u)) + neg\_loss]$
23:             $\{\theta_k\} \leftarrow Adam(\alpha, \{\theta_k\}, g_v, \beta_1, \beta_2)$
```

2.2.5 GNN Training Methodology

To update the parameters of our framework, we introduce a gradient descent optimizer Adam [38] to minimize $\mathcal{L}$ (Equation Equation 7.1). The detailed training methodology is described in Algorithm 12. In lines 1-9, we perform random walks on every node $v \in V'$ to generate the neighborhood structures. Then, starting from the initial features (Table 9.1),

Algorithm 3 Weighted k-means Clustering.
We use default value of $k = 2$.

```
Input: $G'(V', E')$: edge-contracted clique-based graph, $\{w\}$: node weights, $\{y\}$: node representations, $k$: number of clusters.
Output: $\{C_1, ..., C_k\}$: $k$ clusters.
Select $k$ initial centroids $\{c_1, ..., c_k\}$ randomly
repeat
$\{C_1, ..., C_k\} = \underset{C}{\text{argmin}} \sum_{i=1}^{k} \sum_{v \in C_i} w(v) \|y_v - c_i\|^2$
$c_i = \frac{\sum_{v \in C_i} y_v w(v)}{\sum_{v \in C_i} w(v)}, \forall i = 1, ..., k$
until $\{C_1, ..., C_k\}$ no longer change
```

we aggregate the neighborhood features for each node through Equation Equation 2.1 (lines 11-17). Finally, in lines 18-23, we leverage Adam to update the parameters of the GNNs through the introduced unsupervised loss function (Equation Equation 7.1). After the learning process, the learned node representations $\{y\} \in R^{128}$ are fed to the later clustering stage to determine the tier assignment for each cell.

2.2.6 Weighted K-means Clustering

The final stage of the proposed framework is the clustering process, where we leverage the weighted k-means clustering [39] to partition the edge-contracted clique-based graph $G' = (V', E')$. The goal at this stage is to determine the tier assignment for each node $v \in V'$ based on its learned representation y_v from Algorithm 12. In this work, we introduce a weight to each node $v \in V'$ which denotes the total area of the gates that it represents. Note that a node may represent multiple gates in the actual netlist, and gates corresponding to the same node will be assigned to the same tier. Given the learned node representations $\{y\}$ and the weights $\{w\}$, the algorithm clusters the nodes V' into k weight-balanced groups based on the similarity of $\{y\}$. Assume V' is classified into k clusters $\{C_1, ..., C_k\}$, the loss function is derived as

$$\mathcal{L}_{kmean} = \sum_{i=1}^{k} \sum_{v \in C_i} w(v) \cdot \|y_v - c_i\|^2, \tag{2.3}$$

Algorithm 4 Post-partitioning optimization.
We assume a two-tier 3D design and top die is faster.

```
Input: $G(V, E)$: original 2D design, $C_{top}$: instances in top tier, $C_{bot}$: instances in bottom tier.
Output: $C'_{top}$ and $C'_{bot}$: updated partitioning results.
$critCells \leftarrow$ FindCellsOnCriticalPaths($G$)            ▷ in hash map
for $net \in Nets$ do
    $critiCount \leftarrow 0$
    for $cell \in net$ do
        if $cell$ in $critCells$ then                              ▷ O(1) look-up
            $critiCount$++
    if $critiCount \geq 0.5 * countCell(net)$ then
        Fix all cells on $net$ in top tier.                        ▷ in-place
```

where $c_i = \frac{\sum_{v \in C_i} y_v w(v)}{\sum_{v \in C_i} w(v)}$ denotes the weighted centroid of cluster C_i. To update Equation Equation 2.3, we adopt an iterative minimization technique as illustrated in Algorithm 3. Starting from an initial centroids $\{c_1, ..., c_k\}$, for each iteration, we determine the clusters $\{C_1, ..., C_k\}$ by assigning each node to the centroid that has the minimum weighted distance (line 3). After the assignments, we update the centroids based on the newly obtained clusters (line 4). The clustering process is complete when the assignments no longer change.

2.2.7 Post-partitioning Optimization

The clustering results of the weighted K-means algorithm (Algorithm 3) can already be taken as valid tier partitioning solutions. However, for certain design flows such as the heterogeneous 3D design flow, extra handling on timing degradation during tier partitioning phase is needed. The reason is that such design flow leverages different technologies in various tiers, where the performance (timing) between BEOLs can vary by as much as 30% [40]. Therefore, the 3D designs can easily result in worse performance if cells on critical paths in the original 2D design are partitioned randomly as occurred in Shrunk-2D and Compact-2D.

To solve the above issue, in this work, we further propose a post-partitioning optimization algorithm to mitigate the performance degradation of tier partitioning in heterogeneous

3D design flows. The proposed algorithm is shown in Algorithm 4. Given a tier partitioning result that denotes the cell locations in top tier and bottom tier, we first build a hash map to identify the critical cells in the original 2D design (Line 1). Then, for each net in the design, if greater or equal to half of the cells are in the critical cell map, then we fix the entire cells on the net in top tier (Lines 2–8). Note that the algorithm is based on the assumption that top die is faster than bottom die (as in the Pin-3D [36] design flow).

2.2.8 Implementation Details

in this work, we apply the proposed TP-GNN framework to a variety styles of M3D design flows which include partitioning-first, partitioning-last, and heterogeneous 3D design flows. In the partitioning-first design flow, since the tier partitioning happened before the pseudo-placement, the timing related features in Table 9.1 are taken from a synthesis tool (*Synopsys Design Compiler*), where in the partitioning-last design flow, the features are found in *Cadence Innovus*. As for the feature “dist2source”, we take the hop-count as the representation in partitioning-first design flows, and take the actual physical distance on layout as the denotation in partitioning-last design flows. Finally, in the heterogeneous design flow Pin-3D, since it accepts a partitioned design as inputs and continues the design flow through 3D legalization to tape-out, the feature extraction process is same as the partitioning-last design flow.

2.3 Experimental Results

In this section, we perform thorough experiments to demonstrate the achievements of TP-GNN framework. We validate our framework on 7 industrial designs, including two RISC-V-based multi-core systems OpenPiton [41] and RocketCore [42], NOVA, LDPC, TATE, JPEG from *OpenCores.org*, and NETCARD from *ISPD 2012 benchmark* [43]. All the 7 benchmarks are synthesized under *TSMC 28nm* technology node using *Synopsys Design Compiler 2015*. We leverage *Cadence Innovus Implementation System v18.1* to perform

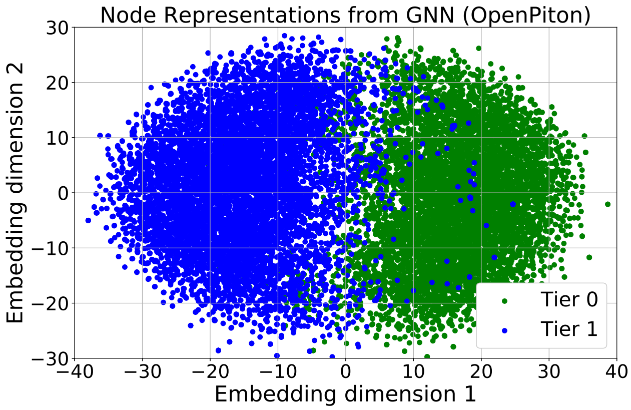


Figure 2.5: t-SNE visualizations of the learned node representations from GNN. Each dot represents a cell in the design and is colored by its final tier assignment from Algorithm 3.

placement and routing, and utilize *Synopsys PrimeTime 2018* for signoff analysis. Finally, the TP-GNN framework is implemented in *Python3* with *Tensorflow* library, and the training time is measured on a machine with 2.40 GHz CPU, 16 GB RAM, and a NVIDIA RTX 2070 graphics card. Note that for all 3D designs implemented by Shrunk2D and Compact2D, we have performed bin sweeping to find the optimal bin size for fair comparisons.

2.3.1 GNN-related Results

First, to evaluate the graph learning, we leverage t-distributed stochastic neighboring embedding [44] (t-SNE) technique to visualize the learned node representations $\{y\} \in R^{128}$ from Algorithm 12 in R^2 with OpenPiton [41]. The visualization result is shown in Figure 2.5, where we observe that the learned representations form two observable linear separable clusters. Based on the embedded locations in R^2, we further color each dot (cell) by its tier assignment from the weighted k-means algorithm (Algorithm 3) and demonstrate that the algorithm efficiently identify the two observable clusters. Now, we conclude that our TP-GNN framework is capable of transforming the initial features into meaningful high-dimension representations. In the later experiments, we demonstrate the superior achievements of TP-GNN in a complete design flow.

Table 2.2: Performance, area, and energy comparison of Shrunk-2D (S2D) [31] and TP-GNN flows on RISC-V-based designs using F2F stacking. Δ denotes the percentage difference between TP-GNN and S2D.

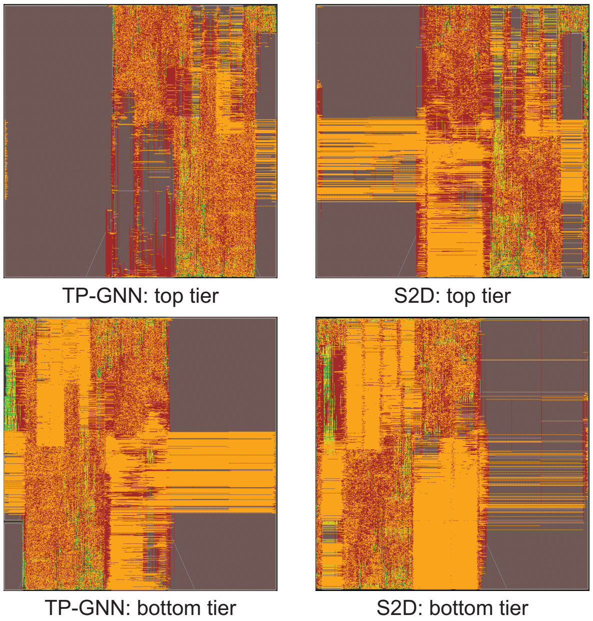
Metrics	2D	S2D	TP-GNN (Δ)
OpenPiton [ ]			
eff. freq. (MHz)	289	270	344 (27.4%)
WL (m)	6.33	4.91	4.56 (-7.7%)
energy/cycle (pJ)	343.94	339.73	270.52 (-20.3%)
footprint (mm^2)	1.22	0.61	0.61
# MIVs	0	76,083	99,423 (30.7%)
critical path WL (um)	542.6	579.3	291.7 (-49.6%)
partitioning time (min)	-	9	26
RocketCore [ ]			
eff. freq. (MHz)	832	921	964 (4.6%)
WL (m)	1.78	1.62	1.51 (-6.8%)
energy/cycle (pJ)	125.67	107.20	101.37 (-5.4%)
footprint (mm^2)	0.28	0.14	0.14
# MIVs	0	38,627	22,738 (-41.1%)
critical path WL (um)	314.2	289.4	128.9 (-55.5%)
partitioning time (min)	-	5	22

2.3.2 Maximum Performance Comparison

In this experiment, we perform maximum performance comparison between 2D, Shrunk2D, and TP-GNN flows on two RISC-V-based designs: OpenPiton [41] (# macros: 28) and RocketCore [42] (# macros: 6). Note that for designs with extensive memory macros such as OpenPiton and RocketCore, Shrunk2D significantly outperforms Compact2D. Therefore, we have taken the best-case scenario (Shrunk2D) of the existing state-of-the-art flows to perform the comparison. The results are shown in Table 2.2, where we observe that our TP-GNN flow significantly outperforms the Shrunk2D flow across the two designs. The savings in timing-related metrics are noteworthy, where the critical path wirelength saving is 52% in average and the effective frequency is 27.4% better in OpenPiton. Also, even with a higher target frequency, TP-GNN consistently large wirelength saving. Figure 2.6 further shows the GDS layout comparison, where we observe that TP-GNN introduces fewer cross-macro wires than Shrunk2D. Note that the partitioning time of the proposed framework TP-GNN includes the runtime of both graph representation learning and the

Table 2.3: Partitioning-first iso-performance comparison of Snap-3D [33] and TP-GNN flows. Δ_{Snap} denotes the percentage difference between TP-GNN and the Snap-3D flow. We report the time spend on tier partitioning in minutes.

Metrics	2D	Snap-3D	TP-GNN (Δ_{Snap})
AES (2.8GHz) (# cells: 133,051)			
WL (m)	1.78	1.35	1.34 (-0.8%)
power (mW)	229.7	213.26	219.09 (+2.7%)
# MIV	0	51,138	44,533 (-12.9%)
WNS (ps)	69	48	36 (-25.0%)
partition-time	-	3 hr	20 min
ECG (1.35GHz) (# cells: 96,416)			
WL (m)	1.17	1.02	1.01 (-0.9%)
power (mW)	307.2	286.52	284.02 (-0.9%)
# MIV	0	26,764	32,105 (+19.8%)
WNS (ps)	49	51	22 (-56.8%)
partition-time	-	3 hr	18 min
JPEG (1.53GHz) (# cells: 219,534)			
WL (m)	2.43	2.07	2.06 (-0.4%)
power (mW)	704.5	668.5	665.4 (-0.4%)
# MIV	0	31,824	32,701 (+2.7%)
WNS (ps)	68	39	28 (-28.2%)
partition-time	-	6 hr	24 min
LDPC (1.5GHz) (# cells: 41,817)			
WL (m)	1.71	1.15	1.14 (-0.8%)
power (mW)	192.2	134.9	133.6 (-0.9%)
# MIV	0	17,182	20,184 (+17.4%)
WNS (ps)	25	40	20 (-50.0%)
partition-time	-	2 hr	14 min
NETCARD (1.0GHz) (# cells: 316,137)			
WL (m)	7.82	6.77	6.78 (-0.1%)
power (mW)	651.7	632.3	635.9 (-0.5%)
# MIV	0	50,341	47,366 (+5.9%)
WNS (ps)	56	37	40 (+8.1%)
partition-time	-	8 hr	42 min
NOVA (1.0GHz) (# cells: 131,737)			
WL (m)	2.33	2.25	2.26 (-0.4%)
power (mW)	479.0	218.9	217.4 (-0.6%)
# MIV	0	18,423	17,238 (-6.5%)
WNS (ps)	47	31	29 (-6.4%)
partition-time	-	8 hr	20 min
TATE (1.37GHz) (# cells: 211,911)			
WL (m)	1.99	1.94	1.93 (-0.5%)
power (mW)	395.7	397.3	396.4 (-0.2%)
# MIV	0	54,698	60,703 (+10.9%)
WNS (ps)	36	45	42 (-6.6%)
partition-time	-	5 hr	22 min



TP-GNN: top tier

S2D: top tier

TP-GNN: bottom tier

S2D: bottom tier

Figure 2.6: GDS layouts of OpenPiton [41] using TP-GNN vs. Shrunk-2D [31] flow. TP-GNN flow achieves 7.7% better wirelength.

weighted k-means clustering algorithm. Since the graph learning is conducted in an unsupervised manner (i.e., we do not need to pre-train the model), there is no runtime overhead to apply the proposed framework.

2.3.3 Iso-Performance Comparison

In the this experiment, we perform iso-performance validation of TP-GNN in the state-of-the-art partitioning-first (Snap-3D [33]) and partitioning-last (Shrunk-2D [31] and Compact-2D [32]) M3D design flows across 6 real-world designs. Furthermore, due to the fact that active research has been conducted extensively on solving the problem of 3D placement [45, 46, 47], in this paper, we take a recent MIV-compatible 3D placement work [46] as our reference "true" 3D placement flow termed True3D. The reason we use the term "true" here is to show the difference between the (3D) placement results obtained by 3D analytical placers [45, 46, 47] and 2D commercial tools (Shrunk-2D, Compact-2D). Note that [46] does not propose a complete 3D design flow, to benchmark it against other flows in full-chip design, we further route the placements results achieved by the 3D analytical placer using Cadence Innovus.

Note that in order to reasonably benchmark the analytical approach [46] that is originally developed for TSV-based 3D ICs with other M3D flows that we focus on in this paper, we relax the penalty of inserting inter-tier vias in the objective function. The reason is because M3D technology provides much cheaper 3D stacking cost than the TSV-based 3D technology.

As aforementioned, in this paper, we validate the proposed tier partitioning framework on two different styles of M3D design flows. The results for partitioning-first design flow, Snap3D [33], are shown in Table 2.3, and the results for the partitioning-last design flows, Shrunk2D [31] and Compact2D [32], are shown in Table 2.4. In the partitioning-last design flows, we observe that TP-GNN consistently outperforms Shrunk2D and Compact2D in QoR across all designs with only a little runtime overhead in tier partitioning. As for the comparison between pseudo-3D (Shrunk2D, Compact2D) and true 3D (T3D) flow, we observe that the pseudo-3D flows consistently achieve much better PPA in terms of wirelength and power, where the T3D flow does not always obtain better QoR metrics compared with the original 2D designs.

In the partitioning-first design flow comparison, we observe that the final PPA results of the original Snap-3D and the proposed TP-GNN enhanced flow are similar. This is mainly because in such design flows, the tier partitioning stage occurs before any physical implementation (e.g. placement, routing etc.). Therefore, the impact of partitioning solutions to the QoR of the 3D full-chip design is not as direct as the case of that in the partitioning-last design flows, where tier partitioning directly determines the design quality degradation occurred by 2D-3D transformation. Furthermore, we want to emphasize that the reason Snap-3D implementation flow takes hours to perform tier partitioning is because it still relies on the partitioning solutions from Shrunk-2D to take them as placement constraints. Hence, we include the time to build the 2D implementation of Shrunk2D in the partitioning time.

Finally, head-to-head comparisons are available between partitioning-first and partitioning-last design flows. We observe that in general, the Snap-3D (partitioning-first) design flow gives better PPA metrics than the partitioning-last design flows. The key reason is that Snap-3D tricks the commercial tool to optimize 3D placements during the pseudo 2D placement stage, where both Shrunk-2D and Compact-2D require die-by-die legalization to obtain a legal 3D placement solution after tier partitioning, which may degrade the quality of the obtained partitioning solutions.

2.3.4 Sweeping Experiments on Contracting Edges

In this experiment, we demonstrate the PPA effect of executing different number of times of the hierarchy-aware edge contraction algorithm (Algorithm 1) on the LDPC benchmark. The results are shown in Table 8.7. As aforementioned, the designs built by TP-GNN in this work are achieved by running the algorithm two times. The straightforward benefit of running Algorithm 1 is to prevent the short nets in the original 2D designs from being partitioned into two separate tiers and turned into long nets, which may cause severe QoR degradation. Nonetheless, as shown in the table, over-running the algorithm may as well incur the QoR degradation in final full-chip design, because some optimization opportunities are lost when various nodes are forced to be merged into one node.

2.3.5 Results on Heterogeneous 3D ICs

In this experiment, we validate the proposed framework on a heterogeneous 3D IC of a commercial CPU-core based on the Pin3D [36] design flow, where fast corner is used for top tier and slow corner is used for bottom tier (both in foundry 28nm). The results are shown in Table 2.5. First, we observe that the proposed TP-GNN framework improves many critical QoR metrics of the original Pin3D design flow. Second, we find that with the post-partitioning optimization algorithm, TP-GNN can further optimize the full-chip PPA with little runtime overhead. In particular, the performance of the 3D full-chip design is

Table 2.4: Partitioning-last iso-performance comparison of True3D (T3D) [46], Shrunk-2D (S2D), Compact-2D (C2D), and TP-GNN flows. Δ_S and Δ_C respectively denotes the percentage difference between TP-GNN vs. S2D and C2D. We report the time spend on tier partitioning in minutes.

Metrics	2D	T3D	S2D	C2D	TP-GNN (Δ_S, Δ_C)
AES (2.8GHz) (# cells: 133,015)					
WL (m)	1.78	1.86	1.45	1.42	1.46 (-0.6%, -2.8%)
power (mW)	229.7	233.1	217.8	215.2	218.3 (-0.2%, -1.4%)
# MIV	0	43,895	41,262	39,403	44,921 (+8.8%, -14.0%)
WNS (ps)	69	51	33	38	41 (+24.2%, +7.8%)
partition-time	-	-	5	5	20
ECG (1.35GHz) (# cells: 96,416)					
WL (m)	1.17	1.55	1.06	1.07	1.05 (-0.9%, -1.8%)
power (mW)	307.2	311.4	290.2	291.3	288.9 (-0.4%, -0.8%)
# MIV	0	19,843	13,681	14,525	14,028 (+2.5%, -3.4%)
WNS (ps)	49	18	80	62	35 (-56.2%, -43.5%)
partition-time	-	-	3	3	18
JPEG (1.53GHz) (# cells: 219,534)					
WL (m)	2.43	4.93	2.09	2.12	1.94 (-7.2%, -8.5%)
power (mW)	704.5	727.9	674.2	675.9	665.1 (-1.3%, -1.6%)
# MIV	0	34,190	27,839	28,231	27,154 (-2.5%, -3.8%)
WNS (ps)	68	20	49	41	23 (-53.1%, -43.9%)
partition-time	-	-	8	8	24
LDPC (1.8GHz) (# cells: 43,381)					
WL (m)	1.78	1.97	1.61	1.57	1.42 (-11.8.%, -9.6%)
power (mW)	362.5	311.1	301.4	292.5	271.4 (-10.0%, -7.2%)
# MIV	0	12,509	8,955	9,237	7,454 (-25.6%, -27.9%)
WNS (ps)	34	29	26	20	16 (-38.5%, -20.0%)
partition-time	-	-	2	2	14
NETCARD (1.0GHz) (# cells: 316,137)					
WL (m)	7.82	8.66	6.83	6.87	6.11 (-10.5%, -11.1%)
power (mW)	651.7	702.4	639.8	639.2	598.9 (-6.4%, -6.3%)
# MIV	0	54,403	43,823	43,754	39,987 (-8.8%, -8.6%)
WNS (ps)	56	11	51	49	26 (-49.0%, -46.9%)
partition-time	-	-	14	14	42
NOVA (1.08GHz) (# cells: 131,737)					
WL (m)	2.33	2.45	2.30	2.28	2.17 (-5.7%, -4.8%)
power (mW)	479	396.5	220.2	216.9	211.0 (-4.6%, -2.7%)
# MIV	0	19,922	16,672	16,935	15,813 (-5.2%, -6.6%)
WNS (ps)	47	0	28	25	19 (-32.1%, -24.0%)
partition-time	-	-	5	5	20
TATE (1.37GHz) (# cells: 211,911)					
WL (m)	1.99	2.41	1.97	1.95	1.92 (-2.5%, -1.5%)
power (mW)	395.7	439.2	398.4	398.6	396.5 (-0.4%, -0.5%)
# MIV	0	70,457	56,467	56,820	59,727 (5.8%, 5.2%)
WNS (ps)	36	0	87	76	31 (-64.4%, -59.2%)
partition-time	-	-	8	8	22

Table 2.5: Iso-Performance comparison on a heterogeneous 3D design of a commercial CPU design implemented by Pin3D [36]. TP-GNN$_{opt}$ denotes the post-partitioning optimization (Algorithm 4) is enabled.

Metrics	Pin3D	TP-GNN	TP-GNN$_{opt}$
commercial CPU design in 1.2GHz # cells: 176,352, # macros: 21			
WL (m)	3.17	3.11 (-1.9%)	3.07 (-3.2%)
total power (mW)	203.3	192.5 (-5.3%)	189.0 (-6.8%)
internal power (mW)	96.3	90.9 (-5.6%)	88.4 (-8.2%)
switching power (mW)	93.9	88.5 (-5.7%)	86.5 (-7.9%)
# MIV	30,155	34,068	28,883
WNS (ns)	0.452	0.237 (-47.5%)	0.085 (-81.2%)
partition-time	7 min	18 min	21 min

Table 2.6: Sweeping experiments on running hierarchy-aware edge contraction algorithm (Algorithm 1) multiple times.

LDPC	0-run	1-run	2-run	3-run	4-run
WL (m)	1.49	1.44	1.42	1.43	1.47
power (mW)	277.5	272.1	271.4	272.8	280.5
# MIV	8,130	9,269	7,454	7,526	6,012
WNS (ps)	26	14	16	22	31
partitioning-time	20	18	14	13	11

pushed much higher.

We attribute the success of TP-GNN in heterogeneous 3D ICs in two-fold. First, due to the fact that TP-GNN comprehends the technology features, timing related information is taken into account while performing tier partitioning, where the original partitioning algorithm that Pin3D adopts only considers to minimize the cutsize of connections. Second, the post-partitioning optimization algorithm reckons the idea that cells on critical paths should be considered carefully, which prevents the severe timing degradation that happens in the original Pin3D design flow.

2.4 Discussion

2.4.1 Why Does GNN Work Better?

Across the experiments for various fashions of M3D design flows, we observe that TP-GNN framework significantly improves the timing from the state-of-the-art flows in consistent. The main reason is that the original bin-based partitioning method ignores the global connections among bins. It only partitions the sub-netlist within a bin. Therefore, critical nets in the projected 2D designs are partitioned randomly. In TP-GNN framework, we solve this issue by introducing timing related features to the graph learning, which encourages cells on critical nets to be partitioned into same tier.

Furthermore, we observe that TP-GNN framework achieves great wirelength savings, which can be explained by two reasons. First, Algorithm Algorithm 1 prevents nearby and connected cells from being partitioned into different tiers, which reduces the significant 3D routing overhead occurred in Shrunk2D and Compact2D flows. Second, with the structural features introduced in Table Table 9.1 and the message passing characteristics of the graph learning, cells that are logically connected would have similar node representations. Therefore, unlike bin-based partitioning method that partitions long nets randomly, our framework partitions long nets based on the netlist structure.

Finally, we want to emphasize that TP-GNN runtime is measured from the beginning of Algorithm Algorithm 1 to the end of Algorithm Algorithm 3. The runtime of our GNN-based tier partitioning algorithm basically involves training our GNN using unsupervised learning. Therefore, we do not report training vs. inferencing time separately as our GNN learns while traversing the nodes in netlist graphs and collecting features from their neighbors. The time complexity of our TP-GNN is linear in terms of the netlist size. This is because our GNN model visits all the nodes in the netlist graph and spends a constant amount of time collecting features from the neighbors. The total number of neighbors for a given node under consideration is constant as we limit our neighbor search within a fixed

Table 2.7: Timing impact between 3D (cross-tier) and 2D (same-tier) nets on ECG benchmark. The unit for net length is μm, and the unit for delay is ps.

net type	# path	avg. length	avg. delay	wst. delay
cross-tier	43,645	21.2	1.3	58.9
same-tier (top)	86,083	7.6	0.3	35.7
same-tier (bot.)	101,493	9.5	0.4	29.5

hop count.

2.4.2 MIV Impact

In the experiments, we do not refrain the router to insert MIVs during the routing stage. This is because as pointed out in [28], the inter-tier vias density of M3D designs can achieve up to $100million/mm$ in a $14nm$ technology node, which leads to the conclusion that in M3D designs, no inter-tier via density constraints are needed as in TSV-based designs. Furthermore, as mentioned in [48], the minimization of MIV count no longer has a major impact on the full-chip PPA of M3D designs. Therefore, in this paper, we do not strive for minimizing the MIV count.

2.4.3 Timing Impact on Crossing Tiers

Due to the small pitch and low parasitic of MIVs (nano-scale), in M3D designs, the timing impact of a net crossing different tiers is not as severe as in TSV-based designs. Since MIVs possess similar RC characteristics to regular vias, the timing delay of a net in M3D designs is proportional to its time (RC) constants. Table Table 2.7 quantifies the delay and wirelength between cross-tier and same-tier nets on the ECG benchmark. As shown in the table, although the average net delay of cross-tier nets is higher than same-tier nets, the reason behind is that cross-tier nets tend to have longer wirelength and therefore higher RC timing constants.

2.5 Conclusion

In summary, we have proposed TP-GNN, a novel tier partitioning framework based on graph neural network. First, we propose a hierarchy-aware edge contraction algorithm to reduce the severe 3D routing overhead occurred in the bin-based partitioning algorithm. Then, we map the classical tier partitioning problem into a clustering problem and solve it with advanced machine learning techniques. The graph representation learning provides the freedom for designers to deal with various partitioning objectives, and the unsupervised learning promises the generality.

CHAPTER 3
VLSI PLACEMENT OPTIMIZATION VIA PPA-DIRECTED SELF-SUPERVISED DEEP GRAPH CLUSTERING

3.1 Background and Motivation

It is widely acknowledged that placement is the heart of every Physical Design (PD) flow as the cell locations determined at this stage directly impacts the on-chip interconnects and hence the capacitances and resistances induced, which are closely related to the Power, Performance, and Area (PPA) metrics of the final full-chip design [49]. Driven by Moore's Law, modern Very-Large-Scale-Integrated (VLSI) designs easily consist of millions of instances that are required to be placed and routed. However, existing placement algorithms in commercial tools leverage various heuristics or analytical methods that do not scale globally, which often leads to sub-optimal optimization results in advanced technologies. Furthermore, modern VLSI designs are commonly over-complicated for tools' inherent algorithms to optimize in a systematic manner. Hence, in real-world practice, designers often provide additional instance grouping information to commercial placers in order to guide placement optimization towards a better direction. In this work, instead of manually specify cell clustering constraints as placement guidance, we explore Machine Learning (ML) techniques to automate this process unsupervisedly [8].

3.1.1 Related Works: ML in VLSI Placement

To improve VLSI placement using ML techniques, several works have been proposed to predict placement metrics using supervised learning. A recent work [50] developed an ensemble framework to predict congestion and timing metrics based on a massive database that contains 72 industrial designs with millions of instances. Another work [10] utilized

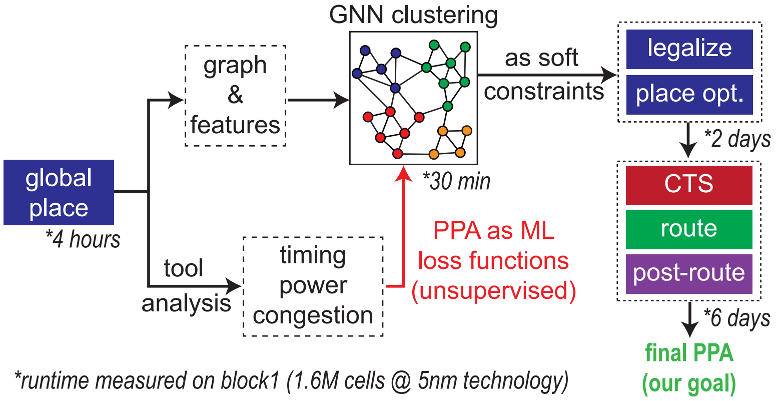


Figure 3.1: Our PPA-directed placement optimization framework in an industrial PD flow.

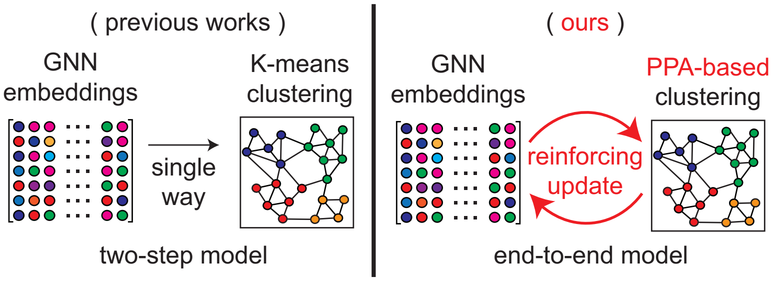


Figure 3.2: Difference between prior works [6, 7] and ours. Our framework is end-to-end differentiable and directly optimizes PPA metrics as ML loss functions.

reinforcement learning (RL) to improve placement quality by tuning dozens of placement parameters through thousands of placement iterations. Still another work [51] further combined simulated annealing and RL in a cyclic framework to conduct iterative placement optimization. The drawbacks of these works are self-evident that they either require a huge database to be pre-generated or an extremely long runtime for learning convergence. Not to mention that their learning-based models are limited to the designs or technologies which they are trained upon. Therefore, whether their models are generalizable to unseen netlists is questionable.

Unlike supervised learning that requires pre-generated labels for training, unsupervised learning strives to discover hidden patterns of input data through self-representation learning. In the realm of EDA, since VLSI netlists are essentially hypergraphs, graph neural networks (GNNs) have been widely used to distill netlist information in an unsupervised manner. Recently, previous works [6, 7] both utilized GNNs to perform node representa-

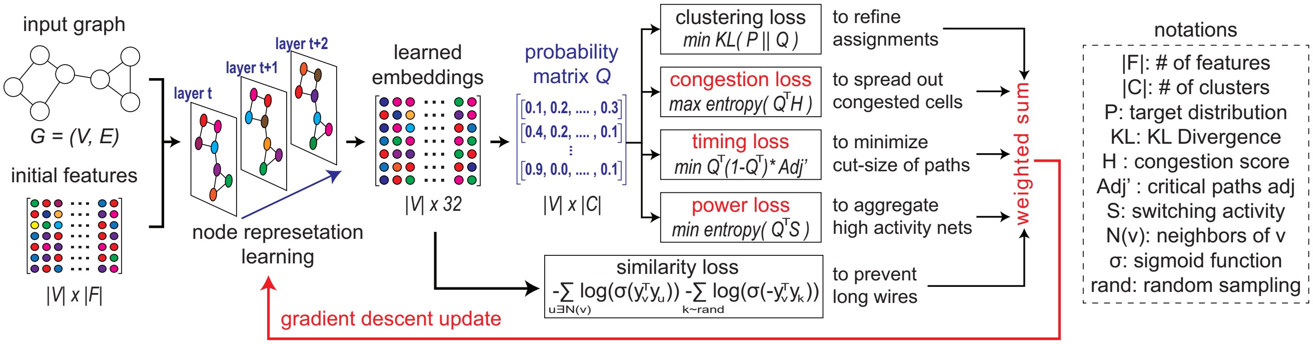


Figure 3.3: Our PPA-directed unsupervised deep graph clustering framework. Given a netlist graph, initial node features, and PPA tool analysis (congestion, timing, power), our framework is trained to optimize PPA metrics as ML loss functions by jointly improving the node embeddings and the clustering assignments in an end-to-end manner.

tion learning to extract the netlist characteristics that are related to placement optimization, upon which the weighted K-means clustering algorithm [34] is applied to identify the cell clusters that are essential to improve the underlying placement. However, this two-step approach often leads to inferior optimization results. The main reason is that the node representation learning conducted by GNNs is not "goal-directed" because the embedding step and the clustering step are not end-to-end differentiable. This implies neither can their GNN models generate better embeddings through the evaluations of the clustering results, nor can their clustering algorithms refine the assignments based on the improved node embeddings.

3.1.2 Our Goal: PPA-Directed Placement Optimization

in this chapter, we aim to overcome the above issue by developing an end-to-end, PPA-directed placement optimization framework. Instead of relying a two-step approach as previous works [6, 7], our framework jointly improves both GNN embeddings and clustering assignments by directly optimizing PPA metrics as ML loss functions. Figure 3.1 shows an overview of the proposed framework in an industrial design flow and Figure 4.2 illustrates the key difference between our work and the previous related works [6, 7]. The only assumption of this work is that there exists some cell clustering assignments that are beneficial to the underlying placement and the subsequent PD stages if being optimized timely, and the main objective of this paper is to present a low-cost yet highly effective technique to identify such clusters.

As shown in Figure 3.1, given an initial placement, the proposed framework learns to discover the cell clusters that are critical for post-route PPA improvements by directly minimizing PPA metrics as ML loss functions, which are formulated unsupervisedly from the timing, power, and congestion analysis based on the current placement. During the learning process, both node representation learning conducted by GNNs and the clustering assignments are jointly optimized through gradient descent. After the learning is complete, the final clustering assignments are taken as soft constraints to the subsequent placement optimization steps where the commercial placer will spend effort in placing cells in the same cluster close to each other. Note that the entire learning process is conducted in an unsupervised manner and does not require any pre-generated database, which implies the proposed framework can be applied to any unseen design or technology. Finally, the detailed runtime of each stage measured with one of our benchmarks is also shown in Figure 3.1. Compared with the entire design flow, the proposed placement optimization technique only introduces negligible runtime overhead.

The goal of this work is to present a placement optimization framework that has a strong positive impact on the PPA quality at the end of the flow with small runtime overhead. It is no doubt that the ultimate goal of every PD implementation is to meet the end-of-flow PPA target closures, and we believe the best way to achieve this is to start from a better placement. The key contributions of this paper are summarized as follows:

- To the best of our knowledge, we are the ***first*** work that directly formulates PPA metrics as ML loss functions and optimizes them to improve the placement quality of commercial tools that are widely used across the entire semiconductor industry.

- To the best of our knowledge, we are the ***first*** work that develops an end-to-end unsupervised clustering framework in the realm of EDA where the embedding learning and the clustering assignments are jointly updated in a goal-directed manner.

- We validate the proposed framework in an industrial PD flow using a commercial $5nm$ technology and benchmarks with *millions of cells*. We not only show that our framework immediately improves the PPA metrics at the placement stage, but also demonstrate that the improvements last firmly to the post-route stage.
- We show that the proposed PPA-inspired ML loss functions can be directly leveraged for placement evaluation, which correlate well with the end-of-flow PPA metrics.
- We show that the proposed framework can not only be integrated with industry-leading commercial tools, but can also be combined with the state-of-the-art academic placer DREAMPlace [24] and greatly improve design quality.
- We demonstrate that the proposed framework PPA-GNN can be integrated with predictive models so as to enable ad-hoc optimization on the predicted metrics.

3.2 Framework Overview

Before diving into the details, we first present an overview of our framework as shown in Figure 3.3. In summary, our framework leverages unsupervised deep embedded clustering [52] equipped with GNN representation learning to identify the cell clustering assignments that can be used to optimize design PPA metrics. The inputs to our framework are a netlist graph $G = (V, E)$, initial node features $Y^0 \in R^{|V|\text{x}|F|}$, and tool-based PPA analysis of the underlying placement, which includes congestion scores $H \in R^{|V|}$, maximum switching activities $S \in R^{|V|}$, and the adjacency matrix of timing critical paths Adj'. The key output of our framework is the probability matrix $Q \in R^{|V|\text{x}|C|}$ where each element Q_{ij} represents the probability of a cell i belonging to a cluster j. During the learning process, our framework will jointly refine the node embeddings and the clustering assignments by minimizing the proposed PPA-inspired ML loss functions and other objectives using a gradient descent optimizer.

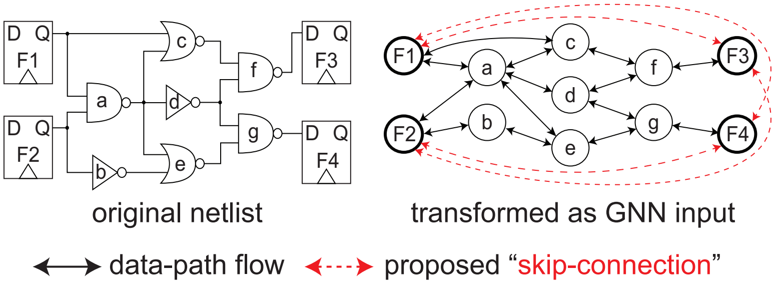


Figure 3.4: Proposed netlist transformation that honors timing propagation. "Skip-connections" link start points and end points of timing paths during GNN message passing.

The main motivation behind the proposed clustering-based placement optimization framework is that if we know a path is timing critical, or if we know a net is in high switching activity, then we would like to shorten the path or the net by moving cells closer to each other in order to reduce the resistances and capacitances involved. In addition, if we know an area is highly congested, then we would want to spread out the cells within to reduce the congestion as it directly impacts the routing afterwards. To summarize, during the placement optimization phase, we want to improve the cell locations based on the current PPA evaluations. In fact, similar strategies [53, 54, 55, 56] have been deployed in placement tools throughout the years using analytical techniques. In this work, we aim to unleash the power of ML to achieve similar optimization goals but in a more systematic and global manner by directly formulating crucial PPA metrics as ML loss functions. Finally, the key difference between our work and previous works [6, 7] is summarized in Figure 4.2. Unlike previous works requiring a two-step approach to identify essential cell clusters, we develop an end-to-end PPA-directed framework that achieves better optimization results. As aforementioned, our main assumption is that the final placement quality can be significantly improved if certain clustering constraints can be acknowledged during the optimization phase. The main focus of this work is to discover such cell clustering assignments using ML algorithms in a fully-automated manner.

3.3 Algorithms

3.3.1 Netlist Transformation

Since a VLSI netlist is inherently a hypergraph $G = (V, E)$ and the node representation learning conducted by GNNs relies on an adjacency matrix $A \in R^{|V| \text{x} |V|}$ where each element $A_{ij} \in \{0, 1\}$ denotes whether learning messages can be passed from node i to node j, a netlist transformation is needed prior to GNN representation learning. Previous GNN-based placement optimization works [6, 7, 10] all adopted the renowned clique-based model [57] for the transformation. However, this approach suffers from the fact that the number of edges in the transformed graph grows quadratically to the number of nodes in the original netlist, which not only causes memory issues on industrial designs, but also weakens the expressiveness of representation learning as it will be hard for GNNs to identify important node connections from a large amount of edges.

To overcome these issues, we propose a new transformation method as shown in Figure 3.4. The proposed method brings two timing-related improvements upon the clique-based technique. First, for every net in the original netlist, we only introduce the driver-to-load connection(s) in the transformed graph instead of forcing every cell on the same net to share connections with each other. Second, given that the receptive field of a GNN model is limited by its number of layers, we introduce "skip-connections" (denoted in red) to link start points and end points of timing paths, which enables GNNs to more easily capture timing-related effects. In our transformation, the number of edges in the transformed graph grows pseudo-linearly to the number of nodes in the original netlist, which is fully applicable on industrial designs.

3.3.2 Node-Level Feature Collection

Prior to the graph learning, we compute initial node-specific features for each design instance (i.e., cell). These features are carefully determined based on domain expertise and are expected to characterize an instance's impact to the optimization engine.

Hierarchical Cell Name Encoding using Transformer

Previous works [6, 7] have proven that the design hierarchy information is critical to placement optimization as cells within the same hierarchy tend to have more connections with each other. However, they either utilize simple integer-based encoding [6] or a suffix-tree model [7] to encode such hierarchical information, which is not desirable for ML applications. Because ML models will assume numerically similar inputs (e.g., integers that are close to each other) to possess resembling meanings, which may not be the case in design hierarchies. Unlike previous works that leverage deterministic approaches, in this work, we leverage the renowned sentence-BERT model [58] to encode the hierarchy information for every cell in the design by considering their hierarchical names as sentences, where each hierarchy or sub-hierarchy denotes a token (i.e., word). Given any design, the sentence-BERT model will perform self-supervised learning to construct the embeddings of each token based on their positions in different hierarchical names (i.e., sentences) and synergy with other tokens. After the learning is complete, the name embeddings of each cell are obtained by taking mean-pooling across the representations of each token within.

Logic-to-Memory Affinity

Due to the fact that logic-to-memory paths are often the critical paths in modern VLSI designs, balancing the locations of cells on these critical paths may prevent wires from detouring, which possibly improves timing [9]. Following from [6], for each cell, we compute its shortest logic distance to each memory macro in the underlying design as initial features.

Timing-Related Features

To capture the timing impact, we compute the worst slack value at a cell's output pin and

Table 3.1: Our initial node features for deep graph clustering. M denotes the number of memory macros.

type	# dim.	description
name embeddings	16	hierarchical name encoded by S-BERT [58]
memory affinity	M	shortest logic distance to each memory
wst output slack	1	worst slack value at output pin
wst output slew	1	maximum transition at output pin
wst input slew	1	maximum transition among input pin(s)
largest activity	1	largest switching activity value among nets
locations	2	(x,y) location of initial placement

the worst slew values at both pins as its timing features using a commercial static timing analysis (STA) engine. Therefore, cells on timing critical paths will be enforced to have similar timing-related features, which increases the probability of them being assigned to the same cluster. During placement optimization, if these critical cells are moved closer to each other, the overall timing will likely be improved. Note that apart from this feature-wise timing enhancement, in subsection 3.3.4, we will present more details on the approach of optimizing timing as an ML loss function directly.

Power-Related Features

Power is another crucial placement objective besides timing. To improve power dissipation, in this work, we specifically focus on improving the dynamic power by taking the largest switching activity among the nets that each cell is connected to as one of its initial features. The key idea is that if we know a net is consuming excessive switching power, it is usually power-wise beneficial to move the cells on this net closer to each other. Again, as aforementioned, besides this feature-wise improvement, we will present more details in later sections on how to systematically improve power consumption by formulating it as a learning objective. Finally, Table Table 9.1 summarizes the initial features we collect for each cell in the design. We want to emphasize that for different designs, the total dimension of the initial cell features will vary depending on the number of pre-placed memory macros.

3.3.3 Node Representation Learning

After collecting the initial features for each cell in the design, we leverage GNNs to perform node representation learning. The main objective of the representation learning is to transform the initial node features into meaningful embeddings by considering the connectivity among cells. In the realm of PD, the embeddings learned by many variants of GNNs have been leveraged to solve the downstream tasks including 2D/3D placement optimization [9, 10, 7, 6], PPA predictions [19, 59, 17, 60, 61], gate sizing [12, 11, 13], activity simulation [14, 62] etc. In this work, considering the runtime and memory benefits that graph inductive learning brings, we leverage GraphSAGE [37] to perform the node representation learning as:

$$\begin{aligned} y_{N(v)}^{k-1} &= mean\_pool\left(\{\mathbf{W}_k^{agg} y_u^{k-1}, \forall u \in N(v)\}\right), \\ y_v^k &= sigmoid\left(\mathbf{W}_k^{proj} \cdot \text{concat}\left[y_v^{k-1}, y_{N(v)}^{k-1}\right]\right), \end{aligned} \tag{3.1}$$

where $N(v)$ denotes the neighbors of node v, W_k^{agg} and W_k^{proj} denote the aggregation and projection matrices at the k-th layer of the GNN module. After the transformation through Equation 9.1, the initial node features of each cell y_v^0 will be transformed into y_v^K, where K denotes the total number of layers. The dimensions of y_v^K is subject to the number of neurons at the last layer. In the implementation, we set $K = 6$ and $dim(y_v^K) = 32$.

Similarity Loss L_{sim}

Now, we define the first objective function of the proposed framework, which is termed as the "similarity loss" and can be directly calculated from the GNN learned embeddings. The key idea of this loss function is to let nodes that are logically close to each other have similar representations, while making nodes that are logically distant have dissimilar embeddings.

The loss function L_{sim} is designed as:

$$L_{sim} = \sum_{v} \left(- \sum_{u \in N(v)} \log \left(\sigma(y_v^\top y_u) \right) - \sum_{k \sim rand} \log \left(\sigma(-y_v^\top y_k) \right) \right) , \tag{3.2}$$

where y_v denotes the learned embeddings of node v, σ denotes the sigmoid function, and $rand$ denotes the random sampling operation over the full netlist graph. By minimizing Equation 3.2, neighboring nodes will be encouraged to have similar embeddings y, which increases the probability of them being assigned to the same cluster and hence prevents creating long pin-to-pin connections. Therefore, as shown in Figure 3.3, we consider Equation 3.2 as preventing the creation of long wires.

3.3.4 PPA-Directed Deep Graph Clustering

One of the highlights of our work is that instead of using the K-means clustering algorithm [34] to discover the cell clusters as in many previous works [7, 6], we devise a methodology to transform the GNN learned node embeddings into a probability matrix $Q \in R^{|V| \text{x} |C|}$ as shown in Figure 3.3, where each element Q_{ij} denotes the probability of a node i belonging to a cluster j. With this probability matrix Q, we can further formulate the traditional PPA metrics as ML loss functions by calculating the “expected PPA impact” of the current clustering assignments. Then, we can leverage gradient descent to optimize these PPA metrics by jointly improving node representation learning and cell clustering assignments. We want to emphasize that unlike previous works [7, 6], the entire framework from representation learning to clustering is end-to-end differentiable. In the following sub-sections, we present step-by-step details on constructing the probability matrix Q and formulating the proposed PPA-inspired ML loss functions.

Unsupervised Clustering Loss L_{cl}

One of the main challenges of the clustering task is the non-existence of label guidance. To overcome this challenge, we devise a self-reinforcing method that iteratively converts "distances" of trained GNN node embeddings $\{y\}$ into "probabilities" of clustering assignments. Particularly, we leverage the Student's t-distribution [63] as a kernel to perform the distance-to-probability conversion as:

$$Q_{ic} = \frac{\left(1 + \|y_i - \mu_c\|^2\right)^{-1}}{\sum_k \left(1 + \|y_i - \mu_k\|^2\right)^{-1}}, \tag{3.3}$$

where Q_{ic} denotes the probability of node i belonging to cluster c, y_i denotes the learned GNN node embeddings of node i, and μ_c denotes the embeddings of centroid c which is a trainable vector that is improved in every iteration, and $\|\cdot\|^2$ denotes the Euclidean distance. Note that Q is a stochastic matrix, which means every element is greater or equal to 0 and every row sums up to 1.

To optimize the clustering assignments (i.e., matrix Q) in a self-reinforcing manner [64], we further construct a target matrix P by strengthening the assignments of Q as:

$$P_{ic} = \frac{Q_{ic}^2 / \sum_i Q_{ic}}{\sum_k Q_{ik}^2 / \sum_i Q_{ik}}. \tag{3.4}$$

The rationale behind Equation 3.4 is that since Q is a stochastic matrix which means $0 \leq Q_{ic} \leq 1$, raising and then normalizing by the second power will make the probability distribution of each row (i.e., assignment distribution of a cell) skew towards to the largest value. Hence, the assignments are strengthened. Now, with the target matrix P and the approximate matrix Q, we can define the clustering loss L_{cl} as:

$$L_{cl} = KL(P||Q), \tag{3.5}$$

where KL denotes the Kullback-Leibler divergence [65]. Minimizing Equation 3.5 will

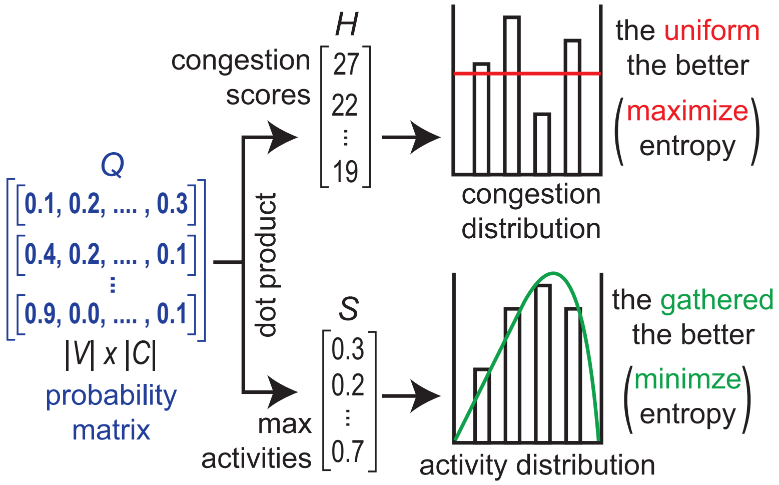


Figure 3.5: Illustration of congestion loss and power loss formulations that rely on entropy maximization and minimization. Note that both distributions are normalized as probabilities.

encourage the matrix Q to approximate the matrix P. To this end, we have bridged the gap between node representation learning and cell clustering by converting the GNN node embeddings into cell clustering probabilities. With the probability matrix Q that represents clustering assignments, we can further quantify its "expected" impact on important PPA metrics, which allows us to perform gradient descent in an end-to-end manner between node representation learning and cell clustering. However, the above presented techniques are still not sufficient to truly discover the cell clusters that are critical for placement optimization as they are not "goal-directed". To overcome this issue, in the following sub-sections, we complete the proposed framework by formulating the conventional PPA metrics as ML loss functions.

Congestion Loss L_{cong}

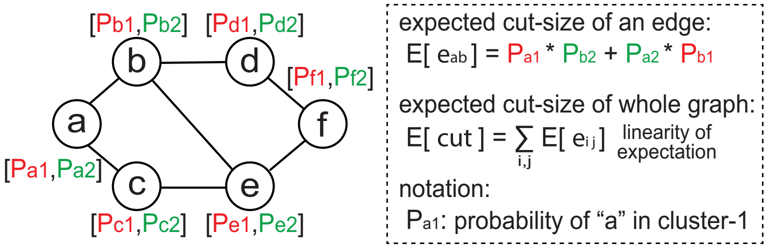
Congestion is one of the most important placement objectives as it highly correlates with the on-chip routability which directly impacts the performance and power of the full-chip design [59]. A placement even with good wirelength estimation is still not usable if the underlying congestion is poor. In this work, to reduce cell congestion, we adopt the concept of cell spreading as many renowned works [66, 54, 55] by introducing a density-aware objective to our framework. The key idea is that if we have the information regarding the current congestion hotspots, we can train the framework to adjust the clustering assignments in order to spread the cells out from those “hot” regions. We formulate this objective and the corresponding loss function L_{cong} as:

$$\max\ entropy\left(Q^\top H\right) \rightarrow L_{cong} = -entropy\left(Q^\top H\right), \tag{3.6}$$

where $H \in R^{|V|}$ is a vector that denotes the congestion score of each cell, $Q^\top H \in R^{|C|}$ thus represents the expected congestion score of each cluster, and finally $entropy(\cdot)$ denotes the function mapping that first normalizes each element by the sum of all elements, and then calculates the Shannon entropy [67] of the normalized probability vector. By maximizing the entropy, the probability matrix Q will be encouraged to find the clustering assignments that spread out cells in the congested regions as the maximum entropy is achieved by having an equal amount of congestion (i.e., $\frac{\sum_v H_v}{|C|}$) in each cluster. The illustration of our congestion objective is shown in the upper-part of Figure 3.5.

Timing Loss L_{timing}

Performance has been the dominant PPA metric in the past decade, where research on timing-driven placement (TDP) has been conducted extensively. Although many ML-based timing optimization techniques [61, 11] have been proposed to improve timing Engineering Change Orders (ECOs) in late design stages, it is widely acknowledged that the best way to improve sign-off performance is to start from a better placement. Generally speaking, TDP is designed specifically to improve interconnections on timing critical paths [56], where popular approaches can be categorized into two streams: net-based and path-based. Since the goal of the proposed framework is to discover the critical cell clusters from a global placement, we reckon that by taking the path-based approach, we can improve the timing more easily without changing the underlying placement drastically.


[Pb1,Pb2]
[Pd1,Pd2]
[Pf1,Pf2]
[Pa1,Pa2]
[Pc1,Pc2]
[Pe1,Pe2]
a
b
c
d
e
f
expected cut-size of an edge:
E[eab] = Pa1 * Pb2 + Pa2 * Pb1
expected cut-size of whole graph:
E[cut] = ∑i,j E[eij] linearity of expectation
notation:
Pa1: probability of "a" in cluster-1

*example is a two-way partition, but can easily be extended to multi-way

Figure 3.6: Illustration of "cut-size" loss formulation.

In this work, our key idea to improve timing is to let the cells on timing critical paths have higher chances in being clustered into the same group and hence the wires in between can be shortened during placement optimization. To achieve this, as in [68], we formulate the "cut-size" of timing critical paths as an ML loss function, which is resulted from the current clustering assignments Q. Figure Figure 3.6 shows an illustration of our cut-size loss formulation. Note that although a two-way partitioning example is shown in the figure, this formulation can be easily extended to handle multi-way partitioning by considering more probability combinations. In a generalized matrix form representation for $|C|$-way partitioning, we formulate the cut-size as timing loss L_{timing} based on the probability matrix $Q \in R^{|V|\text{x}|C|}$ as:

$$L_{timing} = reduce\_sum\left(Q^\top\left(1 - Q^\top\right) \odot A_{critic}\right), \tag{3.7}$$

where A_{critic} denotes the adjacency matrix of timing critical paths, $reduce\_sum(\cdot)$ denotes the operation that adds up all the input elements, and $\odot$ denotes the element-wise multiplication. By minimizing Equation 3.7, the Q will be encouraged to improve the clustering assignments to minimize the cut-size of timing critical paths.

Power Loss L_{power}

Recently, with the burgeoning surge in the demand for hand-held and wearable devices, low-power has become the prime objective in many design implementation flows. Aside from register banking, most of the low-power placement approaches leverage the technique of activity-aware net weighting [69] to reduce dynamic power, where the key idea is to make nets with high switching activities have shorter physical wirelength to reduce their capacitances. Inspired by this concept, in this work, we formulate the power objective and the loss function L_{power} as:

$$\min\ entropy\left(Q^{\top}S\right) \rightarrow L_{power} = entropy\left(Q^{\top}S\right), \tag{3.8}$$

where $S \in R^{|V|}$ represents the largest switching activity of the nets that a cell is connecting to. The idea behind Equation 3.8 is similar to that of the congestion loss as shown in the bottom-part of Figure 3.5. The only difference is that here we are minimizing rather than maximizing the entropy to aggregate the cells that are connected to high switching activity nets in order to shorten their interconnects.

3.3.5 End-to-End Unsupervised Training

Now, after describing all the objectives of our framework as summarized in Figure 3.3, we jointly optimize them using a gradient descent optimizer Adam [38] to minimize the weighted sum of each objective as:

$$L = L_{sim} + \lambda_1 L_{cl} + \lambda_2 L_{cong} + \lambda_3 L_{timing} + \lambda_4 L_{power}, \tag{3.9}$$

where $\lambda_i \geq 0$ controls the contribution of each objective to the clustering assignment. After the training is complete, we obtain the final clustering assignment of each cell v as:

$$\text{assignment of node } v = \operatorname*{argmax}_{c} Q_{vc}. \tag{3.10}$$

Algorithm Algorithm 5 summarizes the entire end-to-end training process. In Lines 1–4, we first pre-train the weights of the GNN module using similarity loss (i.e., Equation 3.2).

Algorithm 5 End-to-End Unsupervised Training Methodology.
We use default values of $sim\_epoch = 10, full\_epoch = 50, \alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \lambda_1 = 1, \lambda_2 = 10, \lambda_3 = 1, \lambda_4 = 0.5$.

```
Input: $G = (V, E)$: transformed graph. $\{y^0\}$: initial node features. $H \in R^{|V|}$: congestion scores.
    $A_{critic}$: critical path adjacency matrix. $S \in R^{|V|}$: maximum switching activities. $\{W\}$:
    weights of GNN. $sim\_epoch$: number of epochs for similarity-only learning. $full\_epoch$:
    number of epochs for full-objective learning. $\{\beta_1, \beta_2\}$: Adam params. $\alpha$: learning rate.
    $\{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}$: objective weights.
Output: $Z \in R^{|V|}$: final clustering assignment of each cell
 1: for $i = 0$; $i < sim\_epoch$; ++$i$ do                                   ▷ Pre-train GNN weights
 2:     $y \leftarrow$ GNN$(G, y^0; W)$                                        ▷ GNN embeddings by Equation 9.1
 3:     $L_{sim} \leftarrow sim\_loss\,(y)$                                    ▷ similarity loss by Equation 3.2
 4:     $W \leftarrow Adam\,(L_{sim}, \beta_1, \beta_2, \alpha; W)$            ▷ update GNN
 5: $\{\mu\} \leftarrow$ obtain initial centroids from $y$ using K-means
 6: add $\{\mu\}$ to ML computational graph                                    ▷ make $\{\mu\}$ trainable
 7: for $i = 0$; $i < full\_epoch$; ++$i$ do
 8:     $y \leftarrow$ GNN$(G, y^0; W)$                                        ▷ GNN embeddings by Equation 9.1
 9:     $L_{sim} \leftarrow sim\_loss\,(y)$                                    ▷ similarity loss by Equation 3.2
10:     $Q \leftarrow$ probability matrix from $\{y, \mu\}$                    ▷ by Equation 3.3
11:     if $i \% 3 == 0$ then
12:         $P \leftarrow$ target matrix from $Q$                              ▷ by Equation 3.4
13:     $L_{cl} \leftarrow$ clustering loss from $\{P, Q\}$                    ▷ by Equation 3.5
14:     $L_{cong} \leftarrow$ congestion loss from $\{Q, H\}$                  ▷ by Equation 3.6
15:     $L_{timing} \leftarrow$ timing loss from $\{Q, A_{critic}\}$           ▷ by Equation 3.7
16:     $L_{power} \leftarrow$ power loss from $\{Q, S\}$                      ▷ by Equation 3.8
17:     $L = L_{sim} + \lambda_1 L_{cl} + \lambda_2 L_{cong} + \lambda_3 L_{timing} + \lambda_4 L_{power}$
18:     $W, \mu \leftarrow Adam\,(L, \beta_1, \beta_2, \alpha; W, \mu)$        ▷ update GNN, centroids
19: $Z \leftarrow$ get $\mathrm{argmax}$ of $Q$ by row                         ▷ final clustering assignments
```

Then, in Lines 5–6, based on the pre-trained embeddings, we obtain the initial clustering centers $\{\mu\}$ (i.e., centroids) in high dimensions using the K-means algorithm [34] and make these centroids trainable by adding them to the ML computational graph. Note that the K-means algorithm is only conducted once and for all to obtain the initial clusters. In Lines 7–19, we compute each objective function as described in the above equations and jointly optimize them using gradient descent. It is worth to mention that in Lines 11–12, we update the target matrix P once in every three iterations to stabilize the convergence. Finally, the computed gradients are taken to update the parameters in the underlying ML computational graph including the GNN weight matrices $\{W\}$ and the center locations $\{\mu\}$. The entire training process is unsupervised and takes less than 30 minutes on designs with millions of cells by using a single Nvidia Tesla A100 GPU, which introduces almost no runtime overhead to the entire flow.

3.3.6 Integration with Commercial EDA Tools

Now, we illustrate how to leverage the obtained clustering assignments to improve the placement optimization of modern placers. For *Cadence Innovus* [70], the command to create a clustering constraint is: *createInstGroup* $\{$*cell names*$\}$ *-softguide*. As for *Synopsys ICC2* [71], the corresponding command is: *create_placement_attraction* $\{$*cell names*$\}$. During the placement optimization phase, both tools will spend effort in grouping cells in the same cluster closer to each other. Note that for each cluster suggested by the proposed framework, we will create a clustering constraint with one of the above commands depending on which tool we are using. Due to proprietary issues, we unfortunately cannot disclose the underlying tool that is being used in the experimental results. But, we can confidently mention that the proposed framework works for both tools.

3.3.7 Integration with DREAMPlace [24]

DREAMPlace is a recent open-source placer that has brought a huge revolution to academic research thanks to its fast placement computation enabled by GPU acceleration. Here, we demonstrate how to integrate the proposed framework with DREAMPlace to improve the final achieved PPA metrics. The key idea is that with the clustering results obtained, for each cluster, we can create a sub-netlist which is optimized jointly with the full-netlist (i.e., whole design) during the gradient descent update. For example, assume at an iteration a cell has a location (x, y), and with the original full-netlist update (i.e., default update in DREAMPlace), it is assigned a gradient $(\Delta_{full}\ x, \Delta_{full}\ y)$. Now, if the cell is in a cluster c, we will compute another gradient $(\Delta_c\ x, \Delta_c\ y)$ by minimizing the wirelegnth of the sub-netlist that is created by other cells in the same cluster c. Note that both full-netlist and sub-netlist updates leverage the same wirelength objective function. Hence, with the integration of the proposed framework, the final gradient of the cell becomes the sum of the two gradients as: $(\Delta_{full}\, x + \Delta_c\, x, \Delta_{full}\, y + \Delta_c\, y)$. Finally, by multiplying this gradient with the learning rate, we obtain the final amount and direction of the location change for this cell.

3.3.8 Integration with Prediction Models

Recently, many learning-driven supervised models have been proposed to predict critical QoR metrics to improve chip design productivity by shortening turn-around time [1]. However, such models lack the ability to perform QoR optimization. Even if they are able to inference on unseen designs, the achieved results are at most "tool-accirate", which means they can never be better than the ones achieved by default tool flows. To overcome this issue, in this chapter, we present the use case of combining the proposed optimization framework PPA-GNN with predictive models, where we specifically focus on timing prediction models. Particularly, we first train a binary classification model based on the feature set shown in Table 9.1 to predict whether a cell will result in a negative slack value after placement optimization. Then, with the pre-trained timing classification model, we introduce an ad-hoc similarity loss L_{pred} on the predicted timing violating cells as:

$$L_{sim} = - \sum_{v' \in V_{pred}} \|y_{v'} - \mu_{pred}\|^2, \tag{3.11}$$

where $\mu_{pred} = \frac{1}{|V_{pred}|} \sum_{v' \in V_{pred}} y_{v'}$ denotes the centroid (after graph learning) of the predicted instances. Essentially, Equation 3.11 encourages the predicted timing violating cells to have similar node representations after graph learning, which greatly increases the probability of them being assigned to the same cluster.

Table 3.2: Our three academic benchmarks and their attributes under a foundry $28nm$ technology (attributes of six commercial benchmarks are described in section 7.4 due to proprietary.)

Design	# Cells	# Flip-Flops	# Nets
B19	34290	3420	36625
DMA	10451	2062	11304
DES	49853	8802	51247
VGA	57228	17054	60350

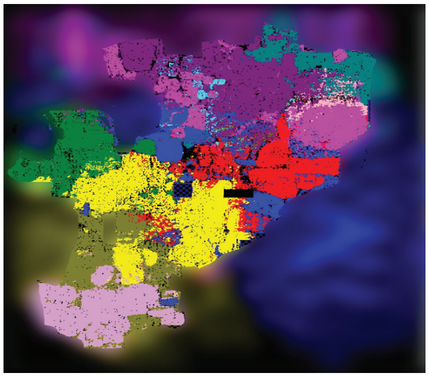


Figure 3.7: Unsupervised PPA-directed clustering results on block1 at a commercial $5nm$ technology. The figure is intentionally blurred due to proprietary.

3.4 Experimental Results

We validate the proposed framework PPA-GNN on 6 industrial GPU/CPU designs in a commercial $5nm$ technology node and 4 OpenCore designs at a foundry $28nm$ technology. Among the industrial designs, the total number of cells ranges from $1.3M$ to $1.6M$, the number of flops ranges from $95k$ to $150k$, and the number of macros ranging from 20 to 150. Due to proprietary issues we cannot disclose the exact attributes of each design. As for the OpenCore designs, their detailed attributes are presented in Table 9.2. PPA-GNN is implemented with the *PyTorch* library. As aforementioned, the entire training process (Algorithm Algorithm 5) is conducted on a single $NVIDIA\ TESLA\ V100$ GPU with $32GB$ memory. For each of the benchmark, the runtime of our framework only takes less than *30 minutes*, which is negligible compared with the runtime of the entire design flow.

Table 3.3: Detailed PPA comparison between the default tool flow and the enhanced flow. We normalize wirelength and power values due to proprietary issues. All designs have ultra-high frequency targets. (since block6 does not have a complete SAIF file for power simulation, we neglect the power comparison.)

design	default industrial PD flow (no clustering)								default + unsupervised clustering (ours)							
#cells	PD	wns	TNS	#	total	clock	total	clock	PD	wns	TNS	#	total	clock	total	clock
#clusters	stage	(ns)	(ns)	vios	WL	WL	power	power	stage	(ns)	(ns)	vios	WL	WL	power	power
block1 $\lvert C\rvert = 10$	post-place	-0.048	-41.45	3306	1	-	1	-	post-place	-0.039	-14.18	1622	0.9997	-	1.000	-
	post-cts	-0.062	-81.31	5890	1	1	1	1	post-cts	-0.039	-61.45	6049	0.9998	0.994	1.004	0.993
	post-route	-0.088	-2.89	574	1	1	1	1	post-route	-0.009	-0.32 (-89%)	149	1.0002	0.994	1.001	0.994
block2 $\lvert C\rvert = 11$	post-place	-0.013	-0.084	23	1	-	1	-	post-place	-0.004	-0.007	2	1.0002	-	1.001	-
	post-cts	-0.005	-0.42	276	1	1	1	1	post-cts	-0.006	-0.43	288	0.9997	0.998	1.000	1.001
	post-route	-0.029	-3.82	1440	1	1	1	1	post-route	-0.022	-3.07 (-19.6%)	1288	0.9988	0.999	0.999	0.989
block3 $\lvert C\rvert = 11$	post-place	-0.004	-0.019	10	1	-	1	-	post-place	-0.001	-0.001	1	0.9989	-	0.997	-
	post-cts	-0.012	-0.461	326	1	1	1	1	post-cts	-0.006	-0.494	349	0.9986	0.979	0.998	0.997
	post-route	-0.020	-2.24	996	1	1	1	1	post-route	-0.015	-1.62 (-28%)	823	0.9967	0.978	0.996	0.998
block4 $\lvert C\rvert = 10$	post-place	-0.005	-0.042	16	1	-	1	-	post-place	-0.002	-0.007	4	0.9778	-	0.992	-
	post-cts	-0.007	-0.421	286	1	1	1	1	post-cts	-0.007	-0.483	305	0.9771	0.986	0.994	0.993
	post-route	-0.022	-2.65	1221	1	1	1	1	post-route	-0.012	-2.16 (-18%)	1103	0.9767	0.985	0.991	0.986
block5 $\lvert C\rvert = 13$	post-place	-0.021	-6.54	1208	1	-	1	-	post-place	-0.021	-5.15	1227	1.0010	-	0.998	-
	post-cts	-0.034	-19.47	2573	1	1	1	1	post-cts	-0.029	-13.77	1978	0.9999	0.993	0.994	0.959
	post-route	-0.010	-0.12	66	1	1	1	1	post-route	-0.008	-0.14	71	0.9995	0.9935	0.999	0.963
block6 $\lvert C\rvert = 8$	post-place	-0.065	-4.853	348	1	-	-	-	post-place	-0.073	-6.06	374	0.9999	-	-	-
	post-cts	-0.152	-22.01	1441	1	1	-	-	post-cts	-0.121	-19.29	1333	1.0007	0.992	-	-
	post-route	-0.133	-40.63	1167	1	1	-	-	post-route	-0.107	-15.85 (-61%)	413	0.9987	0.992	-	-

Table 3.4: Optimization results analysis using proposed PPA losses. Colored entries denote better evaluations.

design	congestion entropy		path cut-size		power entropy	
	default	ours	default	ours	default	ours
block1	1.901	1.902	908.6	905.5	1.940	1.937
block2	2.356	2.360	629.4	603.7	2.142	2.139
block3	2.289	2.291	864.1	835.3	2.106	2.104
block4	2.014	2.015	789.4	784.5	1.995	1.988
block5	2.471	2.469	892.5	864.2	2.485	2.483
block6	1.844	1.848	1113.2	1028.7	-	-

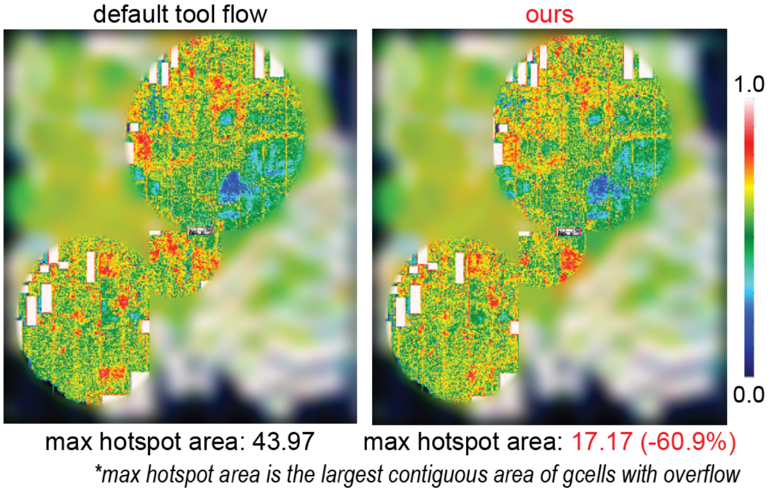


Figure 3.8: Impact on congestion after placement. Our optimization technique reduces the worst congestion by **60.9%**.The worst congestion is defined by the commercial tool to be the largest contiguous overflowing area.

3.4.1 Optimization Results on Industrial Flows

Table 4.2 shows the detailed PPA impact of the proposed framework being integrated with an industrial design flow. We observe that across all the benchmarks, our PPA-directed deep graph clustering technique demonstrates consistent post-route PPA improvements. On the benchmark "block1", it significantly improves the post-route total negative slack (TNS) by 88%, and reduces the worst negative slack (WNS) from $88ps$ to $9ps$. Another trend worth to mention is that the proposed framework consistently improves the clock wirelength and the clock power across all designs. We believe these clock-related improvements are resulted from the "skip-connection" technique proposed in Figure 3.4 that introduces GNN message

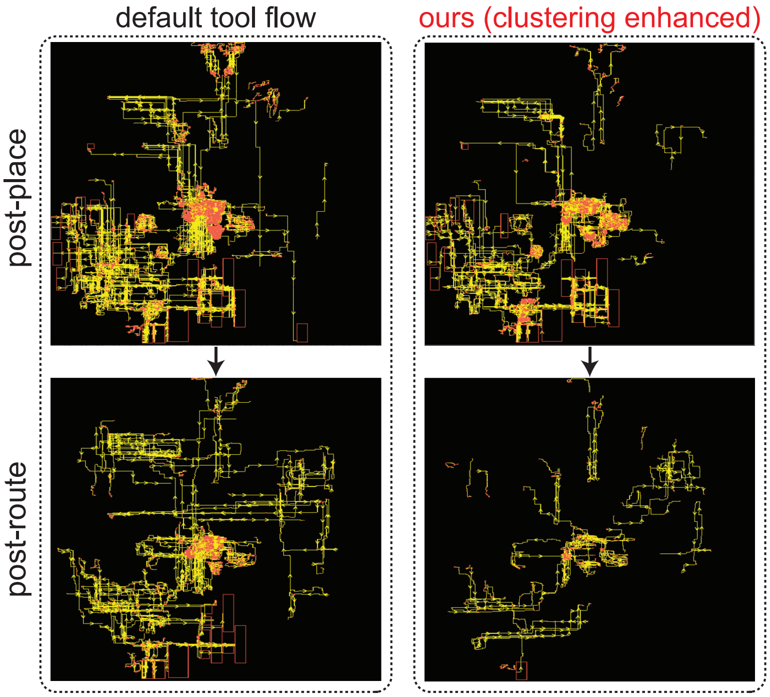


Figure 3.9: Illustration of PPA-GNN impact on timing critical paths from post-place to post-route. Critical wires are in yellow and cells are in red.

passing edges between launch flops and capture flops. In the table, we particularly highlight the improvements at the post-route stage in red as we believe the end-of-flow PPA values are the most accurate metrics to evaluate any PD optimization technique. Finally, to determine the total number of clusters $|C|$ that each benchmark is clustered into, we sweep around the integers between 7 and 14 (inclusive) to find the number that achieves the minimum value of Equation 7.1 (i.e., the weighted sum of each objective). In a latter sub-section we will show that the proposed PPA-inspired ML loss functions can be directly utilized for placement evaluation.

With the detailed optimization results presented, we now take “block1” as a case study to analyze and demonstrate the detailed PPA impact of our framework. Figure 3.7 visualizes its clustering result, where each color represents a cluster and is taken as a soft constraint to the underlying placement engine. Figure 3.8 demonstrates the congestion impact of the proposed framework, where we observe that the worst congested area is greatly reduced by 60.9%. We believe this improvement proves the effectiveness of our congestion loss function (Equation Equation 3.6). Figure 3.9 further shows the impact on timing critical paths from the post-place stage to the post-route stage, where we observe that the proposed technique helps to reduce the number of violating paths significantly.

Analysis on Optimization Results using PPA Losses.

Throughout the past several decades, half-perimeter wirelength (HPWL) and overflow have been the two dominant metrics for placement evaluation. However, these two popular metrics do not show good correlation with post-route PPA values in advanced technology nodes. Take the HPWL metric for example, in Table 4.2, we see that with our framework, both block2 and block5 have worse placement wirelength estimations compared with the default flow, however, at the post-route stage, our framework achieves better PPA metrics including the wirelength itself. This shows the inaccuracy of using HPWL as the evaluation metric.

Here, we show that our PPA loss functions can deliver more accurate placement evaluation than the renowned HPWL metric. The key idea is that given two final placements from the same initial placement, we want to discern which one is better for the subsequent PD stages using the proposed PPA losses with the clustering assignments obtained from the initial placement. To achieve this, we leverage Equation 3.3 to transform the final 2D distances between cells and centroids into probabilities to compute the PPA losses, where the centroid of each cluster is the center of gravity of the cells within. The evaluation results are shown in Table 3.4. We observe that our PPA loss functions deliver accurate placement evaluations and correlate well with the post-route PPA metrics.

3.4.2 Optimization Results on DREAMPlace

In this experiment, we integrate the proposed framework with DREAMPlace [24] and compare the PPA metrics between the flows using the default DREAMPlace placement and our enhanced DREAMPlace placement. Since in this work we are focusing on full-flow PPA

Table 3.5: DREAMPlace [24] integration results on block1. "DREAM" denotes the DREAMPlace default placement.

PD	post-place		post-cts		post-route	
stage	DREAM	ours	DREAM	ours	DREAM	ours
WNS (ns)	-0.107	-0.062	-0.081	-0.066	-0.065	-0.047 (-27.7%)
TNS (ns)	-74.96	-74.50	-145.73	-98.62	-25.05	-12.92 (-48.4%)
# vios	4203	3637	7312	6372	1908	1461 (-23.4%)
WL	1	0.952	1	0.953	1	0.954
power	1	0.979	1	0.981	1	0.979

Table 3.6: Comparison between proposed PPA-directed clustering framework and a single-way clustering work [7]. The number of clusters $|C|_{[23]}$ used for previous work is obtained from the sweeping experiments they proposed.

PD	post-place		post-cts		post-route	
stage	[7]	ours	[7]	ours	[7]	ours
design: block1 ($> 1.3M$ cells), $\|C\|_{[23]} = 13$, $\|C\|_{ours} = 10$						
WNS (ns)	-0.044	-0.039	-0.048	-0.039	-0.056	-0.009 (-83.9%)
TNS (ns)	-35.35	-14.18	-67.71	-61.45	-2.965	-0.032 (-98.9%)
# vios	3119	1622	6411	6049	867	149 (-82.8%)
WL	1	0.999	1	0.998	1	0.999
design: block6 ($> 1.3M$ cells), $\|C\|_{[23]} = 10$, $\|C\|_{ours} = 8$						
WNS (ns)	-0.058	-0.073	-0.193	-0.121	-0.115	-0.107 (-6.9%)
TNS (ns)	-6.42	-6.06	-23.35	-19.29	-27.37	-15.85 (-42.1%)
# vios	479	374	1608	1333	1118	413 (-63.1%)
WL	1	0.993	1	0.981	1	0.992

improvements, the comparison is done by first leveraging DREAMPlace to generate two initial placements (default and ours), and then leveraging a commercial tool to perform placement optimization and the subsequent PD implementations. Note that the integration happens within DREAMPlace. We modify the source code based on the procedures presented in subsection 3.3.7. The full-flow comparison results are presented in Table 3.5. We observe that the proposed framework helps to improve every major PPA metric across all the stages significantly.

3.4.3 Comparisons with Single-Way Clustering [7]

One of the highlights of this work is that we directly formulate crucial PPA metrics as ML loss functions, and leverage them to guide both node representation learning and clustering

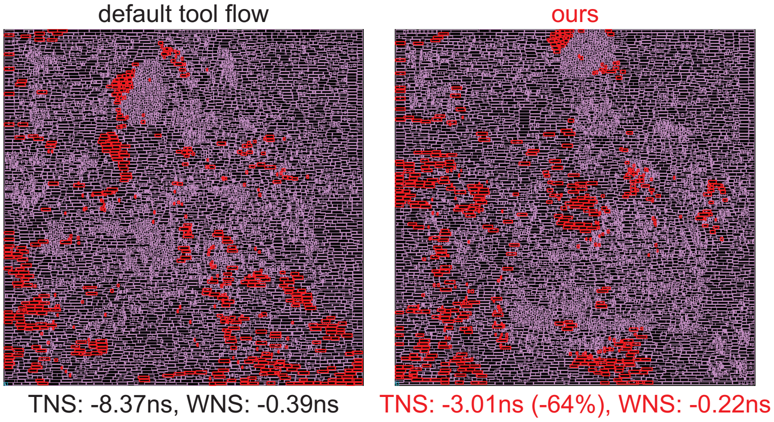


Figure 3.10: Illustration of attraction impact on timing violating cells with the DMA benchmark. The left figure shows the distribution of the cells whose output pins have negative slack values in the default tool flow (i.e., without attraction). The right figure shows that with the clustering constraints generated by PPA-GNN, the initially violating cells aggregate closer.

process in an end-to-end, mutually-reinforcing manner. To demonstrate the effectiveness of our approach, in this experiment, we perform head-to-head comparisons with the previous work [7] that requires a two-step process to determine the clustering assignments (the difference between our work and the previous work is illustrated in Figure 4.2). Table 3.6 demonstrates the comparison results. We observe that our framework achieves significantly better optimization results in every major PPA metric across all the stages, which clearly demonstrates the benefits of our PPA-directed clustering approach. We believe the improvements come from the fact that compared with the previous work [7], not only can our GNN module perform better feature transformation with the PPA feedback from the clustering assignments, but also can our clustering module finds better assignments based on the improved node representations.

3.4.4 Optimization Results with Timing Prediction Models

As described in subsection 3.3.8, in this chapter, we present the use case of integrating the proposed optimization framework with a classification-based timing prediction model that generates a binary decision for each instance on whether it will be timing violating after the placement optimization of the default tool flow. In this experiment, we use the

GNN modeling architecture proposed in [12] to realize the timing classification model, and to validate the integration in a meaningful manner, we perform the experiment in a cross validation setting. That is, for each OpenCore benchmark shown in Table 9.2, we only use the other three benchmarks to train the timing prediction model while remaining it unseen for validation. By introducing the ad-hoc similarity loss function shown in Equation 3.11 in the PPA-GNN training cycle, we obtain the optimization results shown in Figure 3.10, where we clearly observe that the initially timing violating cells in the default tool flow (i.e., without using PPA-GNN) has a more scattered distribution compared with the one showing in the right of the figure, and the achieved TNS value is improved by 64%.

3.4.5 Why Does PPA-GNN Work?

In the experiments (Table 4.2 and Table 3.6), we demonstrate that the proposed placement optimization framework improves the full-chip PPA quality of the default tool flow and a previous work [7] significantly on industrial designs with millions of instances in a commercial $5nm$ technology node. We believe the good optimization results of our framework can be accounted by the following reasons:

Global and Systematic Optimization

We think the first reason comes from the fact that the proposed ML-based placement optimization technique improves design PPA metrics more globally and systematically compared with the existing heuristic algorithms in commercial tools that often perform the optimization in a local and ad-hoc manner (i.e., path-by-path, cone-by-cone, or subgraph-by-subgraph) because of their inabilities to deal with large design complexity. Unlike these heuristic algorithms, our framework optimizes the entire netlist graph as a complete entity, where the PPA-related objectives are calculated across every cell in the design. Therefore, it has the ability to capture the complicated interactions among instances that are distant to each other.

PPA-Directed End-to-End Optimization

Unlike previous graph unsupervised learning works [7, 6] that leverages GNNs to obtain node embeddings without any guidance from the subsequent clustering task, in this work, we design PPA-inspired ML loss functions related to congestion, timing, and power to optimize both node representation learning and clustering assignments in an end-to-end manner. In contrary to previous works that improve PPA metrics indirectly, our framework works as a stand-alone optimizer that directly improves the PPA metrics using ML algorithms. The direct benefits of our approach is demonstrated in Table 3.6.

Strong Correlation with Post-Route Metrics

Table 3.4 shows that the proposed PPA loss functions can be directly utilized as placement evaluation metrics which correlate well with the post-route PPA results (Table 4.2). This means that by minimizing our PPA-inspired ML loss functions, we are improving the underlying placement in the same direction as improving post-route PPA metrics, which drives the optimization in an accurate manner.

3.5 Conclusion

In this chapter, we have presented the first placement optimization framework named PPA-GNN that directly formulates PPA metrics as ML loss functions and optimizes them to discover the cell clusters that can improve the underlying placement and end-of-flow PPA metrics. We perform detailed and thorough validation of the proposed framework with different integrations. First, we show that PPA-GNN can be seamlessly integrated with industry-leading commercial tools in an industrial design implementation flow to immediately improve key PPA metrics after placement optimization. More importantly, we demonstrate that these improvements last firmly to the post-route stage. Second, we show that PPA-GNN can be integrated with a renowned academic placer, DREAMPlace, to significantly improve the achieved end-of-flow QoR metrics. Finally, we demonstrate that the proposed framework can be elegantly integrated with predictive models so as to enable targeted optimization on the predicted metrics, where we use timing as an example. We think this work shall demonstrate that ML algorithms can not only be utilized to solve the prediction or simulation tasks in EDA, but can also be leveraged as stand-alone algorithms that directly optimize PPA metrics.

CHAPTER 4
BRIDGING OPEN-SOURCE AND COMMERCIAL PLACERS USING GENERATIVE ADVERSARIAL NETWORKS AND TRANSFER LEARNING

4.1 Background and Motivation

DREAMPlace [**lin2019dreamplace**] is a renowned open-source placer that provides GPU-acceleratable infrastructure for placements of Very-Large-Scale-Integration (VLSI) circuits. It achieves orders of magnitude speed up over RePlace [54], the state-of-the art academic placer, by converting its placement objectives into costumed CUDA kernels using a deep learning toolkit PyTorch [72]. However, as RePlace, vanilla DREAMPlace only has a limited objective focus on wirelength and density. Particularly, it cannot consume many other essential constraints (e.g., timing, power) as commercial Physical Design (PD) tools, leading to inferior placement quality. This motivates us to ask the following question: Is there any way to advance DREAMPlace towards commercial-quality without knowing the secret sauces of those black-boxed commercial engines? In this chapter, we prove that the answer is yes. Particularly, via generative adversarial learning.

In this work, we present the first-ever learning-driven placement optimization framework named DREAM-GAN that directly improves DREAMPlace using generative adversarial learning. The key idea is that although we do not know the underlying algorithms or constraints used by the tools, we can “quantify placement similarity” between DREAMPlace-generated placements and tool-optimized placements using generative learning, and by optimizing the differentiable similarity scores computed from a discriminator, we can narrow the distribution gap between DREAMPlace and commercial tools. This is a simple, yet highly effective approach, which is greatly motivated by the success of Generative Adversarial Networks (GANs) [21] in real-world applications, where signals in

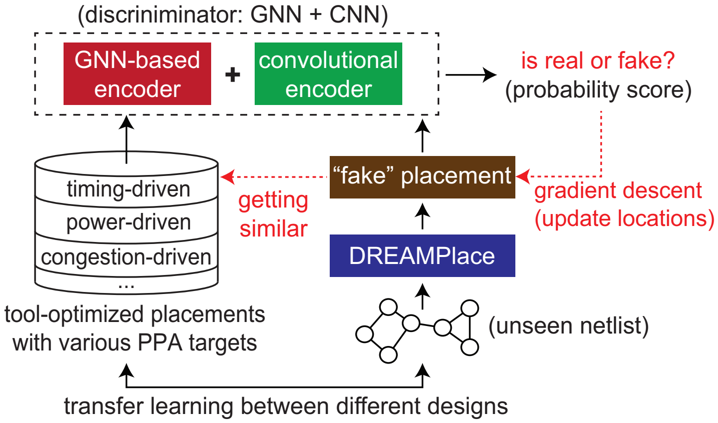


Figure 4.1: High-level overview of DREAM-GAN that performs placement optimization using generative adversarial learning. Note that DREAM-GAN facilitates transfer learning between different designs.

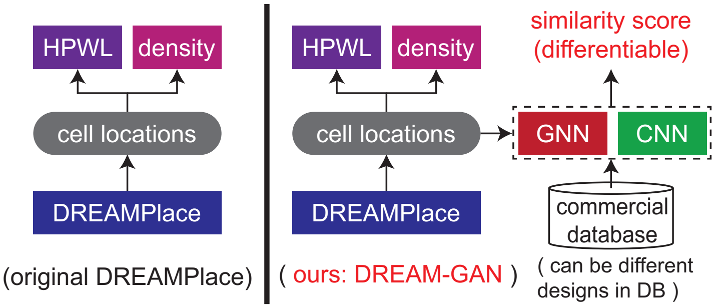


Figure 4.2: Difference in objectives between DREAMPlace [**lin2019dreamplace**] and DREAM-GAN at each placement iteration. The cell locations generated from DREAMPlace are encouraged to follow the ones in commercial database.

different domains (e.g., texts and images) can be converted to each other by parameterizing target distributions using differentiable frameworks.

Figure 4.1 presents a high-level overview of DREAM-GAN, where the vanilla DREAMPlace is considered as a "generator" whose goal is to generate the placements that follow similar distributions as the tool-optimized ones in the databases (which are optimized for different Power, Performance, and Area (PPA) objectives). To achieve this, a discriminator, built upon Convolutional Neural Networks (CNN) and Graph Neural Networks (GNN), is developed to quantify the placement similarity between two types of placement (i.e., DREAMPlace-generated and tool-optimized). The goal of the discriminator is to make correct judgements by telling whether its input is coming from commercial databases or

DREAMPlace, whereas one of the objectives of DREAMPlace (in our DREAM-GAN settings), is to “fool” the discriminator by generating similar (or realistic) placements as the ones in the database.

Figure 4.2 shows the key objective difference between the proposed framework DREAM-GAN and the original DREAMPlace. Aside from optimizing the wirelength (WL) and density objectives as in the vanilla DREAMPlace, in each placement iteration, we further compute a differentiable similarity loss using GNN-based and CNN-based discriminators. Note that the in our generative settings, the input design of DREAMPlace and the designs in the commercial database are not necessarily the same. That is, DREAM-GAN facilitates transfer learning to perform placement optimization on unseen netlists, which is the greatest strength of the proposed method.

The goal of this work is to demonstrate that the placement quality (or style) of a placer can be transferred onto another through generative adversarial learning without knowing its algorithms. Particularly, we show that, DREAMPlace, as a differentiable placement engine, can be elegantly used as a generator that generates tool-alike placements of the underlying gate-level input netlists.

4.2 Related Work

Analytical placers [54, 73] have brought tremendous success to the EDA industry in the last decade. However, modern VLSI designs easily consist of millions of instances that are required to be placed on constrained layouts. Existing CPU-intensive analytical placers are struggling to meet this demand in a reasonable amount of runtime.

To overcome the runtime barrier, DREAMPlace [**lin2019dreamplace**] and Xplace [74] leveraged GPUs to significantly accelerate the runtime of those traditional analytical placers without degrading the Quality-of-Results (QoR), which is benefited from the recent advancement in open-source deep learning infrastructures such as Pytorch [72]. Nonetheless, as aforementioned, the placement solutions these open-source placers achieved are

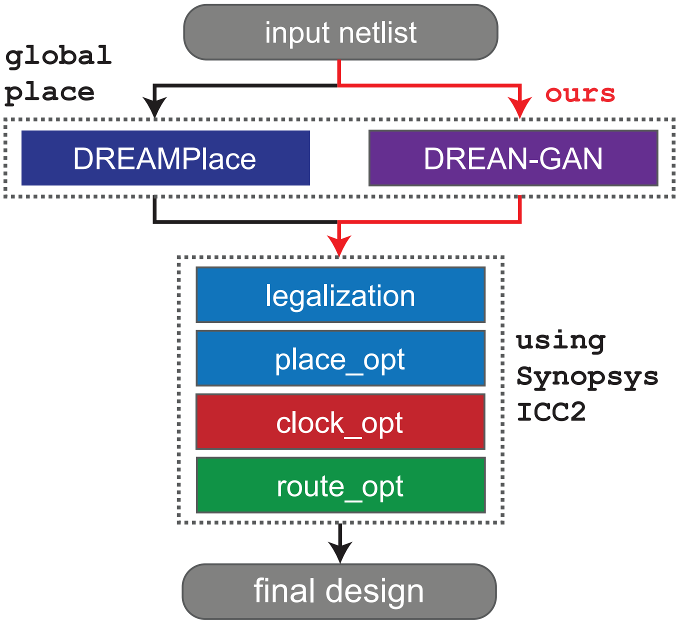


Figure 4.3: Illustration of integrating open-source placers into an industrial design flow using Synopsys ICC2, where the global placement stage is replaced with proceeding stages remain the same.

still not comparable to commercial tools'.

Beyond runtime improvement, the development of ML Theory and its applications have enabled placement quality optimization using learning-driven frameworks. The authors of [8] proposed an unsupervised graph clustering framework equipped with PPA-inspired ML loss functions to improve commercial tool placement quality by generating cell clustering constraints as placement guidance. For macro placement, the authors of [9] presented a seminal work of using Reinforcement Learning (RL) to determine macro locations without human in the loop, achieving superhuman results.

In this work, we aim to combine the best of both worlds: the runtime improvements from GPU-based placers, and the QoR improvements from learning-driven techniques. Particularly, we extend the original DREAMPlace framework using generative adversarial learning to optimize its solution quality. The use of GAN-based approaches in th realm of PD is still in its early stages, which is mainly because it requires creative problem formulation and thinking. Previous work [75] has demonstrated that GAN can be used for commercial tool clock tree parameters prediction and optimization, where pa-

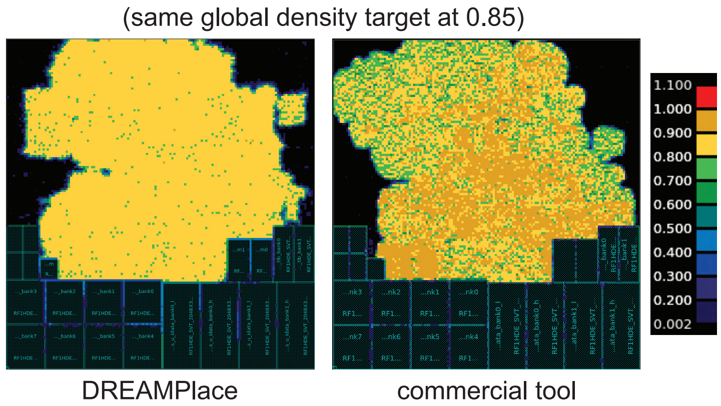


Figure 4.4: Comparison of cell density maps between DREAMPlace and Synopsys ICC2 under the same global placement density target. It is observed that the commercial tool has extra intelligence on where to locally aggregate or spread out cells in order to optimize crucial PPA metrics while satisfying the global density constraints.

rameters are considered as signals to be optimized, and a generator is developed to optimize parameters from input random noise. Motivated by [75], in this chapter, we proposed DREAM-GAN, which transfers the placement quality of commercial placers onto DREAMPlace [**lin2019dreamplace**] effortlessly. We demonstrate that fast and optimized VLSI placement can indeed be achieved.

Finally, in this chapter, we not only show that DREAM-GAN immediately improves the placement quality of DREAMPlace after global placement, but also demonstrate that the improvements last firmly to the post-route stage. Figure 4.3 shows the integration of the proposed framework DREAM-GAN with an industry-leading commercial PD tool, *Synopsys ICC2*. Experimental results demonstrate that DREAM-GAN improves vanilla DREAMPlace by up to 8.3% in wirelength, 7.4% in power on a commercial CPU design at the post-route stage.

4.3 Overview and Motivation

It is widely acknowledged that generative adversarial learning is a promising paradigm that effectively captures complicated distributions using generative and discriminative models which have “opposite” objectives. Generally speaking, the goal of the generator is to generate target distribution alike data from random (or non-meaningful) distributions, while the goal of the discriminator is to distinguish the source of its inputs (i.e., from the generator or from the target distribution). Note that both generator and discriminator can be realized by any differentiable system (i.e., not necessarily neural networks). As aforementioned, in this chapter we consider DREAMPlace, a differentiable placement system, as a generator whose goal is to generate placements that follow similar distributions as the ones in a tool-optimized database.

Our discriminator leverages CNNs and GNNs to determine the origin of its input source that alternates in each iteration of GAN training, where CNNs are responsible to encode cell bin-density maps, and GNNs are responsible to encode netlist connectivity. The rationale behind using GNNs for netlist encoding is that netlists are essentially hypergraphs whose node connectivity is critical to placers. The motivation behind using CNNs for bin-density map encoding is shown in Figure 4.4. We observe that under the same global density target, the commercial tool has extra intelligence on locally aggregating or loosening cells in order to improve design PPA metrics globally, where DREAMPlace naively strives to make every local bin to have the same local density target as the global density target. This density variation is proven to be critical to the success of placement optimization in [8]. The observation shown in Figure 4.4 strongly motivates us to leverage bin-density map as one the indicators of placement similarity, and CNNs thus become the second-to-none choice to perform encoding on it as they are well-known for grid signals (e.g., images) classification.

Finally, it should be mentioned that many placement metrics can be used for evaluations of placement similarity. In this work, we empirically find out that by using bin-density maps and GNN netlist embeddings as indicators, DREAM-GAN can already effectively advance DREAMPlace towards commercial quality. Furthermore, the beauty behind our approach is that our indicators can provide similarity scores for netlists in different sizes. Hence, the design that DREAMPlace is optimizing does not have be inside the database.

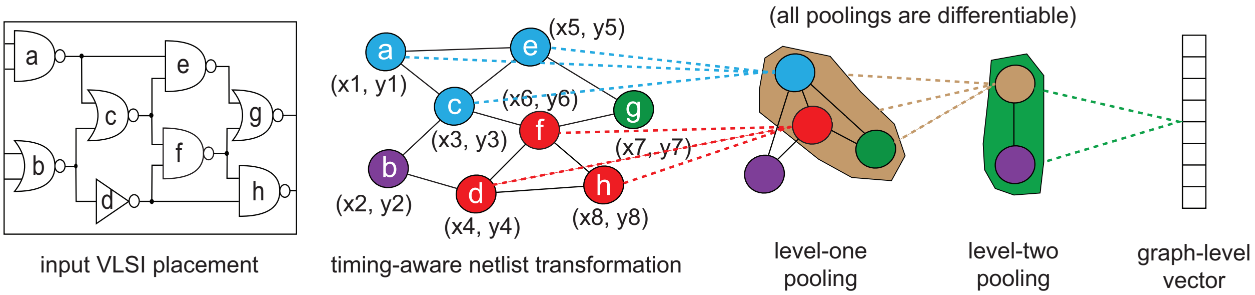


Figure 4.5: Illustration of GNN netlist transformation and graph pooling. First, following from [8], we perform timing-aware netlist transformation by only taking timing arcs as GNN message passing edges. Then, we perform attention graph pooling to construct the final graph-level vector that characterizes the input placement. Note that the poolings are differentiable (i.e, learnable), meaning that the clustering results (e.g., {a, c, e} is now a cluster) will change across training iterations. The obtained graph-level vector is fed to downstream networks to quantify placement similarity.

DREAM-GAN is able to optimize unseen designs.

4.3.1 Database Construction

A high-quality placer is helpful to demonstrate the proposed placement optimization technique using generative adversarial learning. In this chapter, we take *Synopsys ICC2*, an industry-leading commercial tool, as our reference tool for database construction. Particularly, we sweep around essential placement parameters offered by *ICC2* as shown in Table 4.1. The combinations of these parameters form a high-dimensional space, leading to a variety of placements that have distinct PPA focus, including performance-driven placements, low-power placements, routability-driven placements, etc. Nonetheless, we want to emphasize that our framework is not limited to any objective focus. DREAM-GAN can be equipped with any database to transfer the placement quality (or style) within onto DREAMPlace. More importantly, the designs in the database and the design that DREAMPlace is optimizing do not have to be the same. DREAM-GAN facilitates transfer learning for placement optimization.

Table 4.1: Parameters we leverage for database generation.

ICC2 parameters	type (values)	description
set_qor_strategy	enum (3)	set optimization priority
low_power_effort	enum (4)	effort in low power optimization
congestion_effort	enum (3)	effort in congestion optimization
is_timing_driven	bool (2)	is timing-driven placement
is_power_driven	bool (2)	is power-driven placement
buffer_aware	bool (2)	buffering of high-fanout nets
coarse_density	float ([0.7,0.9])	density of global placement
target_route_density	float ([0.7,0.9])	density of early global routing

4.4 DREAM-GAN Algorithms

The key idea of DREAM-GAN is to transfer the placement quality (or more precisely, style) of one placer onto another[1] through generative adversarial learning. To achieve this, a discriminator built upon GNNs and CNNs is developed to quantify similarity of placements between different placers. Since the similarity scores have to be differentiable to update the underlying cell locations (so as to improve DREAMPlace), a differentiable graph pooling methodology is adopted, and a differentiable bin-density map transformation technique named Soft-Bin is proposed.

4.4.1 GNN-based Discriminator

Graph representation learning conducted by GNNs is an effective technique to encode netlist connectivity and node attributes into a meaningful representations. In this chapter, we leverage GNNs to perform graph-level encoding on the netlist graph $G = (V, E)$ of a given placement, where (x, y) locations of each cell are taken as node attributes. The

[1] In this chapter, we show the optimization results on DREAMPlace, but DREAM-GAN can work on any differentiable placement infrastructure.

goal of our GNN module is to obtain a graph-level vector g that is representative of the underlying netlist G. This graph-level vector is further taken as the input of the downstream network to quantify placement similarity.

Figure 4.5 illustrates the graph learning process of the proposed DREAM-GAN framework. Starting from an input placement, we first perform timing-aware netlist transformation as in [8]. The transformation only preserves timing arcs as GNN message passing edges. Then, following from GraphSAGE [37], an inductive based message passing process is leveraged to transform the initial features of each node into better representations through neighborhood aggregation as:

$$\begin{gathered} h_{Neigh(v)}^{k-1} = reduce\_mean\left(\{\mathbf{W}_k^{agg} h_u^{k-1}, \forall u \in Neigh(v)\}\right), \\ h_v^k = \sigma\left(\mathbf{W}_k^{proj} \cdot \text{concat}\left[h_v^{k-1}, h_{Neigh(v)}^{k-1}\right]\right), \end{gathered} \tag{4.1}$$

where k denotes the transformation level, σ denotes the sigmoid function, $Neigh(v)$ denotes the neighbors of node v, W_k^{agg} and W_k^{proj} denote the aggregation and projection matrices at the k-th layer of the GNN model. Note that Equation 9.1 is repeated for every cell in the design. In the implementation, our GNN has three layers ($K = 3$), and each of them has a 32 hidden dimensions. Hence, the final node embeddings $\{h_v^3, \forall v \in V\}$ has 32 dimensions.

At each level k of the node representation learning, a differentiable graph attention pooling mechanism [76] is applied to coarsen the graph via a soft clustering assignment C_k, where nodes belonging to the same cluster will be merged into a super-node through mean pooling, which can be derived as:

$$H_{k+1} = C_k H_k, \quad A_{k+1} = {A_k}^T C_k A_k, \tag{4.2}$$

where $H_k = \{h_v^k, \forall v \in V\}$, A_k denotes the adjacency matrix at level k, and $C_k \in R^{|V|_{k+1} \text{x} |V|_k}$ denotes the mapping of nodes between level k and level $k+1$, where a node

v may be merged into a super-node or stay as itself. At the end of the entire graph representation learning, a mean pooling is applied across the remaining nodes to obtain the final graph-level vector g, which is taken as one of the indicators of placement similarity. A graphical illustration of the entire graph learning process is shown in Figure 4.5.

Finally, we want to emphasize that both DREAMPlace-generated placements and tool-optimized placements go through the same graph learning process as shown in Figure 4.5. Let D_{gnn} denote the discriminator network parameters that are related to graph representation learning, the graph representation learning objective L_{gnn} can be derived using cross-entropy as:

$$\begin{aligned}\mathcal{L}_{gnn} =& \mathbb{E}_{(G,x,y)\sim database}\left[log(D_{gnn}(G,x,y))\right] \\ &+ \mathbb{E}_{(G,x,y)\sim DREAM}\left[log(1-D_{gnn}(G,x,y))\right].\end{aligned} \tag{4.3}$$

Equation 5.8 reflects the adversarial nature in our DREAM-GAN framework that depending on the sources of input, we train the discriminator to make corresponding correct judgements. With enough training iterations, our GNN module is able to extract the information that tells the key difference between DREAMPlace-generated placement and the tool-optimized ones.

4.4.2 Soft-Bin: Differentiable 2D Bin-Density Map Transformation

So far, we have introduced how DREAM-GAN encodes netlist connectivity using GNNs by leveraging a differentiable attention pooling mechanism with cell locations as initial node features. Now, we present the details of how DREAM-GAN leverages bin-density maps to characterize and differentiate different placements with CNNs, where the key motivation behind is illustrated in Figure 4.4.

It should first be mentioned that most common bin-density calculation method using the simple formula of dividing the total cell area in a bin by the total bin area is not differentiable, as the act of assigning a cell to a particular bin is deterministic. Although such computation is “exact” and accurate, a probabilistic calculation method is needed in order

Algorithm 6 Soft-Bin Transformation.

```
Input: $G = (V, E)$: input netlist, $\{X_v, Y_v \; \forall v \in V\}$: cell locations of the input placement, $W$:
    with of floorplan, $H$: height of floorplan, $b\_width$: bin width, $b\_height$: bin height.
Output: $M \in R^{|V|\text{x}|V|}$ : differentiable 2D bin-density map.
 1: $M[*][*] \leftarrow 0$                                   ▷ initialize M to 0
 2: $bin\_area = b\_width * b\_height$
 3: $num\_w \leftarrow floor(\frac{W}{b\_width})$
 4: $num\_h \leftarrow floor(\frac{H}{b\_height})$
 5: for $i = 0$; $i < num\_w$; ++$i$ do
 6:     for $j = 0$; $j < num\_h$; ++$j$ do
 7:         $V' \leftarrow filter\{i * num\_w \leq X_v < (i+1) * num\_w \forall v \in V\}$
 8:         $V' \leftarrow filter\{j * num\_h \leq Y_v < (j+1) * num\_h \forall v \in V'\}$
 9:         for $v \in V'$ do
10:             $b \leftarrow M[i][j]$
11:             $neigh\_bins \leftarrow$ get adjacent or diagonal bins of $b$
12:             $dist\_vec \leftarrow []$               ▷ distance vector of cell $v$ to each bin
13:             $dist\_vec.push\_back(\|b_x - v_x, b_y - v_y\|^2)$
14:             for $nb \in neigh\_bins$ do
15:                 $dist\_vec.push\_back(\|nb_x - v_x, nb_y - v_y\|^2)$
16:             $prob\_vec \leftarrow softmax\left(dist\_vec^{-1}\right)$
17:             $area\_vec \leftarrow area_v * prob\_vec$           ▷ expected values
18:             update $M$ by $area\_vec$               ▷ add area of each bin to M
19: $M \leftarrow \frac{M}{bin\_area}$               ▷ convert expected area to expected density
```

to compute gradients based on cell (x, y) locations so as to improve the underlying placement through gradient descent. To achieve this, we develop a differentiable bin-density transformation technique named Soft-Bin as shown in Algorithm Algorithm 6.

The key idea behind Algorithm Algorithm 6 is that instead of deterministically assigning each cell to an exact bin purely based on its location, we can probabilistically distribute the area of a cell onto its neighboring bins, which can be achieved by any activation function that maps real values into probabilities. In this chapter, we use softmax for such computation. The beauty of our probabilistic bin-density calculation approach is that it enables the underlying cell locations to be updated along with any operation (i.e., maximization or minimization) of the computed density. That is, with the proposed Soft-Bin technique, cell locations can be directly updated using gradient descent by optimizing the similarity loss computed from the CNN-based discriminator.

The proposed Soft-Bin algorithm works as follows. In Lines 1–4, we initialize the bin

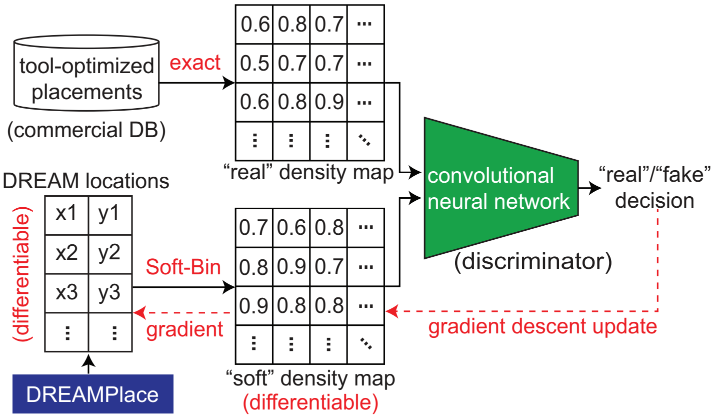


Figure 4.6: Generative adversarial learning conducted by CNNs using the proposed Soft-Bin density map transformation technique. The goal of DREAMPlace is to generate the placements that have similar bin-density maps as the ones in the commercial database.

density map M based on the specifications from inputs. The cells corresponding to each bin can be obtained using Lines 7–8. Now, as shown in Lines 10–15, for each cell that deterministically belongs to a target bin b, we first identify its neighboring bins (adjacent or diagonal) $neigh\_bins$, and then compute a distance vector $dist\_vec$ that denotes the Euclidean distance between the target cell to each bin (including b and $neigh\_bins$). Finally, we transform the distance vector $dist\_vec$ into a probability vector $prob\_vec$ using the softmax function as shown in Line 16, and the "expected" area contribution can be calculated using Line 17. After updating the bin-density map M by the expected area contribution of each cell in the design, we obtain the final density of each bin using Line 19.

4.4.3 CNN-Based Discriminator

With the proposed Soft-Bin technique for differentiable bin-density map transformation, we now describe the adversarial learning conducted by the CNN-based discriminator as shown in Figure 4.6. The rationale behind is to consider bin-density maps as single-channel grid signals where CNN filters can perform 2D convolutions upon. Note that the proposed Soft-Bin technique is only utilized transform DREAMPlace-generated placements into (soft) bin-density maps. The "exact" bin-density map computation as aforementioned is adopted for the transformation of tool-generated placements as they are not required to

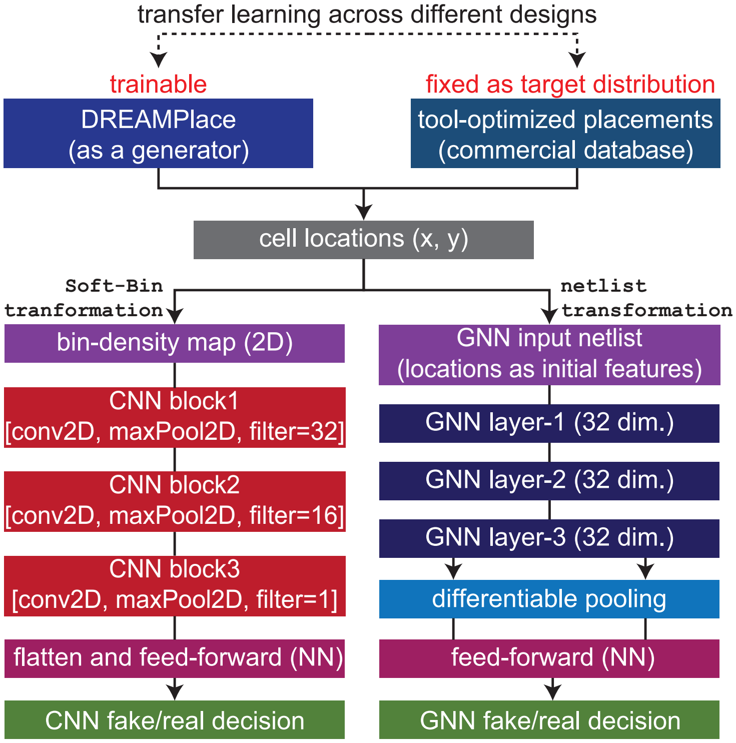

transfer learning across different designs
trainable
fixed as target distribution
DREAMPlace (as a generator)
tool-optimized placements (commercial database)
cell locations (x, y)
Soft-Bin tranformation
netlist transformation
bin-density map (2D)
GNN input netlist (locations as initial features)
CNN block1 [conv2D, maxPool2D, filter=32]
CNN block2 [conv2D, maxPool2D, filter=16]
CNN block3 [conv2D, maxPool2D, filter=1]
flatten and feed-forward (NN)
CNN fake/real decision
GNN layer-1 (32 dim.)
GNN layer-2 (32 dim.)
GNN layer-3 (32 dim.)
differentiable pooling
feed-forward (NN)
GNN fake/real decision

Figure 4.7: Detailed architecture of DREAM-GAN that leverages CNN-based and GNN-based discriminators to quantify placement similarity between different placement sources.

be differentiable. Finally, let D_{cnn} be the network parameters related to bin-density map differentiation, the objective of the CNN-based discriminator L_{cnn} can be derived as:

$$\begin{aligned}\mathcal{L}_{cnn} =& \mathbb{E}_{(G,x,y)\sim database}\left[log(D_{cnn}\left(G, exact\_map(x,y)\right))\right] \\ &+ \mathbb{E}_{(G,x,y)\sim DREAM}\left[log(1 - D_{cnn}\left(G, Soft\text{-}Bin(x,y)\right))\right],\end{aligned} \tag{4.4}$$

where the $exact\_map$ denotes the exact bin-density calculation, which is responsible for placements from commercial databases.

4.4.4 Putting All Together: End-to-End DREAM-GAN Training

Figure 4.7 shows the detailed architecture of the CNN-based and the GNN-based discriminators. Note that placements originating from DREAMPlace do not have to be the same as the ones from the commercial database, since DREAM-GAN does not explicitly take design information such as number of cells or number of nets as features. All the computations involved are “size-independent”, meaning that netlists in different sizes will be encoded to vectors in the same dimensions, which is true for both CNN-based and GNN-based discriminator networks. Finally, at each placement iteration, the similarity objective L_{sim} and the overall objective L of our DREAM-GAN can be formulated as:

$$L = L_{vanilla} + L_{sim}, \qquad\qquad L_{sim} = \lambda_1 L_{gnn} + \lambda_2 L_{cnn} \tag{4.5}$$

where $L_{vanilla}$ denotes the wirelength and density objectives computed from the vanilla DREAMPlace, $\{\lambda\}$ denote the hyper-parameters for loss weighting. In the implementation, we set $\lambda_1 = 1$ and $\lambda_2 = 10$. These values are decided through empirical experiments, where we observe that the CNN-based discriminator plays a more important role in the success of DREAM-GAN. Hence, we weight its objective by a higher number. Finally, the training methodology works as follows. In the first 200 iterations, we only update the cell locations through $L_{vanilla}$ as the placement at this phase is far from valid. Then, for every iteration after 150th, we optimize L_{sim} three times more than $L_{vanilla}$ to advance the underlying placement towards commercial quality.

4.5 Experimental Results

In this chapter, we validate DREAM-GAN with 3 commercial CPU benchmarks and 3 OpenCore benchmarks using the TSMC $28nm$ technology node. The commercial database is generated by randomly sampling the parameters listed in Table 4.1 using *Synopsys ICC2*, where per benchmark, 100 placements are generated with different PPA objectives. In the experiments, we not only demonstrate that DREAM-GAN significantly improves vanilla DREAMPlace in single-design optimization, but also show that it achieves superior optimization results on unseen designs that are not in the database.

Table 4.2: Single-design optimization results. DREAM-GAN optimizes the same design as in the database.

design (# cells)	PD stage	vanilla DREAMPlace [**lin2019dreamplace**]					DREAM-GAN (ours)				
		wns (ns)	TNS (ns)	# vios	total WL (um)	total power (mW)	wns (ns)	TNS (ns)	# vios	total WL (um)	total power (mW)
CPU-1 (220K)	global place	-2.05	-13498	19558	374130	200.1	-1.46	-10601	18425	3546577	193.5
	place opt	-1.74	-6197	13018	4034908	194.7	-1.52	-6024	12697	3870333	179.6
	clock opt	-0.30	-45.89	681	4163129	144.4	-0.24	-34.28	473	4041709	140.1
	route opt	-0.26	-22.4	464	4166459	144.3	-0.18	-21.11	446	4050908 (-2.7%)	141.9 (-1.6%)
CPU-2 (580K)	global place	-432.97	-5634543	48869	12382802	25142.4	-432.98	-5324323	45644	11110278	25098.2
	place opt	-608.91	-7218793	40780	12654907	13244.1	-608.74	-7202230	40544	11493278	12431.0
	clock opt	-0.20	-61.48	1726	17769476	488.1	-0.23	-48.28	1505	16305060	455.0
	route opt	-0.17	-45.83	1405	17765081	490.5	-0.14	-28.61	942	16287654 (-8.3%)	454.2 (-7.4%)
CPU-3 (121K)	global place	-2.13	-8437.48	11730	1711937	149.2	-1.96	-8057.19	11435	1691131	147.8
	place opt	-0.54	-164.78	2466	1439469	155.8	-0.48	-138.74	1981	1413154	153.1
	clock opt	-0.51	-37.68	414	1588135	141.9	-0.57	-32.98	359	1518498	137.7
	route opt	-0.49	-41.21	1207	1582822	143.0	-0.35	-36.24	1023	1520481 (-3.9%)	138.9 (-2.9%)
LDPC (46K)	global place	-1.14	-1411.74	2184	1289738	225.8	-1.10	-1331.30	2048	1233014	219.5
	place opt	-0.25	-292.49	2192	1454863	255.5	-0.21	-217.76	-217.76	1390693	248.6
	clock opt	-0.20	-156.62	1897	1857624	255.4	-0.16	-98.47	1757	1785355	248.4
	route opt	-0.24	-198.72	1976	1878969	261.8	-0.18	-123.94	1846	1803729 (-4.0%)	255.0 (-2.6%)

Table 4.3: Transfer Learning results on unseen designs. Middle column: single-design optimization as Table 4.2. Right-most column: optimization using database built by 5 other designs (the target design at each row remains totally unseen).

design	PD	vanilla DREAMPlace [**lin2019dreamplace**]			DREAM-GAN (no transfer)			DREAM-GAN (with transfer)		
(# cells)	stage	TNS (ns)	total WL (um)	total power (mW)	TNS (ns)	total WL (um)	total power (mW)	TNS (ns)	total WL (um)	total power (mW)
VGA (57K)	global place	-13999.49	2418386	345.5	-8057.19	2211154	342.2	-7426.29	2205350	342.3
	place opt	-2.06	1426918	279.8	-2.55	1456516	276.5	-1.84	1418695	274.9
	clock opt	-7.46	1579559	327.8	-5.37	1536218	322.4	-4.22	1514982	321.3
	route opt	-13.73	1586940	333.5	-7.06	1542569	329.1	-5.25	1528966	327.5
ECG (86K)	global place	-3429.43	665183	104.2	-3357.24	653160	103.8	-3334.45	652891	103.5
	place opt	-9.16	684917	111.3	-8.71	683390	110.1	-8.61	682702	109.9
	clock opt	-19.35	759633	115.3	-16.48	748224	114.4	-16.32	744759	113.4
	route opt	-36.47	765852	124.9	-24.54	753069	122.7	-27.33	751296	121.1

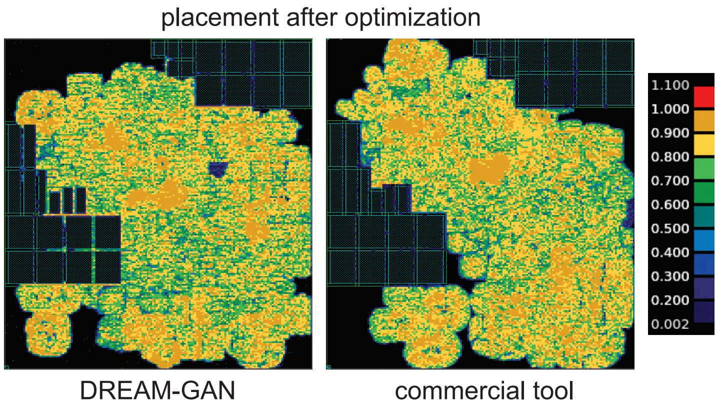


Figure 4.8: Density map comparison between the proposed DREAM-GAN and Synopsys ICC2 on the CPU-2 benchmark.

4.5.1 Single-Design Optimization Results

In this experiment, we perform single-design optimization by using the same design between DREAMPlace (the generator) and the target database. Our purpose is to demonstrate the effectiveness of using generative adversarial learning for placement optimization. Table 4.2 demonstrates the detailed optimization results, where we clearly observe that DREAM-GAN consistently outperforms vanilla DREAMPlace at each major PD stage across all 4 industrial benchmarks. Although the same design is used in the database during training, we still believe the achieved results are highly remarkable as DREAM-GAN is NOT using any "memorization" technique such as explicit net-matching, cell-alignment etc. The superior results are purely achieved by optimizing placement similarity scores via generative adversarial learning. Figure 4.8 further shows the bin-density map comparison between DREAM-GAN and *ICC2*, where we observe that although the underlying placements are visually different, the achieved bin-density maps are indeed similar, which demonstrates the effectiveness of the CNN-based discriminator.

4.5.2 Transfer Learning on Unseen Designs

In this experiment, we demonstrate that DREAM-GAN can indeed perform placement optimization on unseen designs. Precisely, we are not using the same design between DREAM-

Place (the generator) and the target database as in the previous experiment. Instead, we perform the placement optimization on an unseen design by using 5 other different designs to build database (in this work, we use 6 benchmarks in total). The optimization results are shown in Table 4.3, where we observe that with transfer learning on distinct designs (rightmost column) we can further improve the PPA metrics compared with using single-design for the optimization (middle column). Below, we provide our thoughts on the implications of the results achieved.

4.5.3 Discussion of Optimization Results

We believe the tremendous success of the proposed framework, DREAM-GAN, has three major implications: (1) “Placement style” can be transferred from one placer to another. This argument is supported by Table 4.2 that by using single-design optimization, DREAM-GAN can significantly improve vanilla DREAMPlace on industrial benchmarks in consistent. (2) “Placement style” is more related to a placer itself than the designs being placed. This argument is validated by Table 4.3, where we observe that placement quality can not only be transferred in the same designs, but also in completely different ones. (3) Without knowing the underlying algorithms or constraints, the “placement style” of a black-boxed placer can still be parameterized through generative adversarial learning.

Furthermore, in the experiments, we observe that DREAM-GAN not only improves the wirelength significantly, but also introduces notable improvements in power. We think this is because by following the placement distribution of *ICC2* in terms of cell locations, DREAM-GAN will introduce less buffers and logic fixing than the vanilla DREAMPlace during many optimization steps throughout the flow, which effectively results in less power consumption. Finally, we would like to emphasize that the runtime difference between the vanilla DREAMPlace and the proposed DREAM-GAN is no more than 10 minutes across all benchmarks, which is *practically negligible* compared with the global placement runtime of *ICC2*.

4.6 Conclusion

In this chapter, we have presented DREAM-GAN which is the first-ever learning-driven framework that improves an open-source placer towards commercial-quality using generative adversarial learning. In the experiments, we demonstrate that DREAM-GAN not only significantly advances vanilla DREAMPlace in single-design optimization, but also facilitates transfer learning to perform optimization on unseen designs. The main assumption of this work is that "placement style (quality)" is born with a placer itself which can be parameterized and transferred to another placer through generative adversarial learning. This assumption is proven empirically in this chapter. We believe this work shall present new routes in advancing placement.

CHAPTER 5
GAN-CTS: A GENERATIVE ADVERSARIAL FRAMEWORK FOR CLOCK TREE PREDICTION AND OPTIMIZATION

5.1 Background and Motivation

Clock tree synthesis (CTS) is a critical stage of physical design, since clock networks constitute a high percentage of the total 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, and elongated timing closure [77]. However, due to the high complexity and run time of the modern 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 the design targets, 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 [75]. Previous work [78] utilizes data mining tools to estimate the achieved skew and insertion delay. The authors of [79] employ statistical learning and meta-modeling methods to predict more essential metrics such as clock power and clock wirelength. The authors of [80] further consider the effect of non-uniform sinks and different placement aspect ratios. Another work [81] utilizes artificial neural networks (ANNs) to predict the transient clock power consumption based on the estimation of clock tree components. However, these previous works merely focus on enhancing the prediction of CTS metrics rather than the optimization of the outcomes. Therefore, their methods are not sufficient to achieve high-quality clock trees without the aid of other heuristic algorithms. To optimize CTS metrics, a recent work [82] develops an ML-powered clock tree construction algorithm which generates optimized clock trees based on a pre-trained CTS buffer prediction framework that estimates buffer usage. The proposed algorithm is proven to be generalizable on unseen designs.

The goal of this work is to construct a general and practical CTS modeling framework, which has the ability to predict CTS outcomes in high precision and perform CTS optimization by generating the CTS input parameter sets that lead to optimized clock trees for *general* designs in an unsupervised manner. Specifically, we take an industry-leading commercial tool, *Cadence Innovus*, as reference and demonstrates the feasibility of the proposed framework upon it. The proposed CTS prediction and optimization framework named GAN-CTS achieves the following aspects:

- **Generalizability:** We aim to develop a generalizable framework that achieves high-quality clock trees on general designs. We achieve this by leveraging generative adversarial learning which has the capability to inference optimized CTS input parameter sets on unseen designs.

- **Interpretability:** Instead of considering our framework as a black box, we leverage a gradient-based attribution method [83] to interpret the prediction made by the proposed framework.

- **Optimality:** Despite that the optimality of a clock tree is hard to demonstrate, in this work, we compare the optimization results achieved by the proposed framework to the ones achieved by the default settings of *Cadence Innovus*, and show that the proposed framework reaches never-seen, high-quality CTS optimization results.

5.2 Designing Experiments

5.2.1 Database Analysis

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

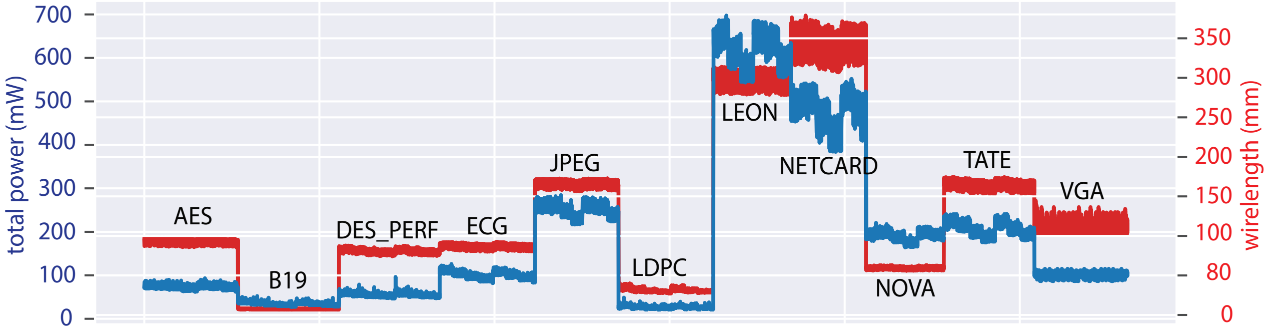


Figure 5.1: Total power (mW) and wirelength (mm) distributions among 115.5k full-chip designs of all 11 benchmarks in our database.

Table 5.1: Our benchmarks and their attributes in *TSMC 28nm*.

Design Name	# Nets	# FFs	# Cells	Usage
AES-128	90,905	10,688	113,168	training
B19	34,399	3,420	33,784	
DES_PERF	48,523	8,802	48,289	
LDPC	42,018	2,048	39,377	
NETCARD	317,974	87,317	316,137	
NOVA	138,171	29,122	136,537	
TATE	185,379	31,409	184,601	
ECG	85,058	14,018	84,127	testing
JPEG	231,934	37,642	219,064	
LEON3MP	341,263	108,724	341,000	
VGA_LCD	56,279	17,054	56,194	

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 process.*

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 $\hat{Y}$.*

In this work, we tackle Problem 1 and Problem 2 through machine learning approaches. Before elaborating the modeling process, we first describe and analyze our database.

5.2.2 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

Table 5.2: Modeling parameters we use and their values.

type	parameters	values or ranges
placement	aspect ratio	$\{0.5, 0.75, \cdots, 1.5\}$
	utilization	$\{0.4, 0.45, \cdots, 0.7\}$
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 earlyRouting layer	$\{2, 3, 4, 5, 6\}$
	min earlyRouting layer	$\{1, 2, 3, 4, 5\}$
	max buffer density	[0.3, 0.8]

CTS processes. The database is constructed based on *TSMC-28nm* technology node. In this work, we leverage 5 designs which are B19, LEON3MP, NETCARD, DES_PERF, VGA_LCD from the ISPD 2012 benchmark [43], and 6 other designs, including AES-128, LDPC, NOVA, ECG, TATE, JPEG, from *Opencores.org* to conduct the experiments. Table 9.2 shows the attributes of all 11 designs after being synthesized at *1125MHZ*.

Table 9.1 presents the modeling parameters and their ranges of values that we utilize. The ranges of the CTS related parameters are determined by reasonably widening the commercial tool's auto-setting values based on domain expertise. The goal is to generate a database with high variety in terms of clock metrics so that the proposed framework can better differentiate good designs from the bad ones, and comprehend which combinations of the parameters can lead to optimized clock trees. 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 the early global routing (EGR) stage, which is a procedure to reserve space for future detailed signal routing during the CTS stage (EGR is contained in CTS). Note that although "skew" is taken as an input target as shown in the table, commercial tools will not necessarily meet the skew target during CTS (skew is often further improved in future design steps), which makes the skew prediction (one of the CTS outcomes we focus in this work) problem non-trivial.

Table 5.3: 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

The combinations of the two placement related parameters give us 35 different placements per netlist. By running CTS with randomly sampled input parameters, we generate 300 clock trees per placement. Therefore, in total, we have $115.5k$ datum (clock tree instances) across 11 different netlists in our database. To substantiate the generality of our framework, in the experiments, we only utilize 7 netlists during the training process and perform the validations on the remained 4 unseen netlists as indicated in Table 9.2.

Figure 5.1 shows the total power and wirelength distributions of all $115.5k$ 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 5.3 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 the commercial CTS engine is very sophisticated and counter-intuitive, which is mainly due to the complicated high-dimensional inter-correlation among different CTS input parameters [79]. In this paper, we aim to employ machine learning methods to demystify the complicated CTS process.

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 5.3, 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 clock tree. In section 8.8, 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.

5.3 Overview of GAN-CTS

It has been widely acknowledged that GAN is a promising model that learns the complicated real-world data through a generative approach [84]. 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 such as random noise, while the objective of the discriminator is to distinguish the *generated* samples from the *real* samples that are targeted to be mimicked. Predicated on 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 based on different conditions, which benefits us to generate suitable CTS input parameters sets with respect to different benchmarks and even the ones that are not utilized in the training process.

A high-level view of our framework named GAN-CTS is shown in Figure 8.1. The framework is comprised of three sequential training (learning) stages. The first training stage is to extract key design features from placement images which represent flip flop distributions, clock net distributions, and trial routing results. Note that trial routing is a process performed in the end of the placement stage in the modern physical design flow. It provides a quick estimation of the routing congestion based on the given placement. To extract features from images, we adopt transfer learning by using a pre-trained convolutional neural network (CNN) named ResNet-50 [85], which is a 50-layer residual network that has skip connections. The goal of transfer learning is to leverage the trained convolutional filters to distill the hidden information in the placement layouts.

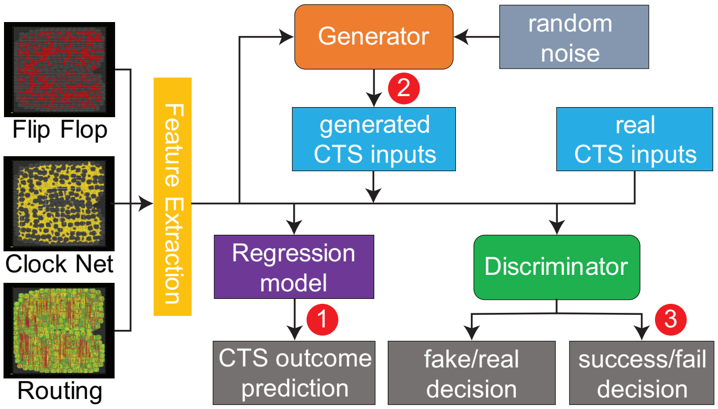


Figure 5.2: A high-level view of this work and the three objectives we have achieved. The first objective is to predict the CTS outcomes in high precision. The second objective is to recommend designers good CTS input settings. The third objective is to determine whether the input settings outperform commercial tool's auto-setting.

In the second training stage, we leverage the extracted features from the previous feature extraction stage as well as the clock tree instances in the database to train the regression model for CTS outcomes predictions. In this work, we select three essential CTS metrics that well represent the quality of a clock tree as the prediction and optimization targets. These three selected metrics are clock power, clock wirelength, and the achieved maximum skew. We compare two different regression approaches which are the meta-modeling technique adopted by previous work [80] and the proposed multi-task learning technique. We demonstrate that the proposed technique reaches better prediction accuracy with a much shorter training time. Furthermore, as mentioned in section 9.1, we do not consider our model as a black box as previous works. To interpret the predictions made by the regression model, we leverage a gradient-based attribution method [86] to quantitatively determine the importance of each CTS input parameter subject to different target outcomes.

The last training stage of our framework involves generative adversarial learning, where we leverage a conditional GAN to perform the CTS optimization and classification tasks. The regression model trained earlier now acts as a supervisor to guide the generator to generate the CTS input parameters sets that lead to optimized clock trees. Note that the extracted placement features from transfer learning are taken as the conditional input, where the original inputs of the vanilla GAN model are termed as the regular inputs in this work. The advantage of having the conditional input is to control the *modes* of the generated data, where we consider different benchmarks as different *modes*. Therefore, with the conditional approach, our framework has the ability to optimize *unseen* benchmarks that are not utilized during the training process.

Finally, a highlight of our framework is that a multi-task learning is conducted by the discriminator. In addition to the conventional task of distinguishing between the generated and the 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 CTS engine.

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.

5.4 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 (Problem 1 and Problem 2). In the meantime, we illustrate the detailed structures of the models. In the end, we summarize the overall training process in a complete algorithm.

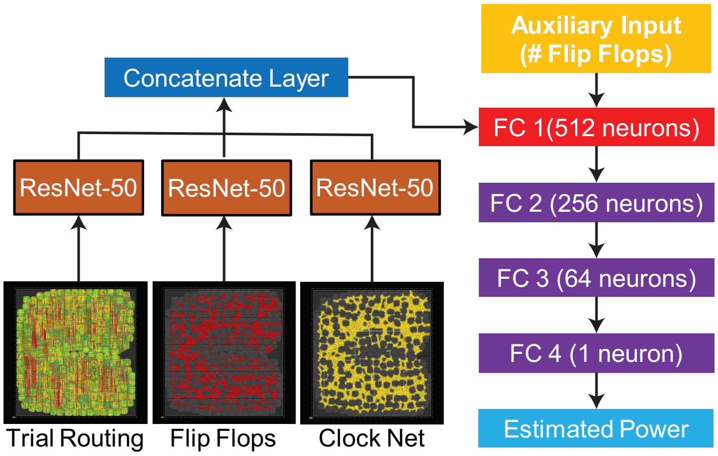


Figure 5.3: Our image feature extraction flow. The extracted features are colored in red. Note that each raw image vector extracted from ResNet-50 has 1024 dimensions. The concatenate layer thereby forms a vector in 3072 dimensions. Finally, with an auxiliary input that denotes the number of flip flops, the input of the self-devised FC layers are in 3073 dimensions.

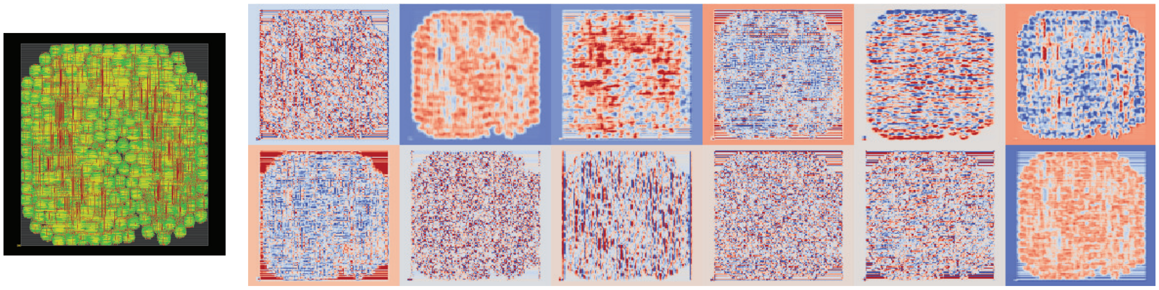


Figure 5.4: Visualization of trial routing image in 12 different convolutional filters of ResNet-50 [85], where the usage of metal layers is well captured across different filters.

5.4.1 Placement Image Feature Extraction

One of the innovations of this work is that we directly utilize placement images as inputs to predict and optimize the target CTS outcomes. The key rationale is that placement images contain important information of designs. Previous work [59] has demonstrated the efficiency of using placement images to predict routability and design rule violations (DRVs). In this work, we leverage the extracted features from placement images to solve CTS outcomes prediction and optimization problems. Our approach is built upon convolutional neural network (CNN) and transfer learning. As shown in Figure 5.3 we devise our own fully connected (FC) layers upon the pre-trained model named ResNet-50 [85], which is a

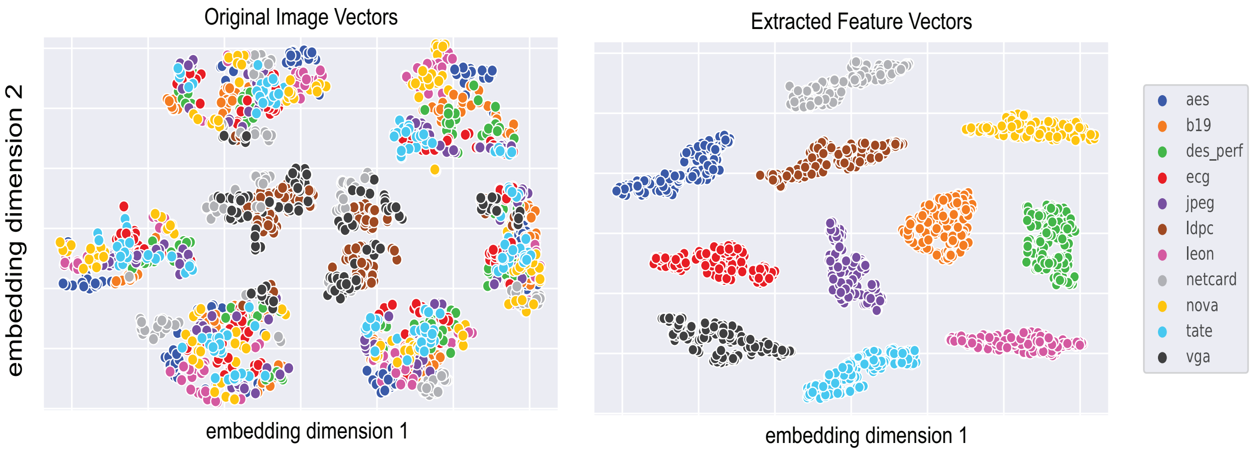


Figure 5.5: t-SNE visualizations of the original placement image vectors and the extracted feature vectors from our transfer learning flow, where the extracted features successfully characterize different designs.

CNN-based model pre-trained on the well-known ImageNet [87] dataset with millions of images.

Figure 5.4 shows the visualization of a trial routing image passing through twelve selected 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. Although ResNet-50 is powerful on many image datasets, it is not devised specifically for the 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 5.3, where 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 which represents the total number of flip flops to the first FC layer by concatenating it with the direct extracted features of the ResNet-50, In the training process, we fix the parameters of the pre-trained ResNet-50 model and only update the parameters of the self-devised FC layers. When the training is completed, each pre-CTS placement is encoded into a vector with 512 dimensions, which is the output of the first FC layer (denoted in red in the figure).

Our transfer learning approach accepts any image sizes, since we only utilize the pre-trained convolutional filters rather than the FC layers of the original ResNet-50 model. In the implementation, all the images in the database are in a dimension of 700 x 717 x 3. The achieved mean absolute percentage error of the unseen validation netlists is less than 0.7%, where the maximum absolute percentage error is 5% across training designs. 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) [44] to visualize the extracted features ($\in R^{512}$) in R^2 as shown in Figure 5.5. 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 design characteristics.

Finally, to justify the achievement of transfer learning, in section 8.8, we perform an experiment of comparing the CTS prediction results between with and without using transfer learning from the ResNet-50 model. Since the goal of transfer learning is to precisely characterize different designs, in the setting without using transfer learning, we handcrafted 4 features to represent the extracted features in Figure 5.3. These 4 features include number of cells, number of flip flops, number of nets, and number of ports in the design.

5.4.2 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 5.3, in this paper, we target at predicting and optimizing three CTS outcomes which are clock power, clock wirelength, and the maximum skew. In this work, we analyze two different strategies to construct the regression model.

Multi-Model Uni-Output. The first strategy is built upon meta-modeling, which is the strategy adopted by the state-of-the-art academic works [79] and [80]. The key concept

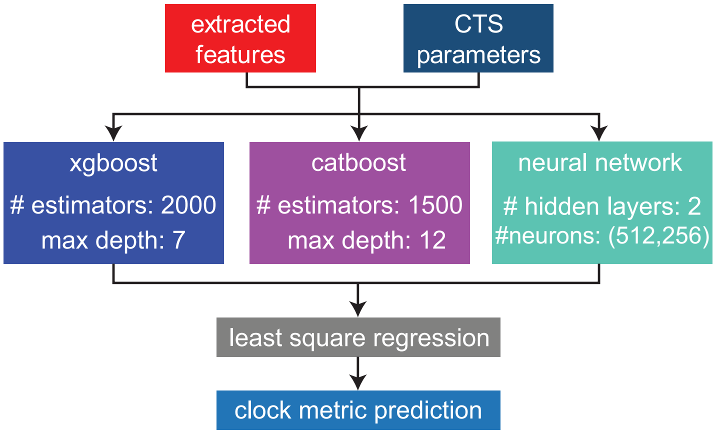


Figure 5.6: Visualization of single metal-model, which is stacked by three base models through weighted least square regression.

of meta-modeling is that for each target CTS outcome, a single meta-model is constructed by combining multiple base models through a high-level aggregation function. This ensembled meta-model is expected to make more stable and accurate predictions than any individual base model. In [79] and [80], traditional regression techniques such as radial basis functions (RBF) [88] and support vector machine (SVM) are utilized as the base models, where the weighted least square regression [89] is leveraged as the aggregation function in both works. However, these regression techniques utilized in the base models of previous works are known to be easily biased to the dataset [**vapnik2013nature**], which thereby cannot be generalized to unseen designs.

To overcome the drawback of previous works, we leverage *xgboost* [90], *catboost* [91], neural network (NN) as the base models of our framework. These base models are combined through the weighted least square regression [89] as previous works to construct the meta-model. Note that each target CTS outcome requires a meta-model for the prediction, therefore, we construct three meta-models to predict the three target CTS metrics as mentioned earlier. As shown in Figure 5.6, each base model takes the features extracted from the feature extraction process and the CTS parameters as inputs, and outputs the prediction of the specific clock metric. The core idea of meta-modeling is to reduce the variance of each base model, and therefore eliminate the impact of the bias in the database. However,

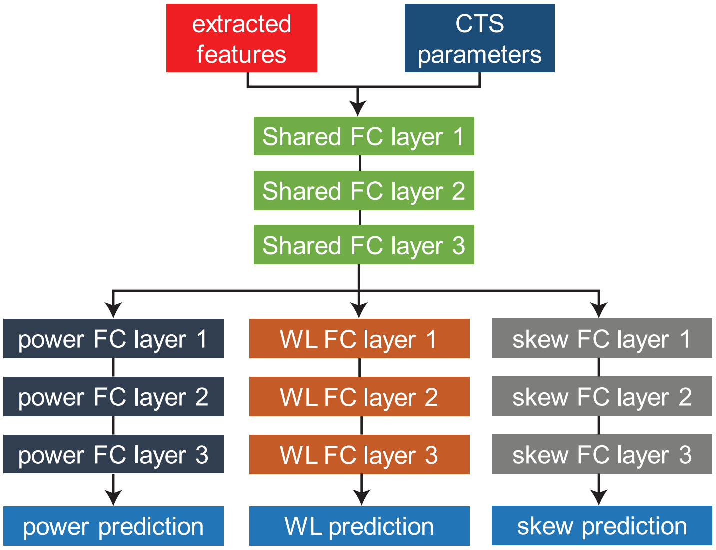


Figure 5.7: Our proposed multi-objective regression model. The detailed architectures are as follows. The shared FC layers colored in green have number of neurons equal to 512, 256, and 128 in sequential, where each dedicated FC layer group has number of neurons equal to 64, 32, 1 from input to output.

a foreseeable issue of using the meta-modeling technique is that it requires a long training time due to the large number of the training parameters.

Uni-Model Multi-Output. The second strategy we adopt to construct the regression model is through multi-task (multi-objective) learning, where we build a multi-output deep neural network to predict the three target CTS metrics simultaneously. The visualization of the model is shown in Figure 5.7. The model takes the extracted features along with the CTS parameters as inputs and outputs three predictions simultaneously. As shown in the figure, the inputs are passed through shared layers and dedicated layers to predict the clock metrics. The key rationale of multi-task learning is that different CTS outcomes are not independent of each other (e.g. clock wirelength often has a high correlation with clock power). Therefore, instead of isolating the parameters for each prediction as in the meta-learning approach, we leverage shared layers to enable different objectives to own mutual information, which helps to reduce the training time as well as enhances the accuracy of the predictions. 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 8.8, where we observe that the multi-task learning approach achieves higher accuracy in a shorter training time comparing to the meta-learning approach.

5.4.3 Interpreting Prediction Results

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 [83] 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

$$a_i^L = \begin{cases} S_i(x) - S_i(\hat{x}) & \text{if neuron i is the output of interest} \\ 0 & \text{otherwise,} \end{cases} \tag{5.1}$$

$$a_i^l = \sum_j \frac{z_{ji} - \hat{z}_{ji}}{\sum_i z_{ji} - \sum_i \hat{z}_{ji}} \cdot a_j^{l+1}, \tag{5.2}$$

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.

5.4.4 CTS Optimization

In this paper, we develop two CTS optimization techniques based on the presented regression model. Specifically, the objective of the optimization problem (Problem 2) is to find the CTS input parameter sets of the commercial tool that lead to optimized clock trees. Note that in the experiments, we select the multi-task learning regression model (uni-model multi-output) as the base model (guidance provider) for optimization, since it achieves the

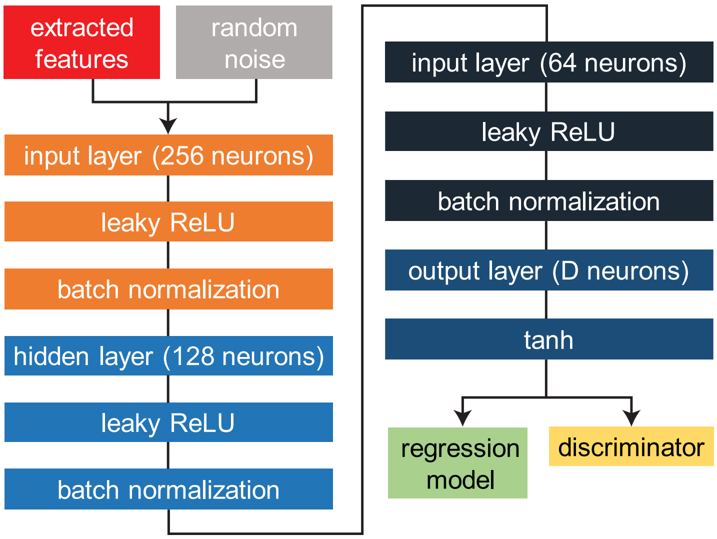


Figure 5.8: Detailed structure of our GAN generator.

best prediction accuracy compared with other models.

GAN-based Optimization

The first optimization approach we develop leverages generative adversarial learning to perform the optimization, where we train a generative model (the generator) that learns to generate the parameter sets which lead to optimized CTS outcomes. As aforementioned, prior to the GAN learning, we pre-train the regression model as the guidance provider. The generator is expected to generate optimized CTS input parameter sets by maximizing the CTS quality predicted by the pre-trained regression model. Before illustrating the objectives of the optimization process, we first describe the model structure of the generator.

Figure 5.8 shows the detailed 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. The leaky ReLU [92] 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 [93] layers are utilized to normalize the inputs of each hidden layers 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 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

$$\hat{x} = \frac{x}{\max_{x \in supp(x)}(x)} \times 2 - 1. \tag{5.3}$$

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

$$\mathcal{L}_{G_D} = \mathbb{E}_{z,f}[log(D(G(z, f))]. \tag{5.4}$$

The second objective is to generate the CTS input parameter sets that lead to optimized clock trees by maximizing the clock tree quality r predicted by the regression model, where r is defined as

$$r := H(G(z, f)) = -\prod_{i=1}^{N} \frac{R_i(G(z, f))}{\text{auto-setting result of target } i}. \tag{5.5}$$

In Equation 5.5, N denotes the number of target CTS outcomes and R_i denotes the corresponding prediction of the regression model. In the implementation, we have $N = 3$ which represents the clock wirelength, clock power, and the maximum skew. The objective function of maximizing the prediction of clock tree quality can be formulated as

$$\mathcal{L}_{G_P} = \mathbb{E}_{z,f}\,[r]. \tag{5.6}$$

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

Algorithm 7 Bayesian optimization for CTS.
We leverage UCB [94] to realize the acquisition function.

```
Input: $R$: a pre-trained regression model, $a(\mathbf{x})$: an acquisition function.
Output: $\{\mathbf{x}\}$: optimized CTS parameter sets.
Initialize a prior function $f(\mathbf{x})$.
repeat
$\{\mathbf{x}\}_{i=1}^{k} \leftarrow$ Sample $k$ sets of CTS parameters based on $a(\mathbf{x})$ such that $f(\mathbf{x})$ is maximized.
$\{r\}_{i=1}^{k} \leftarrow$ Evaluate $\{\mathbf{x}\}_{i=1}^{k}$ using $R$ by Equation 5.5.
Update $f(x)$ with $\{\mathbf{x}\}_{i=1}^{k}$, $\{r\}_{i=1}^{k}$ using Gaussian Process.
until $\{r\}_{i=1}^{k}$ no longer improve.
```

generator as

$$\max_{G} \mathbb{E}_{\substack{z\sim p_z \\ f\sim p_d}}[log(D(G(z,f))) + r]. \tag{5.7}$$

Bayesian Optimization

In addition of training a GAN-based framework that performs the optimization by learning key parameter distributions in a generative manner, in this work, we also leverage the Bayesian optimization [95] technique to solve the CTS optimization problem for comparison. Bayesian optimization is a popular surrogate optimization technique that optimizes black-box functions of arbitrary forms. Unlike neural networks that require gradients to update the network parameters, Bayesian optimization models a prior (surrogate) function using Gaussian process to characterize the target black-box function. Recently, in the realm of EDA, previous work [96] has applied such technique to perform parameter optimization of commercial tools, where it is shown that the achieved results are better than the ones achieved by the Genetic [97] which is used widely for parameter optimization.

In this paper, instead of leveraging Bayesian optimization to directly optimize the commercial tool that introduces significant runtime as the previous work [96] that introduces costly runtime, we leverage it to optimize the pre-trained regression model. The optimization is summarized in Algorithm 7, which outputs the CTS input parameter sets $\{\mathbf{x}\}$ that lead to optimized clock trees. The acquisition function $a(\mathbf{x})$ guides the sampling of the next CTS parameter sets $\{\mathbf{x}\}_{i=1}^{k}$ which are utilized to update the prior (surrogate) function

$f(\mathbf{x})$. In the implementation, the sampling of the acquisition function $a(\mathbf{x})$ is performed based on upper confidence bound [94] (UCB). Note that since in our scenario, our goal is to minimize the reward r (Equation 5.5) to optimize clock trees, we leverage Bayesian optimization to maximize $-r$. At each step of the iteration, Gaussian process is fit into known samples (parameter sets previously explored) to update the prior (surrogate) function $f(x)$.

Joint GAN-based and Bayesian-based Optimization

In this work, we further combine the two presented optimization techniques to optimize the CTS metrics. Since a trained GAN-based framework has the ability to suggest optimized CTS input parameter sets in constant time, for the Bayesian-based approach, instead of starting with randomly sampled observations as shown in Algorithm 7, we leverage the GAN-based model to suggest the initial CTS parameter sets. Furthermore, because the CTS input parameter sets suggested by the GAN-based model are expected to be in a more reliable (optimized) region than the ones achieved by sampling randomly, we further leverage a sequential domain reduction technique introduced in [98] to adaptively refine the search space based on the existing explored solutions. The key rationale behind is that instead of searching new parameter sets from the original wide ranges as shown in Table 9.1, for each parameter, we can refine the sample range based on parameter sets suggested by the GAN-based framework which are already optimized. We expect this combined optimization technique can help us achieve better optimization results than any individual technique. The optimization results are shown in section 8.8.

5.4.5 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 5.9. 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

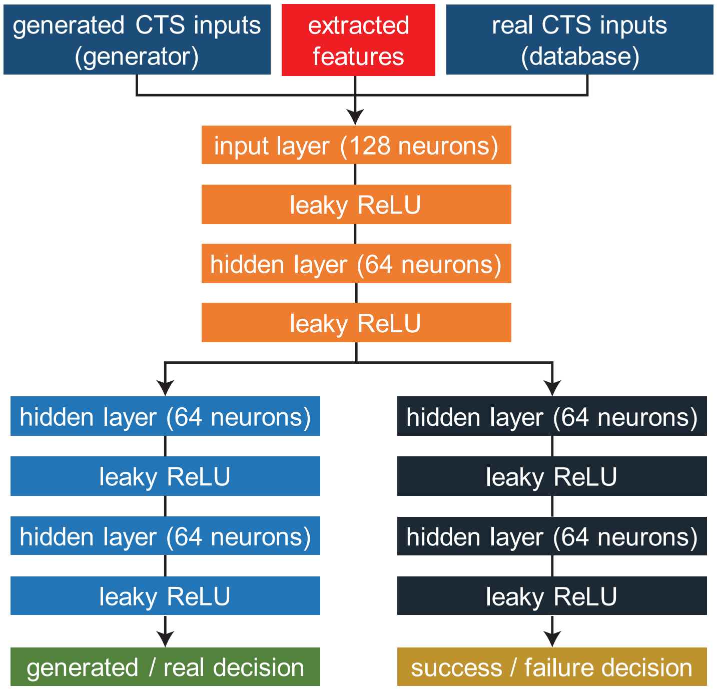

generated CTS inputs (generator)
extracted features
real CTS inputs (database)
input layer (128 neurons)
leaky ReLU
hidden layer (64 neurons)
leaky ReLU
hidden layer (64 neurons)
leaky ReLU
hidden layer (64 neurons)
leaky ReLU
hidden layer (64 neurons)
leaky ReLU
hidden layer (64 neurons)
leaky ReLU
generated / real decision
success / failure decision

Figure 5.9: Detailed structure of our GAN discriminator.

input which denotes the features extracted from placement images. Note that when the regular input represents the real samples x, the conditional input f should 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 5.3, since each CTS input parameters has a different unit, 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 also has two objectives as the discriminator. One is to distinguish the generated samples from the real samples, which is derived as

$$\begin{aligned} \mathcal{L}_{D_g} =& \mathbb{E}_{(x,f)\sim p_d}[log(D_g(x, f))] \\ &+ \mathbb{E}_{\substack{z\sim p_z \\ f\sim p_d}}[log(1 - D_g(G(z, f)))]. \end{aligned} \tag{5.8}$$

The other objective is to classify whether a given input parameter set can lead to a successful CTS run or not, where the objective is be formulated as

$$\mathcal{L}_{D_s} = \mathbb{E}_{x \sim p_d}[log(D_s(x, f))], \tag{5.9}$$

which represents the cross-entropy between the classification groundtruths and the predictions made by our framework. Note that the definition of successful and failed CTS runs are defined in section 5.3, and the discriminator is updated by Equation 5.9 only when the regular input represents the real samples x. The reason we introduce a new objective to the discriminator is because its attribute is similar as the discriminator's conventional objective, where both of them are performing binary classification. Therefore, some latent features can be shared in the early network as shown in Figure 5.9. Finally, the training process of the discriminator is summarized as

$$\begin{aligned} \max_{D} \mathbb{E}_{(x,f) \sim p_d}&[log(D_g(x, f)) + log(D_s(x, f))] \\ &+ \mathbb{E}_{\substack{z \sim p_z \\ f \sim p_d}}[log(1 - D_g(G(z, f)))]. \end{aligned} \tag{5.10}$$

5.4.6 Training Methodology

Based on the structures and the objective functions presented, we now illustrate the training process of our framework in Algorithm 8, where a gradient descent optimizer Adam [38] is utilized across different training stages. First, we train the regression model (lines 2-9) which serves as a guidance provider in the training process of the conditional GAN. Note that the regression model we adopt in the GAN-CTS framework is constructed through multi-task learning. Following from the regression learning, 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} ($\theta_{d_1} \cap \theta_{d_2} \neq \phi$) to represent different tasks, since a multi-task learning is conducted. The overall training process is completed when the losses of the generator and the discriminator reach an equilibrium,

Algorithm 8 GAN-CTS training methodology.
We use default values of $\alpha_r = 1e^{-4}$, $\alpha_{GAN} = 1e^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $m = 128$.

```
Input: $\{f\}$: extracted placement features, $\{x\}$: training data, $\{Y\}$: target CTS metrics for prediction and optimization.
Input: $\alpha_r$: learning rate of regression model, $\alpha_{GAN}$: learning rate of GAN, $m$: batch size, $\{\theta_r\}_0$: initial parameters of regression model, $\theta_{g_0}$: 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.
$N \leftarrow length(y)$
while $\{\theta_r\}$ do not converge do
    Sample a batch of training data $\{x^{(i)}\}_{i=1}^m \sim p_d$
    Take features $\{f^{(i)}\}_{i=1}^m$ corresponding to $\{x^{(i)}\}_{i=1}^m$
    for $k \leftarrow 1$ to $N$ do
        $g_{r_k} \leftarrow \nabla_r[\frac{1}{m}\sum_{i=1}^m (R_k(f^{(i)}, x^{(i)}) - Y_k^{(i)})^2]$
        $\theta_{r_k} \leftarrow Adam(\alpha_R, \theta_{r_k}, g_{r_k}, \beta_1, \beta_2)$
while $\theta_g$ and $\theta_d$ do not converge do
    Sample a batch of training data $\{x^{(i)}\}_{i=1}^m \sim p_d$
    Take features $\{f^{(i)}\}_{i=1}^m$ corresponding to $\{x^{(i)}\}_{i=1}^m$
    Sample a batch of random vectors $\{z^{(i)}\}_{i=1}^m \sim p_z$
    $g_{d_1} \leftarrow \nabla_{\theta_{d_1}}[\frac{1}{m}\sum_{i=1}^m log(D_{\theta_{d_1}}(x^{(i)}, f^{(i)}))$
            $+\frac{1}{m}\sum_{i=1}^m log(1 - D_{\theta_{d_1}}(G_{\theta_g}(z^{(i)}, f^{(i)})))]$
    $\theta_{d_1} \leftarrow Adam(\alpha_{GAN}, \theta_{d_1}, g_{d_1}, \beta_1, \beta_2)$
    $g_{d_2} \leftarrow \nabla_{\theta_{d_2}}[\frac{1}{m}\sum_{i=1}^m log(D_{\theta_{d_2}}(x^{(i)}, f^{(i)}))]$
    $\theta_{d_2} \leftarrow Adam(\alpha_{GAN}, \theta_{d_2}, g_{d_2}, \beta_1, \beta_2)$
    Sample a batch of random vectors $\{z^{(i)}\}_{i=1}^m \sim p_z$
    Sample a batch of features $\{f^{(i)}\}_{i=1}^m$
    $g_\theta \leftarrow -\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^m log(D_{\theta_{d_1}}(G_{\theta_g}(z^{(i)}, f^{(i)})))$
    $\theta_g \leftarrow Adam(\alpha_{GAN}, \theta_g, g_\theta, \beta_1, \beta_2)$
    $r \leftarrow \prod_{k=1}^N \frac{R_k(G(\{z^{(i)}\}_{i=1}^m, \{f^{(i)}\}_{i=1}^m))}{\text{auto-setting result of outcome } k}$
    $g_\theta \leftarrow -\nabla_{\theta_g}\frac{1}{m}\sum_{i=1}^m r^{(i)}$
    $\theta_g \leftarrow Adam(\alpha_{GAN}, \theta_g, g_\theta, \beta_1, \beta_2)$
```

which takes about 24 hours on a machine with 2.40 GHz CPU and a NVIDIA RTX 2070 graphics card.

5.5 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 [99] library. As mentioned in section 5.2, we utilize *Cadence Innovus v18.1* to generate a database containing $115.5k$ clock trees with 385 different placements across 11 netlists under TSMC-

Table 5.4: CTS outcomes prediction results of three regression approaches. MAPE denotes mean absolute percentage error (%), MAXE denotes max. absolute percentage error (%), and CC denotes correlation coefficient. Note the the validations are performed on *unseen netlists*.

modeling	unseen	clock power			clock wirelength			achieved skew		
type	netlist	MAPE	MAXE	CC	MAPE	MAXE	CC	MAPE	MAXE	CC
multi-model	ecg	4.41	24.47	0.963	1.68	21.24	0.959	5.65	32.47	0.955
uni-output	jpeg	4.53	39.65	0.950	2.89	22.52	0.957	6.36	35.43	0.951
(meta-model)	leon	4.78	42.83	0.957	3.94	24.54	0.953	7.34	41.92	0.944
(w. transfer learning)	vga	4.31	27.45	0.964	1.37	29.61	0.961	5.83	35.61	0.967
uni-model	ecg	2.14	17.46	0.972	2.66	8.57	0.971	3.92	34.83	0.958
multi-output	jpeg	3.88	20.29	0.965	1.98	12.35	0.973	3.87	29.63	0.961
(multi-task)	leon	2.37	16.19	0.969	2.02	14.16	0.966	4.28	27.79	0.964
(w. transfer learning)	vga	2.54	11.23	0.962	1.77	17.03	0.969	4.25	31.18	0.963
uni-model	ecg	7.63	79.40	0.920	5.43	35.59	0.945	16.86	129.48	0.837
multi-output	jpeg	8.29	67.33	0.904	6.77	42.98	0.949	15.70	112.15	0.821
(multi-task)	leon	8.75	77.65	0.919	5.12	38.92	0.942	12.26	127.14	0.841
(w.o. transfer learning)	vga	6.81	47.57	0.941	5.36	59.29	0.949	11.78	91.55	0.850

28nm technology node. The designs we utilize are shown in Table 9.2, which are from ISPD 2012 benchmark [43] and *OpenCores.org* To prove the generality of our framework, we only use 7 netlists during the training process, and leverage the rest four designs (ECG, LEON, JPEG, VGA) to perform the validations.

5.5.1 CTS Prediction and Interpretation Results

In this experiment, we evaluate the regression approaches on three target CTS outcomes with two evaluation metrics: mean absolute percentage error (MAPE), maximum absolute percentage error (MAXE), and correlation coefficient (CC). Table 5.4 demonstrates the evaluation results of the three different regression approaches presented earlier. The first approach termed multi-model uni-output represents the meta-modeling method, where for each CTS target outcome, we build a dedicated meta-model using the structure defined in Figure 5.6. The second approach named uni-model uni-output leverages the modeling method shown in Figure 5.7, where we build a single model to predict three target CTS outcomes simultaneously. Finally, the third approach follows the modeling structure of the second approach, however, instead of using the features ($\in R^{512}$) extracted from the transfer learning flow, we handcrafted 4 features to represent different designs. These 4 features include number of cells, number of flip flops, number of nets, and number of ports in a given design.

It is shown that the second approach (multi-task learning with transfer learning enabled) achieves lower evaluation errors among all target metrics on the *unseen* netlists than the other approaches. Note that the training time of the multi-task learning approach is about 3 hours, where the training time of the meta-learning approach is about 6 hours (calculated by summing training time of individual meta-models). Two main conclusions can be drawn from this experiment. First, the multi-task learning takes the advantage of the fact that different CTS outcomes are not independent of each other. Therefore, shared layers not only expedite the training process but also improve the prediction accuracy of CTS tar-

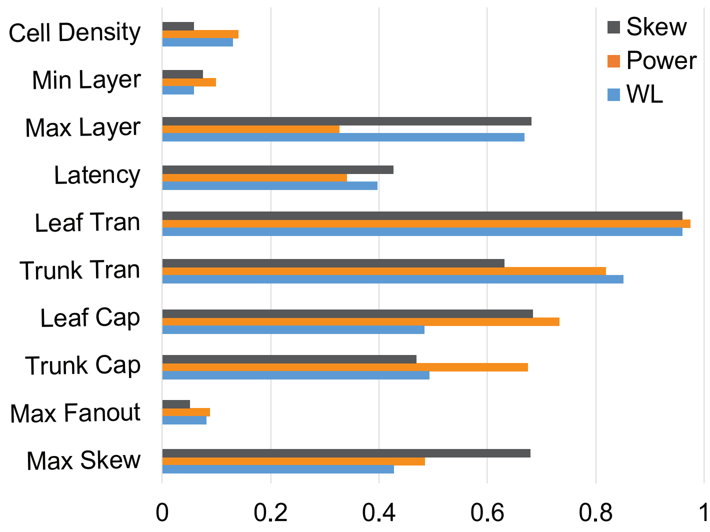


Figure 5.10: Relative importance of CTS input parameters on skew, power, and wirelength for JPEG benchmark.

gets. Second, it is demonstrated that the transfer learning approach provides a better way to characterize different designs compared with using the manually enumerated features. Indeed, many important design characteristics related to CTS process such as the distribution of flip flops and the metal layer usage of trial routing are not straightforward to be enumerated manually. Therefore, transfer learning provides a great benefit to characterize different designs.

In spite of the high prediction accuracy achieved, designers would not benefit much without explaining the predictions made by the model. Understanding the reasons behind the predictions is crucial. As shown in Figure 5.10, we evaluate the importance of each CTS input parameter based on the predictions presented in Table 5.4. As mentioned in subsection 5.4.2, we define the importance through a gradient-based attribution method named DeepLIFT [83]. 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

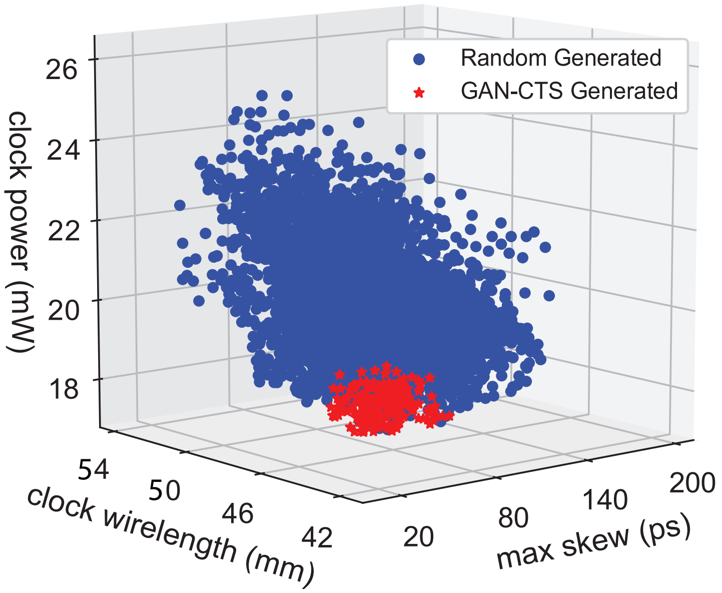


Figure 5.11: Distributions of random generated vs. GAN-CTS generated clock trees on the ECG benchmark. The commercial auto-setting achieved a clock tree with values of 23.56mW in clock power, 49.69mm in clock wirelength, and 16ps in skew.

parameters. Below, we explain two important phenomenons observed from Figure 5.10.

1. The slew constraints for leaf cells and trunk cells have 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 the clock routing because clock nets will inevitably use low metal layers to connect the sinks, which results in long clock paths and hence a large maximum skew.

5.5.2 CTS Optimization Results

In this experiment, we demonstrate the optimization results achieved by our GAN-CTS

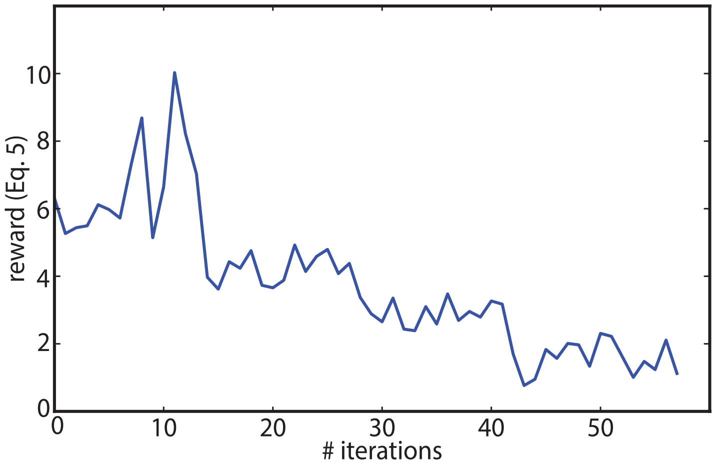


Figure 5.12: Bayesian optimization on VGA benchmark (starting from random sampled CTS parameter sets). Reward r is defined in Equation Equation 5.5.

framework compared with the Bayesian optimization [95] technique leveraged by previous work [96] and the auto-setting offered by the commercial tool, where a joint optimization is performed on three target CTS metrics: clock wirelength, clock power, and the maximum skew. Figure 5.11 first shows the optimization result of the ECG 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 trained generator to suggest 100 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 9.1.

The detailed optimization results on the four unseen netlists are shown in Table 5.5 and the corresponding CTS input parameters are shown in Table 5.6. The method “GAN-CTS + bayes“ denotes the combined optimization technique presented in subsubsection 5.4.4, where we take the GAN-CTS suggested parameter sets as the initial sets of the Bayesian

Table 5.5: Achieved clock metrics comparison between commercial auto-setting (auto), Bayesian optimization (bayes), and GAN-CTS. The method "GAN-CTS + bayes" denotes using the generator suggested CTS parameter sets as the initial solutions of Bayesisan optimization along with the sequential domain reduction technique [98]. Note that the four benchmarks are *unseen* during the training phase.

netlist	CTS metrics	auto-setting	bayes	GAN-CTS	GAN-CTS + bayes
ecg	# inserted buffers	417	128 (-69.3%)	96 (-76.9%)	92 (-77.9%)
	clock power (mW)	23.56	19.11 (-18.8%)	18.72 (-20.5%)	18.61 (-21.0%)
	clock WL (mm)	49.69	43.09 (-13.2%)	42.36 (-14.7%)	42.30 (-14.8%)
	maximum skew (ns)	0.016	0.018 (+12.5%)	0.014 (-12.5%)	0.014 (-12.5%)
jpeg	# inserted buffers	1093	296 (-72.9%)	240 (-78.0%)	268 (-75.5%)
	clock power (mW)	33.26	27.32 (-17.9%)	26.33 (-20.8%)	27.14 (-18.4%)
	clock WL (mm)	130.71	118.07 (-9.7%)	115.22 (-11.9%)	116.49 (-10.8%)
	maximum skew (ns)	0.022	0.024 (+9.0%)	0.024 (+9.0%)	0.023 (+4.5%)
leon	# inserted buffers	2962	1453 (-50.9%)	824 (-72.2%)	798 (-73.1%)
	clock power (mW)	81.12	74.28 (-8.4%)	69.69 (-14.1%)	67.94 (-16.2%)
	clock WL (mm)	326.36	307.81 (-5.6%)	296.15 (-9.2%)	292.88 (-10.2%)
	maximum skew (ns)	0.03	0.032 (+6.6%)	0.028 (-6.6%)	0.029 (-3.3%)
vga	# inserted buffers	505	186 (-63.1%)	109 (-78.4%)	127 (-74.8%)
	clock power (mW)	33.72	29.16 (-13.5%)	26.74 (-20.7%)	27.75 (-17.7%)
	clock WL (mm)	52.61	45.31 (-13.8%)	41.29 (-21.5%)	41.60 (-20.9%)
	maximum skew (ns)	0.036	0.033 (-8.3%)	0.023 (-36.1%)	0.022 (-38.8%)

optimization process and leverage the sequential domain reduction technique [98] to refine the search space. We observe in general, the combined technique reaches better CTS optimization results in terms of the reward defined in Equation Equation 5.5, and the proposed GAN-CTS framework outperforms the basic Bayesian optimization technique adopted by the previous work [96] across all *unseen* designs. This in fact demonstrates that the proposed GAN-CTS framework provides better starting points for the vanilla Bayesian optimization approach. Note that the selection of the GAN-CTS generated trees is conducted by taking the clock tree with the least maximum skew among the 100 trees suggested. Figure 5.12 further shows the Bayesian optimization process on the VGA benchmark, where the iteration stops when the reward evaluation no longer improves after 15 iterations. Finally, Figure 8.8 further exhibits the layout comparison of the four testing benchmarks (unseen during training). It is observant that the clock wirelength of the GAN-CTS optimized tree is much shorter than the one auto-generated by the commercial engine.

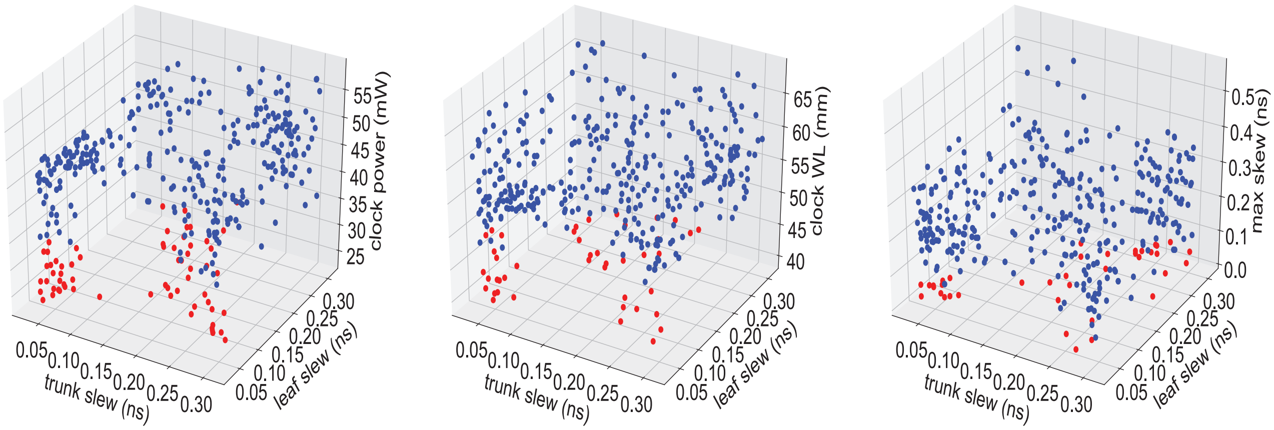


Figure 5.13: VGA slew sweeping experiments. Out of the 10 CTS input parameters as shown in Table 9.1, we sweep around the leaf and trunk target slew values while fixing others as auto-set and generate 500 clock trees in total. For each CTS metrics (i.e., clock power, clock wirelength, and maximum achieved skew), we plot the scatter distribution of the 500 clock trees denoted in blue and red dots, where red dots denote the ones whose underlying CTS metric are better than the auto-generated clock tree from the commercial tool. In summary, compared with the auto-generated clock tree, there are 61 (out of 500) trees whose clock power are better, 50 whose clock wirelength are better, and 32 whose achieved skew values are better, where the corresponding Venn diagram is shown in Figure 5.14.

5.5.3 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 5.3, success and 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. If two out of three target metrics are better, then we consider it as a success. Table 8.5 summarizes the classification results in a confusion matrix with NOVA benchmark. The accuracy and the F1-score are 0.930 and 0.932, respectively. With the accuracy demonstrated, we believe designers can not only benefit from the generator but also the discriminator by efficiently pruning out the CTS input parameter sets that have little advantage over the the commercial auto-setting.

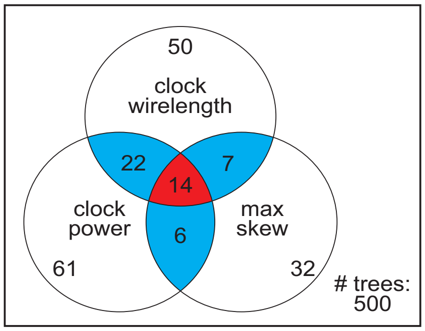


Figure 5.14: Venn diagram of the VGA slew sweeping experiment (Figure 5.13). Note that a number on a colored region denotes the number of trees fall into that region, where a number on an uncolored region denotes the number of trees in the shape boundary.

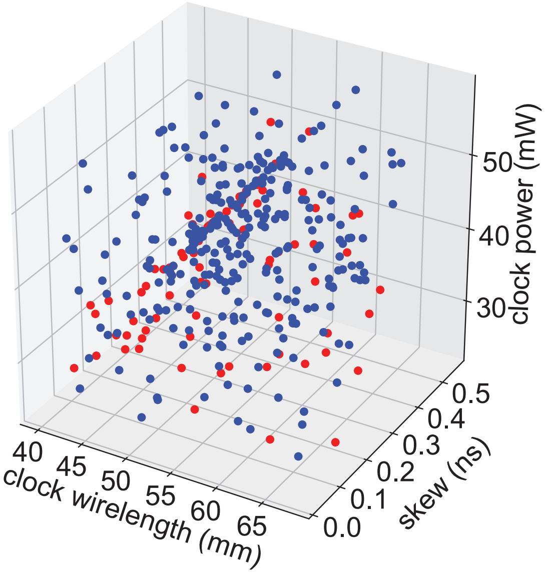


Figure 5.15: CTS metric distribution of the VGA slew experiment (Figure 13). The red dots denote the clock trees that are achieved with both trunk slew and leaf slew targets smaller than $0.1ns$.

5.6 Discussion

The proposed framework, GAN-CTS, is a helper model (rather than a surrogate model) of commercial CTS engines, whose goal is to support the engines to find the CTS input parameter combinations that result in optimized clock trees. In this work, we take *Cadence Innovus* as our reference commercial tool, however, the proposed method can be easily applied to other tools which also parameterize the CTS process into different input settings. Note that the goal of this work is *not* to replace the existing commercial CTS engines, but

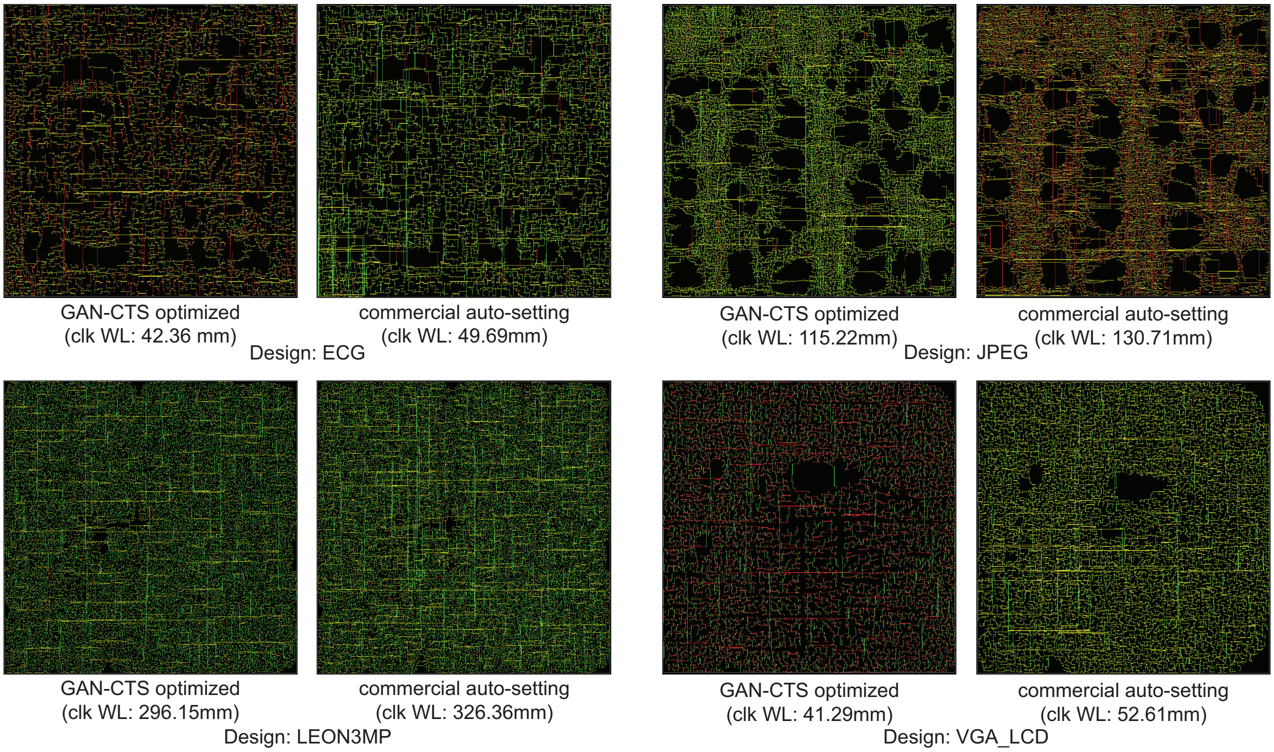


Figure 5.16: Clock tree layout comparison of four validation benchmarks. GAN-CTS optimized clock trees have observant clock wirelength saving. The detailed comparisons are reported in Table 5.5.

to provide tool users fast and reliable CTS prediction and optimization techniques without spending significant amount of time in design space exploration. In the below sub-sections, we further describe different aspects of the proposed CTS modeling method in detail.

5.6.1 Non-triviality of the CTS Modeling Problem

The CTS modeling problem we are dealing with in this work is in fact a high-dimensional modeling problem, which is stated in [80] to be difficult and non-trivial due to the *curse of high dimensionality* [100]. To quantify the non-triviality of this problem in our experimental settings, we perform a slew sweeping experiment on the VGA benchmark, where we generate 500 clock trees by sweeping the leaf and trunk slew targets while fixing all other input parameters in Table 9.1 as auto-set. The experimental result is shown in Figure 5.13. For the three CTS metrics we focus on in this work, we plot the scatter distribution of the

Table 5.6: GAN-CTS suggested and commercial auto-setting's CTS input parameters (refer to Table 5.5). Note that the commercial clock router has the same auto-setting values for different designs. The capacitance constraints in the auto-setting scenario are varied from net to net, which are subject to the max capacitance constraint of the driving pins.

CTS parameters	ecg	jpeg	leon	vga	auto setting
max skew (ns)	0.11	0.14	0.04	0.02	0.05
max fanout	120	139	173	84	100
max cap trunk (pF)	0.24	0.16	0.28	0.11	net-based
max cap leaf (pF)	0.22	0.12	0.19	0.25	net-based
max slew trunk (ns)	0.081	0.285	0.066	0.123	0.05
max slew leaf (ns)	0.047	0.107	0.098	0.074	0.05
max latency (ns)	0.31	0.26	0.15	0.28	0.1
max eGR layer	5	4	4	5	6
min eGR layer	3	2	3	3	2
max buffer density	0.72	0.63	0.69	0.68	0.75

Table 5.7: Confusion matrix of success vs. failure classification in LEON benchmark. Failure means worse than auto-setting.

		Predictions		
		Success	Failure	Total
Ground Truths	Success	6974	522	7496
	Failure	498	2251	2749
	Total	7472	2773	10245

500 clock trees, where the red colored dots denote the trees whose achieved metric is better than the one achieved by the commercial tool auto-generated clock tree. In the figure, we observe that there is no apparent "sweet spot" that guarantees high quality clock trees. In addition, Figure 5.14 shows the Venn diagram from the three subplots in Figure 5.13, and Figure 5.15 demonstrates further the distributions of the tree targeted CTS metrics in this work, where we observe that there is no apparent sweet spot of the slew targets that guarantee to result in optimized clock trees. It is shown that out of 500 generated clock trees, only 14 of them (2.8%) whose all three CTS outcomes (clock power, clock wirelength, and achieved skew) are better than the ones achieved by the auto-generated clock tree.

5.6.2 Train/Test Splitting of Benchmarks

The training and testing split among the 11 designs utilized in this work is not performed in a purely random fashion. Instead, we strive to make the training set to be "comprehensive" that covers a variety of designs from small to large. One of the limitations of the proposed work is that the model is required to be pre-trained on a few designs, however, as shown in the experiment, after training on the 7 designs as shown in Table 9.2, GAN-CTS is able to achieve accurate prediction and high-quality prediction results on the largest design, *(*LEON), which also has the largest power consumption. The main reason is that GAN-CTS does not perform the optimization completely based on previous experience (seen designs). Instead, given a new design, it leverages unsupervised techniques to extract the underlying design features in order to generate high-quality clock trees. Finally, we expect the proposed framework to achieve lower MAPE/MAXE prediction error and better optimization results if a bigger a higher variety of training set is available.

5.6.3 Discussion of Prediction Results

As shown in Table 5.4, we observe that the MAXE (worst-case prediction error) is slightly high for the skew prediction. This in fact can be accounted in two-fold. First, as mentioned in section 5.4, the regression model is trained by least square regression [89], which minimizes the mean squared (L2) error to update the network parameters. It is known that the L2 error (loss) minimization tends to optimize the average prediction error across all samples that reaches stable solutions, where the L1 error minimization tends to optimize the error on outliers and thus results in unstable (sparse) solutions [101]. Therefore, as shown in the table, even the MAPE (average error) for skew prediction is bounded within 5%, some outliers still create corner cases that aggravate the MAXE metric. Second, as pointed out in previous works [79, 80], timing in general is a hard-to-predict metric due to the sophisticated behaviour of the commercial timing engines. In particular, during CTS, commercial tools will often override the skew target that is taken as input in order to optimize other metrics such as power and wirelength, which results in the uncorrelation between the target closure and the final achieved outcome.

5.6.4 Discussion of Optimization Results

The success of GAN-CTS on optimizing CTS metrics can mainly be explained in two-fold. First, instead of performing block-box optimization as the Bayesian optimization technique, GAN-CTS leverages the generator to learn the key distribution for different designs through conditional generative learning, which gives our framework better generality over other approaches. Second, the proposed transfer learning technique well differentiates various designs. The extracted features that contain precious design information help the framework to find better and more curated CTS parameter sets that result in optimized clock trees for unseen netlists.

5.7 Conclusion

In this paper, we have shown that machine learning offers promising solutions for designers to reach the desired CTS targets with small amount of effort. We have proposed a novel framework named GAN-CTS that uses discriminative techniques to predict and classify the CTS outcomes as well as leverages generative adversarial learning to optimize the desired metrics. Experimental results conducted on the unseen netlists demonstrate the proposed framework is generalizable and practical.

CHAPTER 6
RL-SIZER: VLSI GATE SIZING FOR TIMING OPTIMIZATION USING DEEP REINFORCEMENT LEARNING

6.1 Background and Motivation

Gate sizing for power, performance, and area (PPA) optimization is the backbone of modern physical design (PD) flows, which is used extensively from synthesis to signoff. It is an algorithmic process of assigning an appropriate size (gate type) to each optimizable design instance from a set of equivalent standard cell libraries under different process, voltage, and temperature (PVT) corners. For an instance, the number of available gate sizes is discrete and is limited by the underlying technology. This makes gate sizing an NP-hard problem [102], where the solution space scales exponentially with respect to the size of netlist.

Existing gate sizing algorithms in electronic design automation (EDA) tools are based on various pseudo-linear heuristics or analytical methods driven by (statistical) static timing analysis (STA) that easily result in globally sub-optimal sizing solutions. As the benefit of technology scaling saturates, leading edge high-performance low-power design flows are seeking to make the final PPA boost by leveraging more powerful sizing algorithms, even at the cost of runtime (or, increased turn-around time (TAT)), in order to achieve the desired PPA scalability at advanced process nodes. Therefore, time-to-market and best-in-class PPA requirements create a push-pull situation in EDA flows under advanced technologies (e.g., $16nm$ to $5nm$).

Reinforcement learning (RL) is a promising machine learning (ML) paradigm that has been demonstrated to achieve super-human performance in many high-dimensional control problems [103]. In the realm of EDA, a recent work [104] shows that how RL may be used for macro placement to improve design TAT and PPA. In addition, RL is applied to solve transistor sizing for analog designs [105], global routing [106], and technology mapping [107]. Nonetheless, we have to invent and engineer a different RL algorithmic framework to solve our gate sizing problem due to the significantly larger solution space compared with these previous works.

The goal of this work is to build the first high-dimensional RL framework, RL-Sizer, which formulates the classic gate sizing problem as an RL process and solves it by applying advanced RL algorithms equipped with graph neural networks (GNNs). To demonstrate the feasibility of the proposed RL formulation, we specifically focus on the problem of gate sizing for timing optimization at the post-route stage, where the goal is to optimize the total negative slack (TNS) of a design. Unlike prior works [108, 109] that perform aggressive optimization based on meta-heuristics or non-generalizable analytical methods that assume convexity of the objective functions, our RL agent optimizes design performance in a more global and flexible (i.e., customized loss function) manner with the consideration of design and technology features (multi-corner multi-mode) encoded by GNNs.

The outcome of our effort is a universal RL-based gate sizing framework that performs timing optimization across various advanced technologies for industrial-scale designs. To our knowledge, this is the first work that formulates the classic gate-sizing problem as an RL problem (control problem), and presents advanced RL algorithms equipped with graph representation learning techniques to solve it. The contributions of this work are as follows:

- We present RL-Sizer, the first-ever RL-based gate sizing algorithm for timing optimization. RL-Sizer achieves competitive TNS optimization results to an industry-leading commercial tool on six commercial designs using advanced technology nodes ($5nm, 12nm, 16nm$), and specifically, in four designs, RL-Sizer can significantly outperform the commercial sizing engine on TNS and number of violating endpoints (NVEs) with negligible total power overhead.

- We develop a graph-based feature encoder using GNNs that captures the instance characteristics related to timing optimization. These encoded features are taken as the inputs of RL-Sizer and are proven to be highly useful.

- We demonstrate the effectiveness of our "local-graph" method for fast timing approximation. For a target instance, we elegantly take the TNS change of its "local three-hop neighborhood graph structure" (termed as local-graph) as the reward of the sizing move taken instead of the entire netlist. This local-graph approximation can be easily threaded across various instances, which significantly accelerates the learning process.

6.2 Reinforcement Learning Formulation

6.2.1 Gate Sizing as a Control Problem

The gate sizing problem can be intuitively formulated as a Markov Decision Process (MDP), as there are many sequential decisions made iteratively regarding sizes of gates on critical (and, sometimes sub-critical) paths to achieve target timing closure. Therefore, we can conceptually apply RL algorithms to solve it (i.e., maximize the reward of this process). Given a set of design instances to be sized for timing optimization, we train an RL agent to sequentially determine their final gate sizes. Here we present key terminologies and concepts of an RL process, and illustrate how they are mapped to the gate sizing problem in our work.

- *State (s)*: A state s represents a "design instance", which is realized by concatenating the encoded features of its local three-hop neighborhood (by GNNs), and the technology features extracted from libraries.

- *Action (a)*: An action a refers to the "new gate size" assigned to the design instance in state s. In the implementation, it is realized as the "driving strength change" Δd. Assume an instance whose current size has strength d. After taking an action (a sizing move), it is assigned the gate size in the technology whose strength is the closest to $d' = d + \Delta d$ among all possible choices.

- *Reward (r)*: A reward r is the outcome of performing an action a on an instance in state s. In our case, it represents the TNS change of the instance's "local-graph". For each sizing iteration, the goal of RL-Sizer is to maximize the *total reward* (sum of individual rewards) of all instances.
- *Trajectory (τ)*: A trajectory τ refers to a sizing iteration (an RL process), from time step $t = 0$ to $t = T$ (final time step). 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). Note that a complete gate sizing run in commercial tools consists of multiple trajectories.

Figure 6.1 shows an illustration of our RL gate sizing process, where we consider each selected instance as a unique *RL state* and determine their new gate sizes sequentially (the instance selection algorithm is illustrated in section 6.3). Note that STA update using a commercial tool is performed once at the end of a sizing iteration, which provides the *RL reward* for each action taken. We want to emphasize that instances in a common sizing iteration (*RL trajectory*) are not independent of each other. Actions (gate sizes) that are taken in previous time steps (prior instances) will contribute to the sizing decision of the current time step. We leverage a *policy gradient* algorithm named *Deep Deterministic Policy Gradient* (DDPG) [25] to capture this dependency and to optimize the total reward.

6.2.2 Our Key Concept: Local-Graph Approximation

As mentioned earlier, we propose the concept of "local-graph" for *RL state* encoding (Figure 6.1(b)) and *RL reward* approximation. Given a target instance, the "local-graph" of this instance refers to its local "three-hop neighborhood graph structure" from the netlist. The rationale is two-fold. First, the final gate size of a target instance not only depends on the characteristics of itself, but also the behavior of its neighbors (e.g. the capacitive load

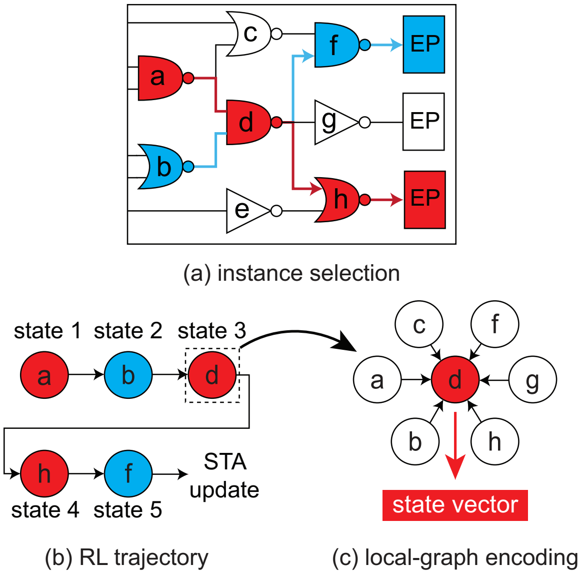


Figure 6.1: Illustration of our RL gate sizing process. (a) Input netlist with 3 end points (EPs). First, we identify the worst critical path in the design (red), and then for each endpoint, we identify the most negative slack path (e.g. blue) overlapping with the design critical path. Finally, instances on these paths (the design critical path and the other paths overlapped with it) are selected for one sizing iteration. (b) Sort the selected instances in topological order, and determine their final gate sizes sequentially by considering each of them as an *RL state*. STA is performed after all selected instances are assigned new sizes. (c) Example of local-graph encoding using GNNs on gate "d". The encoded state vector is taken as the input of the RL agent to determine the action (new gate type).

that this target instance is driving). We leverage GNNs (to be elaborated in Section subsection 6.3.3) to encode such neighboring information into a vector as an *RL state* vector, which serves as the input of the RL agent for the decision of the corresponding *RL action* (i.e., new gate size).

Second, the timing impact of a gate sizing move on a design instance to the overall netlist diminishes as the hop count increases. Therefore, instead of taking the total design TNS change as the *RL reward* of a sizing move (ideal case, but computationally expensive), we take the TNS change of its local-graph. This way, the reward gives fast and good fidelity approximation, while offering an opportunity for parallel computation. That is, an improvement in local-graph TNS mostly results in positive design (netlist) TNS change, and vice versa.

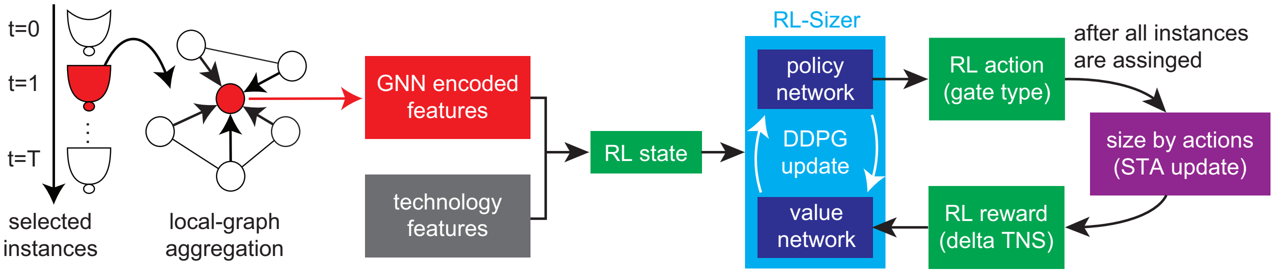


Figure 6.2: Overview of our RL-Sizer framework. Given the selected instances from Algorithm Algorithm 9, for each instance (e.g. red), we take its encoded local-graph features along with the technology information as the *RL state* s_t, and leverage RL-Sizer to determine the *RL action* a_t assigned. An STA update is performed when all selected instances are assigned new gate sizes. Finally, we take the "local-graph TNS change" as the *RL reward* of each action taken. Rewards across time steps (instances) are leveraged to update RL-Sizer through the DDPG algorithm [25].

6.2.3 Graph Representation Learning

GNNs have revolutionized many research areas [37] by performing effective graph representation learning that encodes graph information into meaningful embeddings through a message passing scheme. Since VLSI netlists are represented as hypergraphs, we can apply GNNs to them. Recently, many studies have demonstrated the great potency of applying GNNs to solve EDA problems, such as transistor sizing [105], layout decomposition [110], power estimation [**lufast**, 14], and circuit partitioning [6]. We leverage GNNs to distill netlist features that are related to timing. Specifically, given a target instance for sizing, we utilize GNNs to encode the features within its local-graph (3-hop neighborhood), and take the encoded features as the input of the RL framework to determine its final gate size.

6.3 RL-Sizer Algorithms

In this work, we focus our problem on the post-route stage, which is the PD stage that designers struggle the most for timing optimization. However, our method generalizes to other stages of the PD flow as well. The goal of our framework, RL-Sizer, is to optimize the design performance in terms of TNS by making good sizing moves on combinational instances. Note that we do not size sequential instances.

6.3.1 Overview

Figure 6.2 shows a high-level overview of our framework. First, we develop an instance selection algorithm to select the combinational instances that must be sized to improve design TNS. These selected instances form a sizing iteration (an *RL trajectory*). Note that a complete gate sizing run consists of multiple sizing iterations. For each selected instance, we use GNNs to encode its "local-graph" (described in section 8.4), and take the encoded features along with the technology features that represent the driving strength, capacitance, and slew constraints as the *RL state* s_t. We define the corresponding *RL reward* r_t subject to the *RL action* a_t taken at time step t as the TNS change on its local-graph. Note that as aforementioned, each selected instance belongs to a unique time step and is sized sequentially from time step $t = 0$ to $t = T$ (last instance). This order is based on netlist topology.

At each time step t, the objective of RL-Sizer is to maximize the long-term return G_t, which is denoted as

$$\max_{\theta} G_t(\pi_\theta) = \mathbb{E}_\tau \left[\sum_{k=0}^{T} \gamma^k r_{t+k} \right], \tag{6.1}$$

where π denotes the policy function (network) parameterized by θ, which takes the state s_t as input and outputs the action a_t, γ denotes the reward discount factor. To maximize this objective G, we perform gradient descent on the policy parameters θ using the DDPG [25] loss function update. In the following sub-sections, we present each component of our framework in detail.

6.3.2 Instance Selection

Selecting feasible instances that can possibly improve design TNS is essential to the success of RL-Sizer. Algorithm Algorithm 9 presents our instance selection process. Given a routed design $G = (V, E)$, where V denotes the design instances and E denotes the connections, our algorithm identifies the target instances $V' \in V$ that will further be sized sequentially

Algorithm 9 Instance selection for a sizing iteration (*RL trajectory*).

```
Input: $G = (V, E)$: a post-route netlist.
Output: $V' \in V$: selected instances to be sized.
Run full-chip STA.
$W \leftarrow$ current worst negative slack (WNS) path in the design
Initialize $V' \leftarrow \{$non-overlapping instances in $W\}$
$\{P\} \leftarrow$ for each endpoint, identify its worst negative path
for $p \in \{P\}$ do
    if $p$ is overlapping with $W$ then
        for $v \in p$ do
            if $v$'s local-graph does not overlap with $\{V'\}$'s then
                add instance $v$ on path $p$ to set $V'$
$V' \leftarrow$ topological_sort($V'$)                ▷ linear time, achieved by DFS
```

by RL-Sizer in (netlist) topological order.

Note that as shown in the algorithm, selected instances in V' do not share "overlapping" local-graphs, which means instances in a sizing iteration do not overlap in their local 3-hop neighborhood. This is to minimize the sizing impact between each other, since in our settings (as shown in Figure 6.2), instances in a common iteration are sized simultaneously (i.e., an STA update is performed once per iteration). Ideally, one can perform an STA update per sizing move of an instance to completely address the issue of interference. However, this approach is impractical due to the computation expense of STA on VLSI designs. In our experiments, we find that with the proposed technique of "non-overlapping local-graphs", RL-Sizer can effectively determine the feasible size for each selected instance that optimizes TNS.

6.3.3 Encoding RL State using GNNs

Initial Node Features

Prior to the local-graph encoding using GNNs, we compute the initial node-specific features for each design instance as shown in Table 9.1. These features are carefully chosen based on domain expertise and are expected to characterize an instance's sizing impact to design timing. However, these features are not sufficient for RL-Sizer to determine the

Table 6.1: Initial node features for GNN encoding.

features	descriptions
slack	worst slack of paths through instance
in_slew	worst input pin slew
out_slew	output pin slew
arc_delay	worst cell arc (input to output pin) delay
nom_delay	nominal delay (fan-out of 4)
cell_cap	cell capacitance
drv_length	driving (output) net length
drv_load	sum of driving capacitance (net + cell)
drv_res	sum of driving resistance
fanin_cap	average capacitance of fan-ins
sibling_cap	sum of capacitance of siblings

gate sizes that optimize design performance, because the final gate size of a target instance not only depends on these features, but also the information from its neighboring nodes. Therefore, we use GNNs as a local-graph encoder to obtain better representations in graph-level. Note that the initial features are not normalized instance-wise, since for each sizing iteration, we select a new set of instances to be sized.

Local-Graph Encoding

Based on the initial node features defined in Table 9.1, we leverage GraphSAGE [37], a variant of GNNs, to encode local-graph features for each selected instance. Given a local-graph sG of a target instance v, for each node $v' \in sG$, we first transform the initial node features $h^0_{v'}$ into embeddings at level $k = K$ as:

$$\begin{aligned} h^{k-1}_{N(v')} &= mean\_pool\left(\{\mathbf{W}^{agg}_k h^{k-1}_u, \forall u \in N(v')\}\right), \\ h^k_{v'} &= sigmoid\left(\mathbf{W}^{proj}_k \cdot \text{concat}\left[h^{k-1}_{v'}, h^{k-1}_{N(v')}\right]\right), \end{aligned} \tag{6.2}$$

where $N(v')$ denotes the neighbors of node v', W^{agg} and W^{proj} denote the aggregation and projection matrices that are achieved by neural networks (neurons). At the end of the transformation (level K), we take the mean pooling of $h^{k=K}_{v'}$ across every node $v' \in sG$ to obtain the final local-graph feature vector s_t of the target instance v at time step t as:

$$s_t = \text{concat}\left[mean\_pool\left(\left\{h_{v'}^{k=K}\right\}\right), tech(v)\right], \tag{6.3}$$

where $tech(v)$ denotes the technology features (from library files) of instance v in terms of driving strength, capacitance, and slew constraints of the current gate size. This vector s_t, which characterizes the local-graph and the underlying instance, is taken as the input of the RL-Sizer agent to determine the new gate size that helps improve the design performance. Note that the dimension of the GNN-encoded vector $h_{v'}$ is subject to the number of neurons in the last layer of the GNN module, which is 64 in our implementation.

6.3.4 Policy and Value Networks

We use DDPG [25], a variant of actor-critic algorithms, to build RL-Sizer. All actor-critic algorithms have two components that learn jointly: *actor* and *critic*. In deep RL (RL powered by neural networks), *actor* refers to the policy network which learns a parameterized policy $\pi(s)$ that maps a state vector s to an action a. Next, *critic* refers to the value network which learns a value function $Q(s, a)$ that evaluates the (discounted) reward of taking an action a on a state s.

Algorithm Algorithm 10 presents the training process of RL-Sizer based on the DDPG [25] loss function update. In DDPG, the learning update of the Q-function Q is based on the Bellman equation, which suggests the Q-value $Q(s, a)$ at current state s to be computed in a dynamic programming manner as

$$Q(s_t, a_t) = \mathbb{E}\left[r_t + \gamma * \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})\right]. \tag{6.4}$$

In DDPG, the goal of the policy network π is to generate the action a_t subject to the state s_t that maximizes the Q-value $Q(s_t, a_t)$. The idea is that the higher the Q-value $Q(s, a)$ is,

Algorithm 10 RL-Sizer training methodology.

```
Input: Initial Policy Network parameters $\theta_\pi$, Initial Q Network parameters $\theta_Q$, Target networks
    update ratio $\rho$, Netlist $G = (V, E)$
Output: Policy Network parameters $\theta_\pi$; Q Network parameters $\theta_Q$
Initialize target networks (policy-, Q-) parameters $\{\phi\}$ as $\phi_\pi \leftarrow \theta_\pi$, $\phi_Q \leftarrow \theta_Q$, Replay Buffer
    $B \leftarrow \{\}$
while design TNS does not converge do
    $\{V'\} \leftarrow$ instance_selection$(G)$                          ▷ Algorithm Algorithm 9
    $\{s\} \leftarrow$ local-graph_encoding$(V')$                       ▷ Equation 9.1, Equation 9.2
    $T \leftarrow |s|$                                                  ▷ # of states (instances)
    for $t = 0;\ t < T;\ t++$ do                                        ▷ Assign actions for all cells
        $a_t \leftarrow \pi(s_t|\theta_\pi)$
    Perform actions $\{a\}$ and STA update to get rewards $\{r\}$
    Store all $(s_t, a_t, r_t, s_{t+1})$ pairs in the replay buffer $B$
    Sample a batch of T buffers $\{(s_t, a_t, r_t, s_{t+1})\}$ from $B$
    for $t = 0;\ t < T;\ t++$ do                                        ▷ Compute update targets $y$
        $y_t \leftarrow r_t + \gamma * Q_{\phi_Q}\left(s_{t+1}, \pi\left(s_{t+1}|\phi_\pi\right)\right)$
    Update Q Network $\nabla_{\theta_Q} \sum_t (Q_{\theta_Q}(s_t, a_t) - y_t)^2$
    Update Policy Network $\nabla_{\theta_\pi} \sum_t Q_{\theta_Q}(s_t, \pi(s_t|\theta_\pi))$
    $\phi_\pi \leftarrow \rho\phi_\pi + (1-\rho)\theta_\pi$
    $\phi_Q \leftarrow \rho\phi_Q + (1-\rho)\theta_Q$                   ▷ Temporal difference update
```

the better the action a is. The objective of the policy network π can thus be formulated as

$$\max_{\theta_\pi} E\left[Q(s_t, \pi(s_t|\theta_\pi))\right], \tag{6.5}$$

where $\pi(s_t|\theta_\pi))$ is the action output by the policy network π based on the encoded state vector s_t.

As shown in the algorithm, both the value and policy networks are trained by a technique named *temporal difference update*, where for each network, we maintain a "target network" (with parameter ϕ) whose update is a trajectory slower than that of the main network (with parameter θ). For example, if the main network is updated in τ_i, then the the target network is updated in τ_{i+1}. By using a replay buffer B that contains old experiences from previous trajectories, the temporal difference update is expected to stabilize the training process. Finally, when the training completes, we obtain an *actor*, the policy network π, that performs the gate sizing moves to improve design performance. Figure Figure 6.3

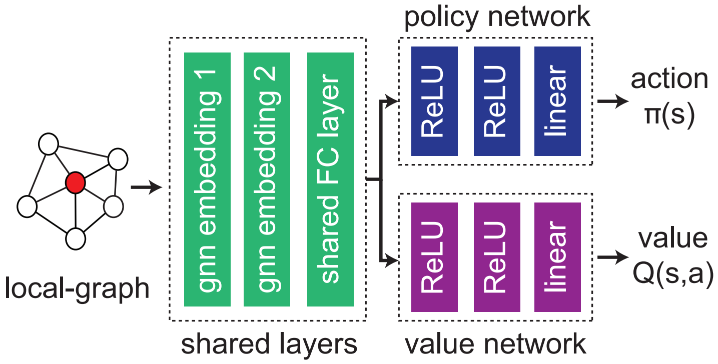


Figure 6.3: Our RL agent architecture that consists of value and policy networks. Table Table 6.2 provides the dimension information.

Table 6.2: Dimension of RL-Sizer layers.

component	input	hidden	output
shared layers	local-graph G=(V,E)	(64, 64) (GNN)	64 (FC)
policy network	64 (shared)	(64, 32) (ReLU)	1 (action)
value network	65 (shared + action)	(64, 32) (ReLU)	1 (value)

further shows the architecture of our RL agent that utilizes the value and policy networks.

6.3.5 Implementation Details: Challenges of ML in EDA

Our framework, RL-Sizer, is implemented in the source code of *Synopsys IC-Compiler II* (ICC2). Due to the fact that EDA tools are generally implemented in C++, while machine learning frameworks are mainly supported in Python, one of our main challenges is to communicate between these two language interfaces, since the communication introduces costly runtime overhead. Ideally, at each time step t of a sizing iteration, we can perform an action a_t on an instance and calculate the reward r_t immediately, so that instances in a common iteration will not interfere with each other. However, in our implementation, this ideal approach is not feasible considering the sizes of VLSI netlists. Even with the proposed technique of local-graph approximation, the runtime will still explode due to the communication overhead between C++ and Python. Therefore, to make the proposed framework practical, we only perform a full-chip STA update (i.e., switching from Python to C++) when all selected instances in a common iteration are assigned actions by the

Table 6.3: Our commercial benchmarks and their attributes.

Design	Tech. Node	# Nets	# Macros	# Instances
block1	5nm	93,370	0	95,636
block2	5nm	145,893	0	151,258
block3	12nm	145,545	0	142,528
block4	12nm	430,141	35	462,755
block5 (SoC)	16nm	36,783	9	6,850
block6	16nm	72,748	0	71,604

policy network (Lines 6–7 in Algorithm Algorithm 10). After calculating all the rewards $\{r\}$ in the commercial tool, we again switch from C++ to Python and leverage gradient descent to update the network parameters.

6.4 Experimental Results

In the experiments, we validate the proposed framework on 6 commercial designs (renamed due to confidentiality) in advanced technology nodes as shown in Table 9.2, and demonstrate how RL-Sizer improves the native sizing algorithms in Synopsys ICC2, an industry-leading commercial tool.

6.4.1 Optimization Results

Table 7.2 demonstrates the optimization results on our benchmarks as shown in Table 9.2, where we observe that RL-Sizer can outperform or match the optimization results achieved by the reference commercial tool. Note that this is a head-to-head comparison, where the objectives of RL-Sizer and the commercial tool are exactly the same, which is optimizing design TNS through combinational sizing at the post-route stage. In RL-Sizer, we run Algorithm Algorithm 10 for each design from scratch (i.e., the policy and value networks are trained from the beginning). We terminate the algorithm when design TNS no longer improves across 10 consecutive iterations. The results suggest that RL-Sizer is able to generalize across various designs and technology nodes. It also generalizes for both macro-heavy and macro-less designs.

Table 6.4: TNS optimization results comparison between RL-Sizer and Synopsys ICC2. The unit for timing is ns, and the unit for power is mW. WNS denotes worst negative slack; TNS denotes total negative slack, and #vio. EPs denotes the number of violating endpoints. The runtime for the commercial tool and RL-Sizer is measured on the same machine without GPU support.

Design (tech)	Initial (post-route stage)				Synopsys ICC2					RL-Sizer (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 (5nm)	-0.08	-61.99	2191	68.4	-0.08	-46.68	1728	68.7	30m	-0.07	-44.7	1631	68.8	6hr
block2 (5nm)	-0.05	-101.82	1305	204.4	-0.05	-1.19	182	205.3	1hr	-0.04	-0.81	116	205.7	14hr
block3 (12nm)	-0.31	-357.72	7934	44.2	-0.02	-0.07	4	44.9	1hr	-0.07	-0.37	39	45.1	15hr
block4 (12nm)	-0.22	-523.75	18845	123.6	-0.21	-8.71	348	123.9	1hr	-0.20	-8.11	201	124.1	22hr
block5 (16nm)	-0.80	-104.74	430	718.2	-0.79	-90.13	383	743.0	10m	-0.78	-80.54	379	718.4	5hr
block6 (16nm)	-0.15	-46.84	1377	25.2	-0.02	-0.03	5	25.5	24m	-0.04	-0.68	74	26.0	6hr

Figure 6.4 further shows the design TNS after each sizing iteration (*RL trajectory*) of RL-Sizer on "block2" ($5nm$). Although the entire training process takes about 14 hours (250 iterations) to reduce TNS from -101.82ns to -0.81ns, it takes less then 3 hours (13 iterations) to quickly recover the initial TNS to -2.18ns. Note that the runtime in the table (and above) for both commercial tool and RL-Sizer is measured on the same machine without GPU support, and we do not limit the runtime of the commercial tool in order to perform thorough optimization (i.e., the tool stops the sizing optimization when the timing can no longer be improved). As for RL-Sizer, the stopping mechanism is aforementioned, and we expect the runtime to be significantly improved when GPUs are utilized.

6.4.2 Discussion of Optimization Results

The fact that RL-Sizer is able to perform commercial-grade timing optimization results on 6 different designs demonstrates its generality. However, we observe that RL-Sizer does not always outperform the commercial tool even though it adopts a more global approach to perform the optimization. This inferiority in fact happens on the designs that both the commercial tool and RL-Sizer are able to optimize the design TNS to "near zero". However, since RL-Sizer lacks rigid heuristics to "close design timing" as the commercial tool, the optimization results stagnate when no sizing action leads to positive reward. Ways to locally improve optimization results so as to completely close design timing are the areas for our future investigation.

Our further analysis reveal the following regarding the success of our RL-Sizer compared with the commercial tool:

1. We accept "setback" moves: the goal of RL-Sizer is to maximize the total reward (i.e., sum of individual rewards) of a given iteration. Instead of striving to completely fix the entire slack violation for each selected instance, at some states, RL-Sizer learns to "setback" to create more sizing room for future states in order to achieve a global optima.

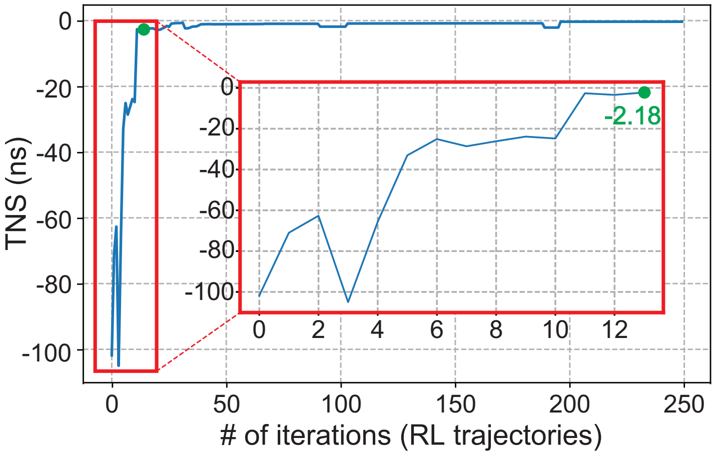


Figure 6.4: RL-Sizer sizing iterations on block2 (5nm). It takes 250 iterations (about 14 hours) to improve TNS from -101.82 to -0.81 (ns). However, the TNS quickly converges in the first 13 iterations (less than 3 hours).

2. Our well-defined features: the initial node features in Table 9.1 accurately characterize the sizing behaviour of each instance. Despite these features are not sufficient to determine the final gate size of an instance, with the aid of the graph representation learning, they provide vital information for policy and value networks to effectively determine the sizes that optimize design performance.

Finally, we want to emphasize that although the main focus of this work is timing optimization, our framework can be extended to jointly optimize other PPA metrics such as power and area by incorporating them in the reward calculation (e.g., $r = \Delta_{timing} + \alpha * \Delta_{power} + \beta * \Delta_{area}$).

6.5 Conclusion

Several prior works have made significant progress to improve VLSI gate sizing. In this chapter, we take a new approach to solve the well-studied gate sizing problem using novel RL algorithms. We propose RL-Sizer, a GNN-powered policy gradient-based framework that performs automatic gate sizing for timing optimization without any human intervention. We believe the achieved optimization results shall demonstrate the great potentials of leveraging RL algorithms to solve classic EDA problems.

CHAPTER 7
RL-CCD: CONCURRENT CLOCK AND DATA OPTIMIZATION USING ATTENTION-BASED SELF-SUPERVISED REINFORCEMENT LEARNING

7.1 Background and Motivation

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 chapter, 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 [111]. 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

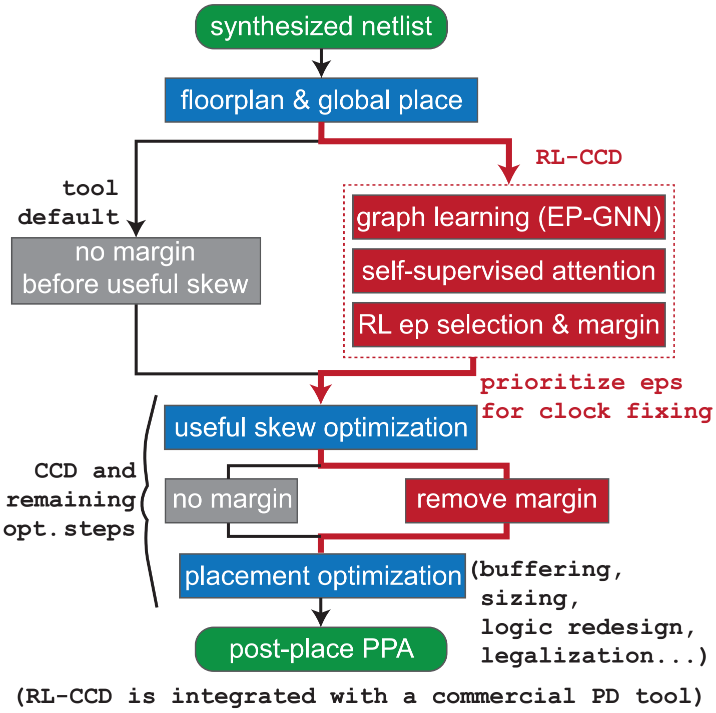

synthesized netlist
floorplan & global place
RL-CCD
tool default
no margin before useful skew
graph learning (EP-GNN)
self-supervised attention
RL ep selection & margin
prioritize eps for clock fixing
useful skew optimization
CCD and remaining opt. steps
no margin
remove margin
placement optimization
(buffering, sizing, logic redesign, legalization...)
post-place PPA
(RL-CCD is integrated with a commercial PD tool)

Figure 7.1: Default tool flow vs. our RL-enhanced flow that performs endpoint prioritization using graph learning and self-supervised attention [111].

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 (name will be disclosed upon acceptance), and demonstrate that RL-CCD significantly improves the tool's native implementation flow.

Figure 7.1 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 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.

7.2 Related Works

7.2.1 Predictive Useful Skew

Useful skew is a well-known technique that improves design timing by adjusting clock arrival time. However, as pointed out in [113], computed skew adjustments often require redo synthesis or placement to truly realize timing benefits. To break the chicken-egg problem

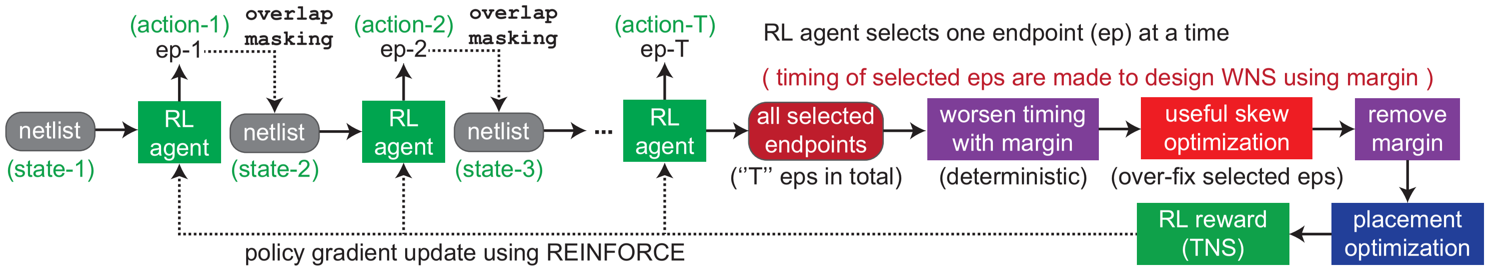


Figure 7.2: High-level overview of our framework. At each RL time step (training iteration), our agent selects one endpoint at a time and mask out other endpoints based on overlapping calculation. The selection process completes when all violating endpoints are either masked or selected. Then, with the 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 [112].

(i.e., iterative back annotation of skew), the authors of [114] proposed a "predictive" useful skew technique for one-pass timing optimization, where the TNS value can be improved by up to 5%. In this work, 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.

7.2.2 Learning-Driven Timing Prediction

Fast and accurate timing prediction methods are essential to improve design productivity. Previous work [61] 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 [60] presented a timing engine inspired GNN-based framework for slack and arrival time prediction on timing endpoints. Recently, the authors of [115] explored the use of Transformer [111] 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 work, we are not comparing RL-CCD with above methods as the focus of RL is to improve optimization quality rather than prediction accuracy.

7.2.3 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 [112]. In PD, the authors of [9] developed an RL agent for floorplanning, which generates superhuman floorplans that human labor is not able to achieve. Another work [11] proposed RL-Sizer to tackle the VLSI gate sizing problem, a traditional EDA optimization task. It is shown that RL-Sizer can outper-

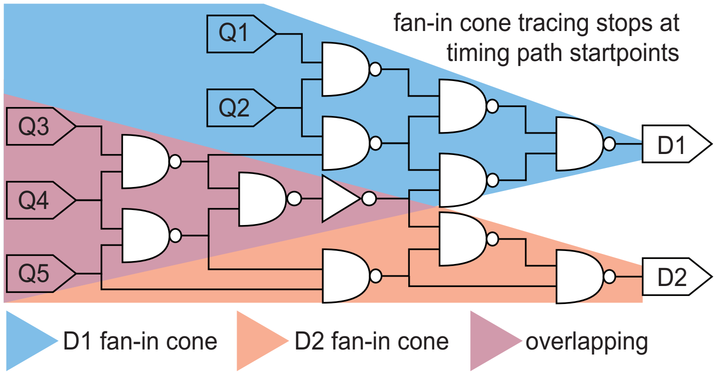


Figure 7.3: Illustration of endpoint fan-in cone overlapping. Note that the fan-in cone tracing of an endpoint stops at its previous startpoints. The overlapping ratio is calculated as dividing the number of overlapped cells by the total number of fan-in cone cells.

form the default sizing algorithms in a commercial tool although it adopts a fundamentally different approach. With the above success stories, in this work, 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.

7.3 RL-CCD Algorithms

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 work, we demonstrate that by selecting proper V', the achieved TNS value can be drastically improved.

7.3.1 Overview and Reinforcement Learning Formulation

Figure 8.1 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

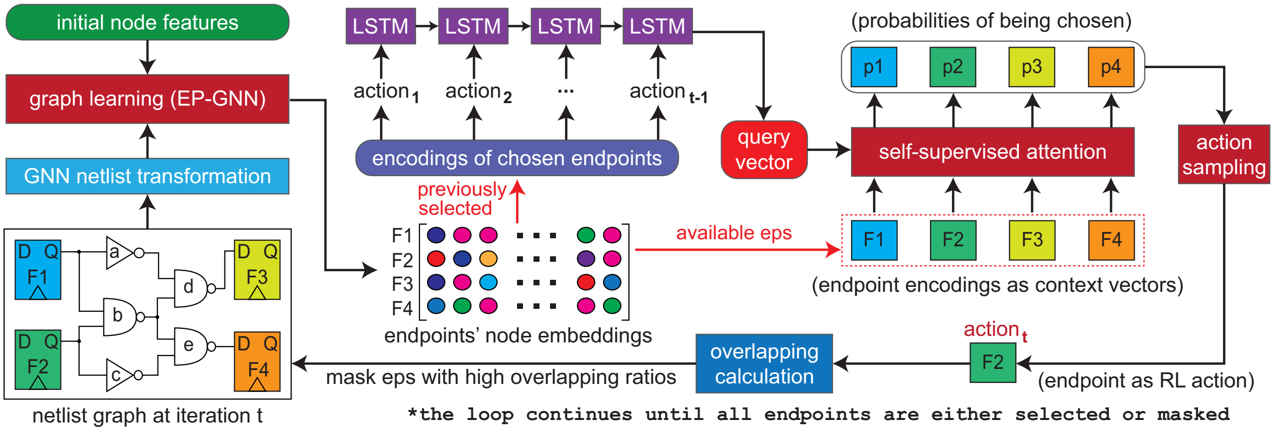

initial node features
graph learning (EP-GNN)
GNN netlist transformation
netlist graph at iteration t
D Q
F1
F2
F3
F4
a
b
c
d
e
LSTM
action1
action2
...
actiont-1
encodings of chosen endpoints
previously selected
endpoints' node embeddings
query vector
self-supervised attention
(probabilities of being chosen)
p1
p2
p3
p4
available eps
(endpoint encodings as context vectors)
action sampling
actiont
(endpoint as RL action)
overlapping calculation
mask eps with high overlapping ratios
*the loop continues until all endpoints are either selected or masked

Figure 7.4: 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 [111], 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 7.3). The features (Table 7.1) are updated accordingly and the loop continues until all endpoints are either selected or masked.

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 work, we choose to formulate it as a Markov 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:

$$\max_{\theta} J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)], \tag{7.1}$$

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

7.3.2 Detailed Architecture

RL-CCD consists of three main components: (1) a GNN module that generates endpoint embeddings, (2) a Long Short-Term Memory (LSTM) [116] 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 7.4 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:

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 work, we present EP-

Table 7.1: 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

GNN, an endpoint-oriented GNN framework that focuses on generating node embeddings of timing path endpoints through iterative neighborhood and fan-in cone aggregation.

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 7.1, 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:

$$f_v^l = \sigma\left(\gamma f_v^{l-1} \cdot \Theta_{proj} + (1-\gamma) \cdot \Theta_{agg}\left(\frac{1}{|N(v)|}\sum_{j \in N(v)} f_j^{l-1}\right)\right), \qquad (7.2)$$

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:

$$f_e = \Theta_{FC}\left(f_e^{l=3} + \sum_{j \in cone(e)} f_j^{l=3}\right), \tag{7.3}$$

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 7.1) 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).

Past Actions Encoding using LSTM

Our encoder-decoder structure as shown in Figure 7.4 is inspired by the renowned Transformer architecture proposed in [111]. In this work, 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 [116], 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:

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, a_{t-1}] + b_i), & f_t &= \sigma(W_f \cdot [h_{t-1}, a_{t-1}] + b_f), \\ o_t &= \sigma(W_o \cdot [h_{t-1}, a_{t-1}] + b_o), & \tilde{c_t} &= tanh(W_c[h_{t-1}, x_t] + b_c), \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c_t}, & h_t &= o_t \odot tanh(c_t), \qquad q_t = h_t \end{aligned} \tag{7.4}$$

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 $t+1$.

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 work, we leverage a self-supervised attention mechanism [111] to build the decoder network. Inspired from from pointer networks [117], given a query vector q_t and GNN embeddings $F_{EP}^{(t)} \in R^{|EP|\text{x16}}$ 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:

$$A_t^{(i)} = \begin{cases} v^T \text{tanh}\left(W_1 \cdot F_{EP}^{(t)} + W_2 \cdot q_t\right) & \text{if ep-}i\text{ is valid} \\ -\infty & \text{otherwise,} \end{cases} \tag{7.5}$$

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 7.5 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:

$$P_t^{(i)} = softmax(A_t^{(i)}) = \frac{e^{A_t^{(i)}}}{\sum_k A_t^{(k)}}, \forall i \in EP. \tag{7.6}$$

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 7.5. 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.

7.3.3 Fan-in Cone Overlap Masking and the Rationale Behind

As shown in Figure 7.4, RL-CCD selects endpoints sequentially and the selection process completes when all endpoints in the design are either masked or selected. Our masking

Algorithm 11 RL-CCD training methodology. We use $\rho = 0.3$ as default.

```
Input: Netlist $G = (V, E)$, Initial EP-GNN parameters $\theta_{gnn}$, Initial LSTM encoder parameters
    $\theta_{LSTM}$, Initial attention-based decoder parameters $\theta_{attn}$, Overlapping threshold $\rho$, 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 Equation 7.2, Equation 7.3
 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 Equation 7.5, Equation 7.6
10:     $a_t \leftarrow$ sample one endpoint from $P_t$                        ▷ selected ep
11:     $G, EP \leftarrow$ overlap_masking$(G, EP, a_t, \rho)$                  ▷ 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
```

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 7.3. 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 [61], 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.

7.3.4 Training Methodology

In this work, we leverage REINFORCE [112], a renowned policy gradient algorithm, to train our RL-CCD framework. The objective of our RL agent is defined in Equation 7.1, 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 [112] that the gradient of the objective J can be derived as:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T} R(\tau) \nabla_\theta \log \pi_\theta \left(a_t | s_t \right) \right], \tag{7.7}$$

where in our settings, $R(\tau)$ is the achieved TNS value after completing the entire placement optimization. Equation 7.7 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 Algorithm 11 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.

7.4 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-

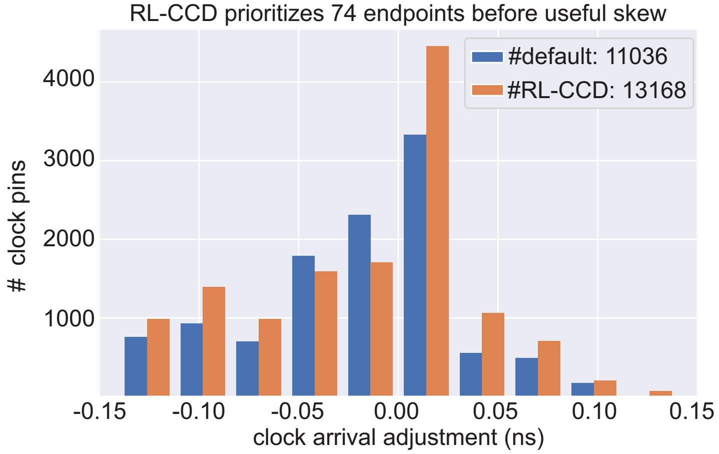


Figure 7.5: Histogram of clock arrival adjustments on block11 (180K cells). Each pair of juxtaposed color bars has the same range of arrival values.

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 7.1, 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.

7.4.1 Single-Design Optimization Results

Table 7.2 demonstrates the optimization results achieved by training Algorithm 11 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

Table 7.2: 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	begin (post global place)				default tool flow (16 threads)					RL-CCD enhanced (ours)				
(# cells)	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%	

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 7.5 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.

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 work, 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.

7.4.2 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

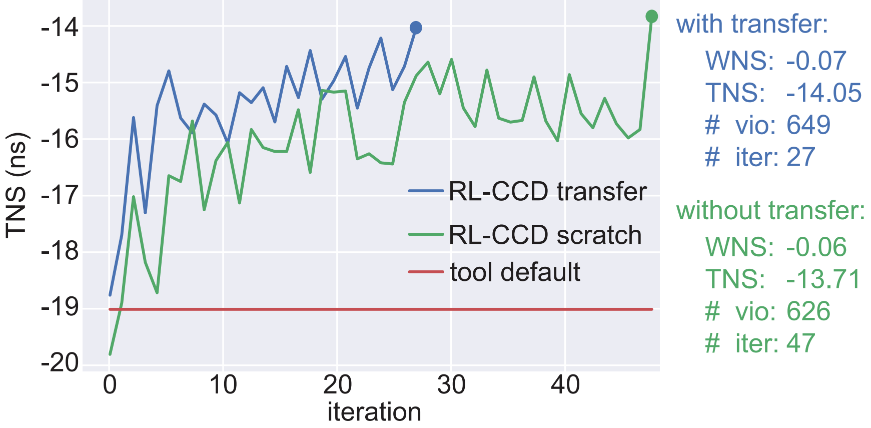


Figure 7.6: Transfer learning on block19 (922K cells) by using a pre-trained EP-GNN model, where comparable optimization results is achieved in a much faster convergence rate.

unseen ones. In this work, 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 11) on "unseen" designs. Figure 7.6 shows the transfer learning results on block19 (922K cells). It is shown that with transfer learning, RL-CCD can quickly converge to comparable optimization results compared with training the entire framework (i.e., EP-GNN + encoder-decoder) from scratch as in Table 7.2. 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 7.6.

7.4.3 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.

7.5 Conclusion

In this chapter, 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.

CHAPTER 8
ECO-GNN: SIGNOFF POWER PREDICTION USING GRAPH NEURAL NETWORKS WITH SUBGRAPH APPROXIMATION

8.1 Background and Motivation

Handheld and wearable devices are significantly proliferating in today's semiconductor markets and demand extremely low power dissipation while operating at voltages as low as $0.45V$. However, in advanced technology nodes, power optimization has become much more complicated than optimizations on other design metrics such as wirelength and delay, which is due to the dominance of leakage power and its complicated relation with the dynamic power [118]. It is well known that leakage power increases exponentially with the scaling of threshold voltage (V_{th}). Therefore, low-voltage design in advanced technology nodes such as $7nm$, $5nm$ and below require design implementation tools to aggressively optimize leakage power at various stages of commercial physical design (PD) flows, which are often achieved by Engineering Change Orders (ECOs) that involve gate-sizing and V_{th}-assignment. In this chapter, we specifically focus on improving the power ECO at the signoff stage, which is more time-consuming than the power optimizations at other PD stages because it requires a precise calculation of the delay budget.

In modern chip design flows, ECOs are performed extensively from synthesis to signoff with an aim to optimize power, performance and area (PPA) metrics. Every top semiconductor design company runs multiple iterations of signoff ECO to achieve the target PPA. However, in advanced technology nodes, power optimization has become much more complicated than optimizations on other design metrics such as wirelength and timing, which is mainly due to the dominance of leakage power and its complicated relation with the dynamic power [118]. Even though design implementation tools have developed various power optimization techniques throughout the years, designers still heavily rely on signoff tools to recover power at the signoff stage using ECO change-lists. Nonetheless, since power ECO in signoff tools requires accurate timing budget calculation (e.g., path-based timing analysis) during the optimization, it is extremely time-consuming and thus bottlenecks the chip design process. Therefore, in this work, we aim to develop a learning-based framework, ECO-GNN, that has the ability to perform signoff power prediction to improve the chip design turn-around time.

Gate sizing and V_{th}-assignment are the two popular techniques to optimize design power consumption. However, since gate-sizing requires further legalization and routing to validate the design after the optimization, V_{th}-assignment is the preferred approach during signoff ECO as it causes minimum disturbance to the overall placed and routed layout. In *Synopsys PrimeTime*, V_{th}-assignments during signoff ECO not only optimize the leakage power, but also reduce the dynamic power simultaneously [119]. Nonetheless, this optimization conducted by *PrimeTime* is time-consuming and the tool itself remains a blackbox for designers. Therefore, in this work, our goal is to develop a fast, explainable signoff power optimization framework that has the ability to perform commercial quality signoff power optimization instantly as well as the facility to explain the achieved optimization results.

V_{th}-assignment refers to assigning an appropriate V_{th} type for each design instance from a set of standard cell libraries to perform power optimization without violating timing constraints [120]. Note that for a given design instance, all the available V_{th} types have the same footprint, and the total number of the available types is limited to the discrete values of threshold voltages specified by the technology. This optimization problem is proven to be NP-hard [102], which means the optimality of a sizing solution is hard to be demonstrated and implies great opportunities to employ machine learning techniques for solving this problem.

Modern commercial signoff tools perform signoff power ECO based on sophisticated in-house timing models. The models precisely calculate the timing budget for every design instance to help the signoff engines conduct timing-constrained power optimization. The optimization results achieved by these tools are considered as golden QoR in the industry, however, there are two significant drawbacks in the current industrial signoff flows, namely:

- **Extremely long runtime.** A signoff power ECO run often takes several days on an industrial scale design and requires human-in-the-loop for enhancement, which drastically bottlenecks the chip development process.

- **Obscure improvement.** The power improvement is unknown in advance. Designers tend to run multiple optimization configurations in parallel in order to select the best one in the end, which consumes significant amount of computing resources.

- **Partial netlist update.** In many real-world scenarios, designers would only want to perform the signoff power optimization on a few selected instances with positive slack margin in order to prevent severe timing degradation. However, the improvements that can be generated from these instances are often unclear until after spending significant amount of time in ECO iterations.

In this work, we overcome the above issues by presenting ECO-GNN, which is a graph-learning-based framework that leverages graph neural networks (GNNs) to perform V_{th}-assignments for fast signoff power optimization [12]. Specifically, we present two approaches to overcome the issues. First, we present a classification-based technique to predict the final V_{th}-assignment of each design instance that will be made by *PrimeTime* during the ECO using the information of the entire netlist. Second, we further propose a "subgraph approximation" technique to demonstrate the ability of our framework on predicting the actual power savings of targeted design instances. In summary, after performing supervised learning on several designs with the assignment ground-truths given by *Synopsys PrimeTime*, our framework has the ability to perform tool-accurate signoff power optimization on *unseen* designs instantly without degrading the performance or introducing new design rule violations (DRVs). To validate our framework, we consider *Synopsys PrimeTime* as our baseline, and demonstrate that ECO-GNN achieves comparable optimization results with up to 14X runtime improvement on the ISPD-2012 benchmarks [43] and other real-world designs, including a RISC-V based multi-core system.

The goal of this work is to provide designers a fast and accurate signoff power optimization framework with high fidelity as the industry-standard commercial tool, *Synopsys PrimeTime*. The key contributions of this chapter are summarized as follows:

1. Our first major finding is that ECO-GNN learns the behaviour of *Synopsys PrimeTime* effectively and generates comparable optimization results at inference time.

2. Our second major finding is that ECO-GNN generally shows better power saving but worse timing saving compared with *Synopsys PrimeTime*. This indicates that ECO-GNN algorithms are more effective in power optimization.

3. Unlike commercial tools or previous works (see section 8.2) that require multiple iterations to assign appropriate V_{th} types, our framework ECO-GNN only needs one-pass to determine the final V_{th} type for every design instance.

4. Rather than treating our learning-driven framework as a blackbox, we implement a GNN-based explanation method [121] to quantitatively interpret the V_{th}-assignment predictions made by our framework. Given a target node, the method identifies the influential local sub-graph that has high contribution to its V_{th}-assignment. We believe this interpretability would help designers understand the complicated characteristics of discrete sizing during signoff ECO.

5. After demonstrating the effectiveness of using GNNs to model using whole netlist information, we propose a subgraph approximation technique to speed up the training and inferencing time of the proposed GNN model using local graph structures of targeted instances without sacrificing the accuracy.

6. In this chapter, we explore both classification and regression approaches to predict ECO power optimization results, with an ultimate goal of helping designers reduce the chip design turn-around time. We demonstrate that the proposed framework can not only predict the end V_{th}-assignment of each design instance with high F1-score, but also deliver accurate estimations of the power saving from the optimization.

7. We demonstrate that the proposed subgraph approximation technique can not only be utilized to solve the regression problem of predicting actual power saving, but also be leveraged to improve the classification accuracy of the prediction of V_{th}-assignment.

8. To the best of our knowledge, this is the first work that formulates signoff power optimization problem into a graph learning problem, and validates the proposed framework using an industrial-leading commercial tool under an advanced technology node.

8.2 Related Works

The literature in V_{th}-assignment for power optimization has been researched extensively throughout the past decade. Early works mainly focus on using analytical and heuristic (i.e., non-analytical) methods to improve the optimization, however, these methods demonstrate poor generalization results across different technologies and designs. Recently, ML-based approaches emerge as promising alternatives to tackle the problem, which often demonstrate better optimization results in much lesser runtime. In the following list, we summarize previous works into these three categories:

- Non-Analytical Methods: First, we introduce the heuristic-based algorithms. This category contains methods that leverage greedy-based [122, 120, 123], simulated annealing [124, 125], or dynamic programming [126, 127] algorithms to find feasible solutions. However, there are a few major drawbacks in these algorithms. First, these algorithms often demonstrate poor convergence results, which is because they often assume the design global optimum of power optimization can be achieved by iterative finding the local optimum of cell-based power saving. Second, these approaches are highly sensitive to heuristics and are often design or technology specific. Therefore, they are hard to be extended to never-seen designs or various technologies.

- Analytical Methods: Another popular category is the analytical-based approach. Algorithms in this category often formulate the power optimization problem into a convex optimization [128, 129] or a Lagrangian optimization problem [130, 127, 131, 109] whose objective is to maximize the power reduction through discrete sizing under certain timing constraints. These methods are considered to yield better and more reliable optimization results than the non-analytical methods. However, solving an optimization problem using numerical approaches is extremely time-consuming. Given that a real-world design can easily introduce tens of thousands of variables, these approaches are limited for real-world usage.

- Machine Learning (ML): It is widely acknowledged that ML has emerged as a promising approach to solve the V_{th}-assignment problem with huge benefits in runtime saving, which is critical for productivity. The authors of [132] leverage linear regression to find feasible solutions based on path slack estimation. Another work [118] utilizes support vector machine (SVM) with lazy timing analysis to further enhance the optimization quality. However, these studies neglect that the final gate-type of each design instance highly depends on the characteristics of its neighbors. Therefore, they are not sufficient to perform the power optimization accurately without spending significant amount of time in feature engineering.

To leverage the netlist graph information in solving the power optimization problem, recently, the authors of [13, 133, 134, 12] propose graph learning-based frameworks to predict the leakage power saving of each design instance based on its local neighborhood information. These approaches demonstrate significant accuracy improvement compared with the above traditional ML-based approach while achieving similar runtime saving. Hence, it is proven that the sizing result of a targeted cell highly depends on the features of its neighbors. However, these GNN-based literature neglect the fact that the receptive field of their GNN models are only subject to the number of layers of the graph convolutional layers, which is less than 3 across all previous works. That is, if a GNN model has k convolutional layers, then the power saving prediction of an instance will only depend on the features within its local k-hop neighborhood structure and nothing beyond. In other words, the final gate-sizing prediction of a design instance will only depend on its local subgraph. Unlike previous works that rely on full-graph approaches to predict sizing solutions, in this chapter, we present a subgraph approximation technique find the solutions in much faster runtime and higher accuracy. Details of the proposed methodology will be discussed in section 8.6.

Finally, besides the specific shortcomings raised in each of the category above, there exist several common drawbacks in most of the previous works. First, the timing models they leverage are over-simplified, which does not reflect real-world scenarios. In this chapter, we think the validation from commercial signoff engine is critical to the application of the proposed models. Second, the original ISPD-2012 benchmarks [43] that most of them leverage for evaluations are problematic. We analyze the benchmarks using *Synopsys PrimeTime*, and discover that the original worst negative slack values across all the designs range from $-1ns$ to $-8ns$, where all the target frequencies are less than $1GHz$. This simple fact makes previous works unrealistic, because the power optimization is meaningful only if the optimized designs are in signoff quality. Finally, none of the previous works interpret the optimization results achieved by their methods, where they all consider their optimization engines/models as blackboxes.

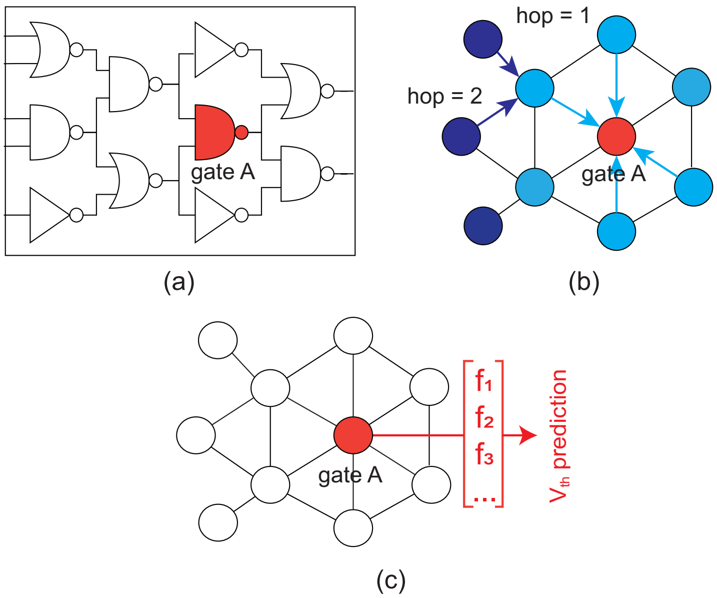


Figure 8.1: High-level view of our ECO-GNN framework, (a) input netlist, (b) graph representation learning, (c) V_{th} prediction. Note that (b) and (c) visualize the original netlist in a clique-based representation.

8.3 Designing of Experiments

Inspired from the limitations and drawbacks of the previous works, in this chapter, we consider *Synopsys PrimeTime*, a leading industrial signoff tool, as our baseline and propose ECO-GNN, a transferable graph-learning-based signoff power optimization framework that can be easily integrated with any modern PD flow. To provide fair and meaningful comparisons with prior works, we re-synthesize the ISPD-2012 benchmarks using TSMC 28nm technology node and demonstrate that our framework ECO-GNN performs commercial-quality signoff power optimization instantly on these designs. Furthermore, we leverage a GNN-based explanation method [121] to interpret the V_{th}-assignments made by our framework to ensure that our framework is reliable.

8.4 Overview of ECO-GNN Framework

Recently, GNNs have revolutionized many research areas, spanning from biology, social science, chemistry, and many others [37]. They perform effective graph representation learning, where the goal is to construct meaningful node embeddings that accurately characterize the nodes in the graph. In general, GNNs follow a message passing scheme, where a feature vector of a node can be considered as a message being iteratively transformed and passed to its neighboring nodes. At the end of the graph learning process, the initial node features are transformed into better representations that can be utilized in downstream tasks such as link prediction, node classification, and clustering [5].

Figure 8.1 presents a high-level view of our framework ECO-GNN. Since VLSI netlists can be naturally represented as hypergraphs, in this chapter, we leverage a specific variant of GNNs named GraphSAGE [37] to conduct graph representation learning directly on the netlist graphs. After getting the learned representations, we utilize a softmax-based classification model to predict the V_{th}-assignments that optimize the signoff power. Note that the entire learning is an end-to-end process. The classification loss that represents the cross-entropy between our predictions and the ground-truths from Synopsys PrimeTime is utilized to update the parameters inside GNN and the classification model through gradient descent.

The detailed learning process shown in Figure 8.1 works as follows. Given an input netlist as shown in Figure 8.1(a), to determine the V_{th}-assignment of the target cell (red-colored), we first leverage a GNN to sample and aggregate the features from its neighboring cells as shown in Figure 8.1(b). Then, we predict its V_{th}-assignment based on the aggregated representation vector as shown in Figure 8.1(c).

8.4.1 Our Objectives: Regression and Classification

The goal of this work is to construct a "general framework" that achieves *commercial-quality* signoff power optimization results *at inference time* (the testing time of the model). To achieve this goal, we explore two modeling approaches to solve different categories of problems: the regression and the classification problems. In this chapter, a regression-based model refers to the framework that predicts the "power difference" before and after ECO

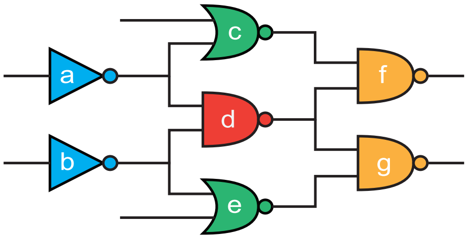


Figure 8.2: Initial feature construction of cell d, where its fan-ins $\{a, b\}$, siblings $\{c, e\}$, and fan-outs $\{f, g\}$ are taken into consideration.

optimization, where a classification-based model refers to the framework that generates the sizing results (i.e., assigning a new V_{th}-type for each design instance). These two types of problems are inherently difference and subject to various real-world applications depending on the use cases. In section 8.6, we present a "subgraph approximation" technique to solve the regression problem, and leverage a node representation learning-based technique to solve the classification problem. Finally, note that our framework does not assume any pre-defined netlist structure, so it is generalizable to every design. After learning on a few designs, it has the facility to determine the V_{th}-assignments on the unseen ones that optimize the signoff power.

8.5 Design of Experiments

In this work, we follow the experimental setting of the ISPD-2012 power optimization contest as many previous works, where all the cells in a given design are initially in the lowest V_{th} type (tightest timing constraint). As mentioned in section 8.2, we re-synthesize the ISPD benchmarks using *TSMC 28nm* technology node to ensure all the designs are in signoff performance before conducting the power optimization through V_{th}-assignments.

8.5.1 Problem Formulation

Given a netlist $G = (V, E)$, where V denotes the instances in the design, and E represents the logical connections. Assume that for each instance $v \in V$, there are n V_{th}-assignments available from the standard cell libraries. Let $x_v^j = 1$ if instance v is realized with j-th

Table 8.1: 20 initial node features used in our GNN. We obtain them using an initial PPA analysis.

type	# dim.	description
max output slew	1	max transition of output pin
max input slew	1	max transition of input pin(s)
wst output slack	1	worst slack of output pin
wst input slack	1	worst slack of input pin(s)
output cap limit	4	max driving cap of output pin per V_{th}
max leakage	4	max leakage per V_{th}
tot input cap	1	sum of input pin cap
tot fanout cap	1	output net cap + input pin cap of fan-outs
tot fanout slack	1	sum of worst slack of fan-outs
wst fanout slack	1	worst. slack of fan-outs
avg fanin cap	1	average cap of fan-ins
wst fanin slack	1	worst slack of fan-ins
tot sibling cap	1	sum of input pin cap of siblings
tot sibling slack	1	sum of worst slack of siblings

V_{th} choice in the libraries and $x_v^j = 0$ otherwise. We formally define the signoff power optimization problem as follow:

$$\text{minimize} \sum_{i=1}^{|V|} \sum_{j=1}^{n} P(v_i^j) x_{v_i}^j, \tag{8.1}$$

where $P(v_i^j)$ represents the signoff power of instance v_i when j-th choice of V_{th}-assignment is realized such that the worst negative slack (WNS) along with the total negative slack (TNS) do not degrade after the assignments, and no new DRVs are added.

8.5.2 Initial Node Features

Before leveraging GNN to conduct graph learning, we define an initial feature vector for each design instance as shown in Table 9.1. The term "initial" indicates that during the graph learning process, these original features are transformed to other representations that are more beneficial for the classification model to determine the appropriate V_{th}-assignments that optimzie signoff power.

Features in Table 9.1 are extracted from technology files, SPEF files, and timing reports.

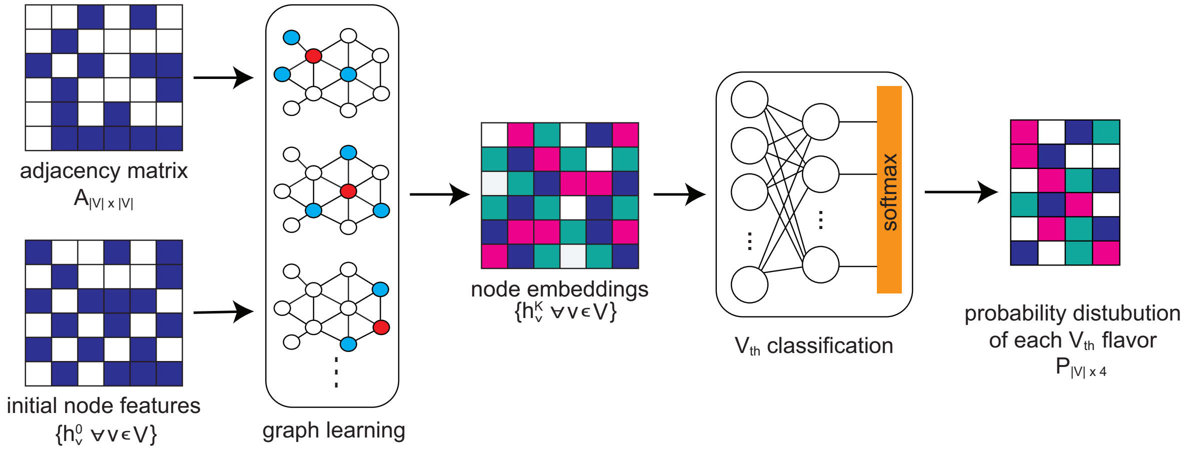


Figure 8.3: Illustration of our ECO-GNN learning process. The inputs include a netlist graph represented in an adjacency matrix A and its initial features h_v^0 defined in Table 9.1. First, we perform graph learning to generate the node embeddings that represent the netlist better than the initial features. Figure Figure 8.4 provides details of the GNN structure used. Next, with the learned node embeddings, we conduct softmax-based classification to determine the final V_{th}-assignment that optimizes the signoff power.

These 20 features are chosen based on domain knowledge and parameter sweeping experiments. Most of them are related to timing, because during the signoff power optimization, an instance's V_{th}-assignment changes only if the *WNS* and *TNS* do not degrade, and no DRV is introduced. Figure 8.2 further illustrates the feature construction process. To determine the initial features of a target instance d, we take the information of its fanins (instances $\{a, b\}$), siblings (instances $\{c, e\}$), and fanouts (instances $\{f, g\}$) into account. However, these manually engineered features are not sufficient to predict the V_{th}-assignments that optimize the design signoff power. To get better node representations, we leverage GNNs to perform the graph representation learning.

8.6 ECO-GNN Algorithm

8.6.1 Overview of the Algorithm

Figure 8.3 shows a detailed illustration of the learning process in ECO-GNN framework. Given a netlist graph $G = (V, E)$, our framework first takes the initial node features defined in Table 9.1 as inputs. Then, it leverages GraphSAGE [37], a variant of GNNs to perform

Table 8.2: Dimension of matrices used in our work (see Figure Figure 8.3). v denotes the number of gates in the circuit.

matrix	meaning	dimension
A	adjacency matrix of the netlist graph	$v \times v$
h_v^0	initial node features from PPA analysis	$v \times 20$
h_v^k	node embedding extracted by GNN	$v \times 128$
P	V_{th}-assignments from softmax function	$v \times 4$

graph learning. The goal of graph learning is to obtain the node representations that better capture the underlying characteristics of the given netlist than the intial features. After graph learning, the learned representation vector of each node $v \in V$ is projected to a logit vector P_v through a softmax-based classification model, which is a neural network. The vector P_v represents the probability distribution of node v belonging to different V_{th} flavors that are available in the standard cell libraries.

Table Table 8.2 shows the size of matrices used in our framework. The adjacency matrix A represents the logical connections in the netlist, and the initial node features $\{h_v^0 \, \forall v \in V\}$ are the cell attributes shown in Table 9.1. Note that the whole learning process, from graph learning to V_{th} classification, is end-to-end differentiable. Therefore, the parameters in the GNN and classification modules can be updated simultaneously using gradient descent.

8.6.2 GNN: Feature Aggregator

The goal of graph learning is to construct accurate node embeddings through effective feature aggregation. GNN functions as a feature aggregator that transforms the initial features h_v^0 for each node $v \in V$ into better representations h_v^K by *sampling and aggregating* the features within v's K-hop neighborhood. This aggregation process is performed iteratively, where for each hop $k \in \{1, ..., K\}$, a dedicated neural network (NN) $\mathbf{W}_k$ is developed to perform the transformation. These K dedicated NNs together form the GNN module in our framework as shown in Figure 8.4. Since the number of neighbors of a node scales exponentially as the hop-count increases, we fix the sampling size s_k at each hop k to improve the computational efficiency and to prevent overfitting.

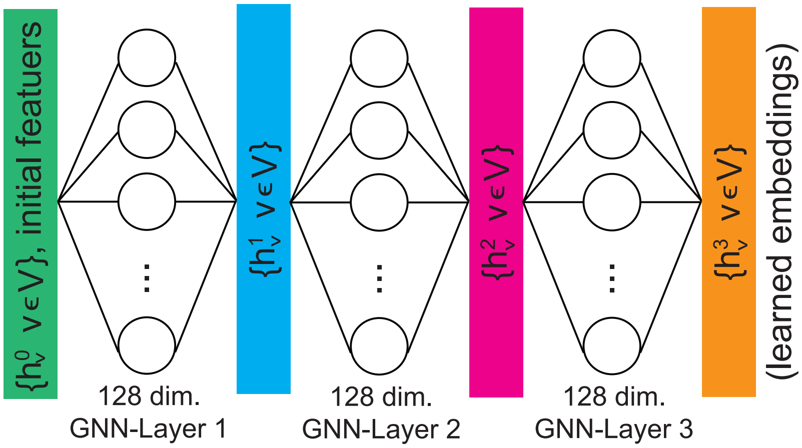


Figure 8.4: Our GNN architecture that maps the initial node features (20) into learned embedding features (128).

Following the graph learning approach presented in [37], in this work, for each node $v \in V$, we obtain its representation vector h_v^k at level[1] k by aggregating its representation h_v^{k-1} at the previous level with the features of its neighbors $N_k(v)$ sampled at k-hop as

$$\begin{aligned} h_{N_k(v)}^{k-1} &= \text{maxpool}\left(\{\mathbf{W}_k^{agg} h_u^{k-1}, \forall u \in N_k(v)\}\right), \\ h_v^k &= sigmoid\left(\mathbf{W}_k^{proj} \cdot \text{concat}[h_v^{k-1}, h_{N_v(v)}^{k-1}]\right), \end{aligned} \tag{8.2}$$

where W_k^{agg} and W_k^{proj} denote the aggregation and projection matrices respectively, which together form the weights of the NN dedicated in *sampling and aggregating* features at the k-hop neighborhood. In the implementation, we set $k \in \{1, 2, 3\}$, and each NN (W_1, W_2, W_3) in the GNN module has an output dimension of 128. Note that the numbers 128 and 3 are chosen empirically based on parameter sweeping experiments.[2]

In summary, the initial feature vector h_v^0 for each node $v \in V$ is transformed to $h_v^{K=3}$ in R^{128}. The GNN model utilized in our framework can be considered as a "node filter", because it iterates through every design instance to find better node representations that can

[1] Level is corresponding to the hop-count. When aggregating the features of a node at level k, the information within its k-hop neighborhood is considered.

[2] We varied them while monitoring the overall power saving vs. training time tradeoff. Due to the page limit, we omit the related experimental results. But, a general trend shows that the higher the values are, the more the power saving is at the cost of training time. But, the power saving saturates after some point.

be utilized in the latter classification task of determining the V_{th}-assignments that optimize the design signoff power.

8.6.3 Loss Function

After leveraging GNN to perform graph representation learning, we take the learned node embeddings $\{h_v^K \in R^{128}, \forall v \in V\}$ as the inputs of our softmax-based classification model, which is a neural network, in order to determine the appropriate V_{th}-assignment for each design instance. As shown in Figure 8.3, the end of the classification model connects to a softmax function that outputs P, which is a $|V| \times n$ matrix denoting the probability of each node v belonging to n different V_{th} flavors, where $\forall v \in V$, $\sum_{c=1}^{n} P_{vc} = 1$. Note that n is limited to the discrete V_{th} values specified by the technology. The technology we utilize in this work is ***TSMC 28nm*** which has $n = 4$. A novelty of this work is that we map the discrete V_{th}-sizing problem into a multi-class classification problem, where the classification loss function is defined as:

$$\mathcal{L} = -\sum_{i=1}^{|V|}\sum_{c=1}^{n} Y_{ic} log(P_{ic}), \tag{8.3}$$

where $Y \in R^{|V| \times n}$ denotes the V_{th}-assignments made by the *Synopsys PrimeTime* ECO engine, which are taken as ground-truths. Essentially, our loss function (Equation 8.3) represents the cross-entropy between Y and P distributions. By minimizing Equation 8.3, we can update the parameters in the entire ECO-GNN framework.

8.6.4 Training Methodology

The training process of ECO-GNN is supervised, where we use the V_{th}-assignments obtained from *Synopsys PrimeTime* ECO engine as ground-truths, and minimize a supervised loss function to update the GNN parameters. The main reason we discard traditional machine learning techniques as the ones used in previous works [118, 132] is that these

Algorithm 12 ECO-GNN training methodology.
We use default values of $K = 3, \alpha = 0.001, s_1 = 25, s_2 = 20, s_3 = 15, \beta_1 = 0.9, \beta_2 = 0.999$.

```
Input: (1) $G(V, E)$: netlist graph, (2) $A_{|V|\times|V|}$: adjacency matrix, (3) $Y$: tool optimization results,
    (4) $n$: number of available $V_{th}$ flavors, (5) $\{h_v^0, \forall v \in V\}$: initial features. (6) $K$: depth
    of aggregation level, (7) $\{s_k, \forall k \in \{1, ..., K\}\}$: sampling size at k-hop neighborhood, (8)
    $\{\mathbf{W}_k, \forall k \in \{1, ..., K\}\}$: parameters of NN at hop $k$, (9) $\alpha$: learning rate, (10) $\{\beta_1, \beta_2\}$: Adam
    parameters.
Output: $P_{|V|}$: $V_{th}$-assignment prediction of each instance.
 1: while $\{\mathbf{W}_k\}$ do not converge do
 2:     $h_v^0 \leftarrow \frac{h_v^0}{\|h_v^0\|_2}, \forall v \in V$                    ▷ initial features from Table 9.1
 3:     for $k \leftarrow 1$ to $K$ do
 4:         for $v \in V$ do                                   ▷ sample and aggregate by Equation 9.1
 5:             $N_k(v) \leftarrow$ Sample $s_k$ neighbors at $k$-hop
 6:             $h_{N_k(v)}^k = \text{maxpool}\left(\{\mathbf{W}_k^{agg} h_u^{k-1}, \forall u \in N_k(v)\}\right)$
 7:             $h_v^k = sigmoid\left(\mathbf{W}_k^{proj} \cdot \text{concat}[h_v^{k-1}, h_{N_v(v)}^k]\right)$
 8:         $h_v^k \leftarrow \frac{h_v^k}{\|h_v^k\|_2}, \forall v \in V$                ▷ reduce gradient oscillation
 9:     for $v \in V'$ do                                         ▷ minimize Equation 8.3
10:         $p_v \leftarrow softmax(\mathbf{W}_k^{NN} \cdot f_v^K)$
11:         $g_v \leftarrow \nabla_\theta[\sum_{c=1}^{n} Y_{ic} log(p_{vc})]$
12:         $\{\mathbf{W}_k\} \leftarrow Adam(\alpha, \{\mathbf{W}_k\}, g_v, \beta_1, \beta_2)$
```

techniques fail to consider the neighborhood information of an instance while determining the V_{th}-assignment, where the assignment certainly depends on the neighborhood structure such as the impacts of the propagated arrivals and transition effects.

Algorithm 12 summarizes the training process. Lines 3–10 illustrate the *sampling and aggregating* process in graph learning, where for each node $v \in V$, we aggregate its neighboring features at each hop $k \in K$ through Equation 9.1. Note that before performing each aggregation, we normalize the node representations at previous level as shown in Line 2 and Line 9. This normalization accelerates the overall training process by reducing the oscillation of gradient descent. Based on the learned representation vectors, in Lines 11–15 we calculate the cross-entropy loss (Equation 8.3) from the softmax-based classification model, and leverage a gradient descent optimizer named *Adam* [38] to update the parameters in the framework by minimizing the loss function. The overall training process takes about 12 hours on the 9 training designs shown in Table 9.2 with a machine that has a 2.40 GHz CPU and a NVIDIA RTX 2070 graphic cards with 16 GB memory.

8.6.5 Complexity Analysis

The time complexity of ECO-GNN is linear with respect to the netlist size. Since the sampling size (s_k) at each aggregation level is constrained, GNN modules spend constant time in visiting every design instance and collecting features from its neighbors. Due to the large sparsity of the netlist adjacency matrix, we realize the adjacency matrix A shown in Table 8.2 in the compressed sparse row (CSR) format [135]. Therefore, the space complexity of ECO-GNN is pseudo-linear rather than quadratic with respect to the netlist size because it is mainly constrained by the number of nets of the underlying design. As shown in Algorithm 12, our framework conducts instance-based learning, where each instance (cell) in the design can be considered as a data point. Given a netlist $G = (V, E)$, Therefore, after learning on the 9 training designs presented in Table 9.2 which in total contain millions of data points, our framework achieves remarkable optimization results.

8.6.6 Handling Unseen Designs

A highlight of this work is that a trained ECO-GNN framework has the ability to perform commercial-quality signoff power optimization on *unseen* designs *at inference time*. This capability is independent of the netlist structure or the netlist size, because to determine the V_{th}-assignments that optimize the signoff power in an unseen design, we only need to take the initial features and the adjacency matrix as inputs, and ECO-GNN will determine the appropriate V_{th}-assignments through constant time inferencing. Unlike *PrimeTime* and previous works that require multiple iterations to determine the final V_{th}-assignments, our framework is a one-pass tool that generates tool-accurate results instantly.

8.6.7 A Regression Perspective: Subgraph Approximation for Fast Power Prediction

Up to now, we have presented a complete graph learning-based framework that can predict the final V_{th}-assignment of each design instance without actually running signoff power ECO which is highly time-consuming. Specifically, we cast the ECO signoff power optimization prediction as a classification problem, where we present a GNN-based model that classifies each design instance to a specific V_{th}-type. Nonetheless, in many real-world scenarios, designers are seeking for a direct estimate of the actual power saving value (i.e., regression) before running any optimization or performing any netlist update using ECO change-lists. In addition, on certain occasions, designers would only want to conduct the ECO power optimization on partial netlist rather than the whole netlist. For instance, in modern industrial design flows, it is common to solely perform the signoff power optimization on the design instances whose slack values are larger than a pre-defined threshold, which is because the ECO changes made on these instances (with large positive slack values) will introduce the least timing impact to the overall placed and routed design. Therefore, in this section, we will present a new methodology that meets these needs of designers.

Although we can apply Algorithm 12 to overcome the above problem of partially optimizing the netlist by performing a full-graph (i.e., full-netlist) inference on every design instance, this computation will introduce excessively unnecessary computational resources and runtime given that the optimization is targeted on a few instances. To overcome this issue, we propose a new methodology named “subgraph appoximation”, where instead of using the information of the entire netlist to predict the final gate types of a few instances, we leverage GNN to encode the features from their local subgraphs.

Given a target instance $v \in G$, the subgraph sG_v of this instance v refers to its local three-hop neighborhood graph structure of the underlying netlist. For each selected instances V that meet the slack threshold for signoff power optimization, we leverage GNNs to perform subgraph encoding, where the goal is to construct a meaningful graph-level feature representations that capture the characteristics of the targeted instances. Note that the philosophy behind this subgraph-based approach is fundamentally different from the previous approach. The current approach focuses on learning “graph-level” vectors that characterize the information related to optimizing single design instance, where the previous approach (i.e., Algorithm 12) dedicates on learning the “node-level” embeddings that

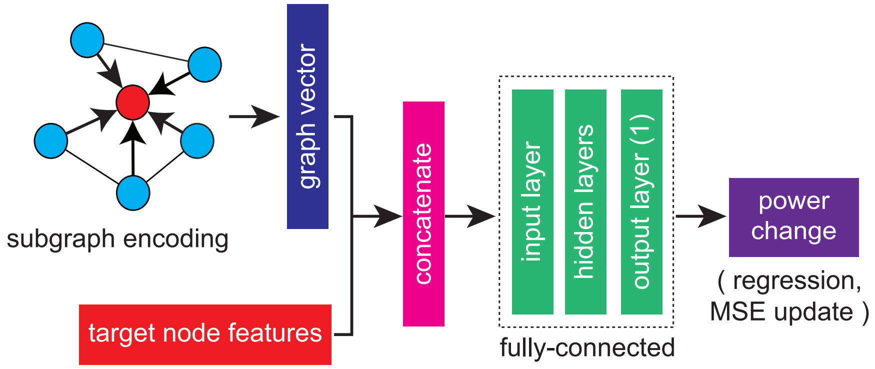


Figure 8.5: Overview of subgraph approximation for power prediction of target instance (red-colored). Target nodes refer to the design instances that are selected for partial netlist update.

capture the interaction among all instances in the netlist. Based on the initial node features defined in Table 9.1, we first leverage Equation 9.1 to transform the initial features to high-dimensional representations. Then, for each node v in a subgraph sG, we obtain the graph-level vector s through

$$s = \text{concat}\left[mean\_pool\left(\left\{h_v^{k=K}\right\}\right), feat(v)\right], \tag{8.4}$$

where $feat(v)$ denotes the underlying features of the target instance v that is selected for the power optimization, which includes the current power consumption (internal and driving net switching power), cell capacitance (input pin cap), driving strength, and the worst transition values of input and output pins. The subgraph vector s, which characterizes the "local" information of the target instance v that is related to its power optimization, will be taken as the input of fully-connected layers to directly predict the change of power consumption.

Finally, Figure 8.5 demonstrates an overview of the subgraph-based power prediction flow, where the goal is to predict the power difference before and after performing power optimization on the target instance colored in red. The detail steps are as follow. First, we leverage GNN to preform node representation learning on the local 3-hop neighborhood subgraph of the target instance, which includes the fanout/fanin cells up to three levels

Algorithm 13 Assisting V_{th}-assignment with subgraph approximation.

Input: (1) $G(V, E)$: netlist graph, (2) $A_{|V|\times|V|}$: adjacency matrix, (3) t: target instance, (4) $\mathbf{W}_{ECO-GNN}$: weights of ECO-GNN, (5) $\mathbf{W_{nn}}$: weights of feedfoward neural networks, (6) Y: tool optimization results

Output: $\{p_t\}$: V_{th}-assignment of instance t.

```
while $\{\mathbf{W}\}$ do not converge do
    $s_t \leftarrow$ 3-hop local subgraph of instance $t$
    $g_t \leftarrow$ subgraph encoding of $s_t$        ▷ graph-level vector as in Figure 8.5
    $h_t \leftarrow$ ECO-GNN$(G, A; \mathbf{W}_{ECO-GNN})$        ▷ node embeddings of cell $t$ from full-graph learning
    $c_t \leftarrow$ concat$[g_t, h_t]$        ▷ concatenate graph-level vector with node embeddings
    $p_t \leftarrow softmax(\mathbf{W}_{nn} \cdot c_t)$
    $g_t \leftarrow \nabla_\theta[\sum_{c=1}^{4} Y_{ic}log(p_{tc})]$
    $\{\mathbf{W}\} \leftarrow Adam(\alpha, \{\mathbf{W}\}, g_t)$
```

and the sibling cells up to two levels. This node representation learning will transforms the initial features of each cell in the subgraph into high-dimensional representations (128 dimensions). Then, a global mean pooling is performed across each cell in the subgraph to obtain a graph-level vector, which is expected to represent the ECO-related characteristics of the target instance. Finally, the graph-level vector, along with the initial feature vector of the targeted instance, is fed to a downstream feed-forward neural network to predict the actual power saving and mean-squared-error (MSE) is utilized as the loss function to train the entire framework.

8.6.8 Assisting V_{th}-Assignment with Subgraph Approximation

The V_{th}-assignment problem is a classification task by nature as the available gate sizes are discrete, which we solve by leveraging GNNs to encode full-netlist information as shown in Algorithm 12. However, as aforementioned, the purpose of the proposed subgraph approximation technique is to perform regression-based power prediction by merely using partial-netlist information. Therefore, to assist the V_{th}-assignment task with the subgraph approximation technique, a learning methodology needs to be developed to bridge the gap between the classification and the regression tasks.

Algorithm 13 summarizes how the subgraph approximation technique can be leveraged to assist the traditional V_{th}-assignment task, where the key idea is to leverage the encoded graph-level vector from subgraph approximation as additional information to help predict the final V_{th} type of the target instance. The detail steps are as follow. First, we leverage the subgraph approximation technique to encode the local 3-hop neighborhood subgraph of the target instance t to obtain a graph-level vector g_t (Lines 2–3). Then, we leverage the aforementioned ECO-GNN framework to perform node representation learning and obtain the learned embeddings h_t of the target instance (Line 4). Finally, we concatenate the graph-level vector (describing the local neighborhood structure) and the learned embeddings as input to the downstream feedforward classification network to predict the final V_{th} assignment. Note that the entire algorithm is end-to-end differentiable.

8.7 Explaining Prediction Results

Understanding the reasons behind the predictions of ML models can give users better trust in the models. In this work, we explore explanation techniques that provide insights of the V_{th}-assignment predictions made by the proposed framework ECO-GNN. Given a targeted instance for explanation, we will explore its local subgraph to determine what are the important factors in terms of neighboring nodes and connected nets that drive the prediction of the proposed graph learning-based framework.

8.7.1 Inner Workings of GNN Predictions

Unlike previous works who consider their optimization engines as blackboxes, in this chapter, we implement a GNN-based explanation method [121] to interpret the V_{th}-assignment predictions made by our framework ECO-GNN. Given a set of target instances $\{v\} \in V$ in a netlist graph $G = (V, E)$, the goal is to find an influential sub-graph $G_S = (V_S, E_S)$ that has high contribution to the decision of $\{v\}$'s V_{th}-assignments. The objective of finding such sub-graph G_S can be quantitatively formulated as maximizing the mutual information

(MI) between the original graph G and the sub-graph G_S as:

$$\max_{G_S} MI(G, G_S) = H(Y) - H(Y|G = G_S), \tag{8.5}$$

where $H(\cdot)$ denotes the entropy of the given distribution and Y represents the V_{th} prediction distribution of the target instances. Since $H(Y)$, the entropy of the prediction distribution based on the original graph, is a constant, maximizing Equation 8.5 is equivalent to minimizing the conditional entropy $H(Y|G = G_S)$ which can be formulated as:

$$H(Y|G = G_S) = -\mathbb{E}_{Y|G=G_S}\left[log\left(P_\theta(Y|G = G_S)\right)\right], \tag{8.6}$$

where θ denotes the parameters of the trained ECO-GNN framework. Note that due to the fact that the number of neighbors of the target nodes increases exponentially as the hop-count increases, in the implementation, we constrain G_S to search within the one-hop neighbors of the target instances $\{v\}$. In the context of the actual netlist, G_S represents the cells that are either the fanins, fanouts, or siblings of $\{v\}$ as well as the message passing flows (edge connectivities) that demonstrate how important features are aggregated. We believe this interpretability would give designers precious insights on what the framework has learned and whether the V_{th}-assignments are reliable or not.

8.8 Experimental Results

In this section, we demonstrate the achievements of our ECO-GNN framework, which is implemented in *Python3* with *Tensorflow 1.0* library. We leverage 7 designs from the ISPD-2012 benchmark [43] and 7 other industrial designs to conduct the experiments. All 14 designs are synthesized under *TSMC 28nm* technology node by *Synopsys Design Compiler 2015*, and placed and routed using *Cadence Innovus v18.1*. To validate the signoff power optimization results of ECO-GNN, we use *Synopsys PrimeTime 2018* to perform timing and power analysis, and consider the *PrimeTime* ECO engine as the baseline across all

Table 8.3: Our benchmarks and their attributes in *TSMC 28nm*. MPL denotes the maximum path length of timing paths, SR denotes the spectral radius of the adjacency matrix, and RCC denotes the Rich Club Coefficient $(10^{-}4)$.

Design Name	# Nets	# FFs	# Cells	MPL	SR	RCC	Usage
RocketCore	93,812	16,784	90,859	68	381	7	training
AES-128	90,905	10,688	113,168	17	68	12	
NOVA	138,171	29,122	136,537	32	185	3	
ECG	85,058	14,018	84,127	29	76	8	
LDPC	42,018	2,048	39,377	23	229	14	
DMA	10,898	2,062	10,215	15	29	52	
PCI_BRIDGE	1,381	310	1,221	24	33	307	
DES_PERF	48,523	8,802	48,289	17	28	29	
B19	34,399	3,420	33,784	35	29	22	
TATE	185,379	31,409	184,601	31	26	5	testing
JPEG	231,934	37,642	219,064	26	173	4	
VGA_LCD	56,279	17,054	56,194	24	25	19	
LEON3MP	341,263	108,724	341,000	48	28	3	
NETCARD	317,974	87,317	316,137	37	31	4	

experiments.

8.8.1 Benchmarks Details and Timing Corners

As mentioned in section 8.2, due to the unrealistic nature of the ISPD-2012 benchmark that the worst negative slacks in the original designs range from $-1ns$ to $-8ns$, we re-implement all seven ISPD designs using *TSMC 28nm* technology node and commercial PD tools. Aside from the ISPD benchmarks, we introduce 7 other renowned industrial designs, including JPEG, TATE, LDPC, AES-128, NOVA, ECG from *OpenCores.org*, and RocketCore [42] which is a RISC-V-based multi-core system. To substantiate the generality of our framework, we utilize 9 designs in the training process, and perform the validations on the 5 *unseen* ones. The characteristics of these 14 designs are shown in Table 9.2. In the table, we also demonstrate the graph-related statistics, which include the maximum path length of timing paths, the spectral radius (i.e., maximum eigenvalue of adjacency matrix), and the Rich Club Coefficient (an indicator of connectivity).

Following the experimental settings of the ISPD-2012 contest where all the designs are synthesized with one timing corner and one V_{th} flavor which has the tightest timing

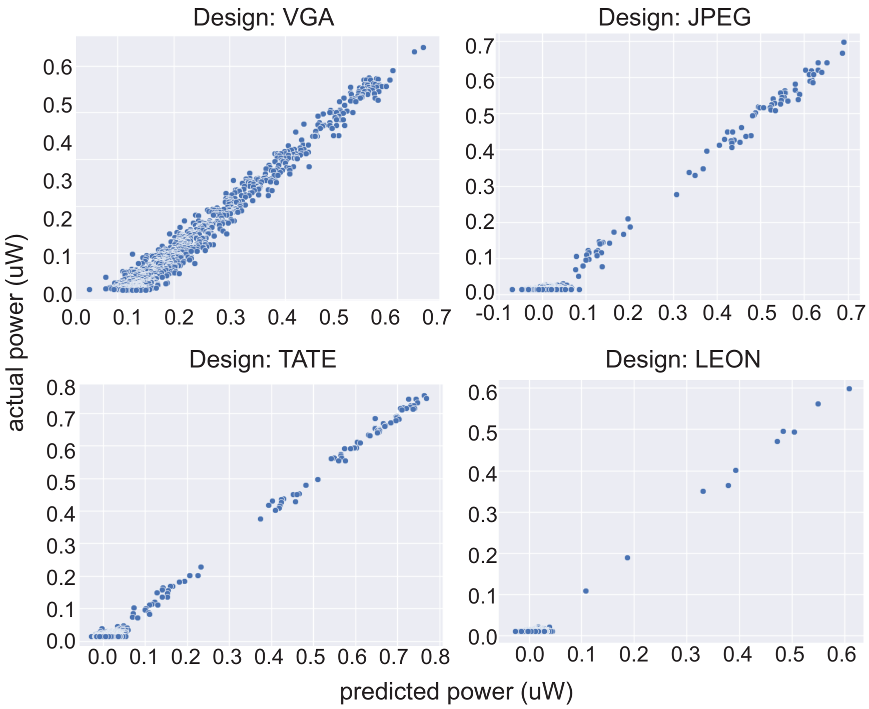


Figure 8.6: Prediction results of subgraph approximation. We validated the subgraph model on 4 unseen designs. Each dot in the plots represents a design instances whose initial slack is above $200ps$ before ECO power optimization.

constraint, in this work, we synthesize all the designs using typical corner and ultra-low V_{th} flavor (tightest timing constraint) in *TSMC 28nm* for fair comparisons. In the *PrimeTime* ECO for signoff power optimization, each design instance is enabled to be swapped into one of the three other V_{th} flavors, which are low, high, and ultra-high types, or remain as the ultra-low type (4 choices in total). Therefore, the solution space of our V_{th}-assignment problem is $4^{|V|}$, which is almost impossible for designers to perform design exploration in an exhaustive manner.

8.8.2 Subgraph Approximation Results

As aforementioned, in this chapter, we not only develop a classification-based model to predict the final V_{th}-type of each design instance using entire netlist information, we also present the subgraph approximation technique to estimate the power recovery of individual instances during signoff ECO. Table 8.4 demonstrates the prediction results on 4 unseen

Table 8.4: Subgraph approximation prediction results on unseen benchmarks. CC denotes the Pearson correlation coefficient and is calculated against the *ICC2* optimization results.

Unseen Design		VGA	JPEG	TATE	LEON
NRMSE %		2.2	2.8	1.7	1.4
CC		0.96	0.97	0.97	0.98
total power	before	212.7	376.8	345.0	576.6
(mW)	after	197.9	342.3	328.7	552.4
WNS	before	-3.4	-13.1	-2.4	-16.3
(ps)	after	-3.1	-12.8	-2.3	-16.2
TNS	before	-14.1	-228.7	-3.2	-246.0
(ps)	after	-12.4	-202.5	-3.0	-239.3

Table 8.5: Confusion matrix comparison of V_{th}-assignment with and without subgraph approximation on the VGA benchmark. Each count represents an instance whose slack value is greater than $200ps$ before the PrimeTime ECO optimization.

		predictions (accuracy: 0.94)				
	w.o. subgraph	ultra-low	low	high	ultra-high	Total
	ultra-low	429	17	53	82	581
ground	low	22	1557	144	202	1925
-truths	high	14	18	5026	166	5224
	ultra-high	45	111	189	9110	9455
	Total	510	1703	5412	9560	17185

designs. In this chapter, we specifically focus on two metrics to evaluate the model: normalized root-mean-squared error (NRMSE) and the correlation coefficient (CC). Note that NRMSE is calculated by normalizing the RMSE which inherently comes with a "unit" (e.g., mW) by the difference between the maximum and minimum ground truth values (i.e., $NRMSE = \frac{RMSE}{power_{max} - power_{min}}$). NRMSE is a popular comparison metric that removes the effect of unit scale. As shown in the figure, we observe that the proposed model consistently delivers highly accurate prediction results across the unseen benchmarks. Finally, Figure 8.5 shows the scatter distribution of the prediction results, where each dot represents an actual subgraph whose worst slack value before the optimization is $200ps$.

8.8.3 Prediction results of V_{th}-Assignment with Subgraph Approximation

In this experiment, we demonstrate the effectiveness of using the proposed subgraph ap-

		predictions (accuracy: 0.96)				
	w. subgraph	ultra-low	low	high	ultra-high	Total
	ultra-low	518	26	23	14	581
ground	low	12	1645	188	80	1925
-truths	high	21	93	4973	137	5224
	ultra-high	37	19	124	9275	9455
	Total	588	1783	5308	9506	17185

proximation technique to improve the prediction task of V_{th}-assignment. Table 8.5 demonstrates the detailed prediction results in the format of confusion matrices. The upper table shows the results without using the subgraph approximation technique, and the lower table demonstrates the results achieved with subgraph approximation using Algorithm 13. In the table, we observe that the subgraph approximation technique can indeed boost the prediction accuracy, which is expected as more information (e.g., the graph-level vector) is curated for the model to make better predictions.

8.8.4 Discussion of Subgraph Approximation

One of the highlights of this chapter is the proposed concept of subgraph approximation, which enables fast and accurate prediction of power optimization. The rationale behind the proposed subgraph approximation technique is two-fold. First, given that power and timing are often inter-related with each other and the power recovery in ECO often comes under the sacrifice of timing degradation, the final V_{th}-type of a target instance v will not only depends on its direct one-hop neighbors, but also other neighbors that may or may not locate on the same timing paths. Therefore, we leverage GNNs to encode such neighboring information for the final prediction of its power recovery. Second, the reason we do not select a huge number of hops to perform the subgraph encoding is because the QoR impact of a single gate sizing move on a design instance to the overall netlist diminishes quickly as the hop count increases.

8.8.5 Optimization Results on Unseen Designs

In this experiment, we compare the signoff power optimization results achieved by our framework ECO-GNN with the commercial tool *Synopsys PrimeTime*. To substantiate the generality of ECO-GNN, we only use 9 designs for training, and perform validations on the 5 unseen ones as shown in Table 9.2. Note that to perform meaningful and reasonable signoff power optimization, each design is originally implemented in signoff frequency, where the WNS is close to 0. The optimization constraints are that the WNS and TNS do not degrade and no violation is introduced after the optimization.

Table 8.6 demonstrates the optimization results. Compared with *PrimeTime*, ECO-GNN achieves up to 14X runtime improvement with similar optimization quality. Unlike previous works that do not utilize commercial signoff tools for validations, we demonstrate that our framework performs tool-accurate signoff power optimization without degrading the original signoff performance of each unseen design. Note that each design in Table 8.6 has different target frequencies, which proves that the optimization achieved is not confined by design characteristics. The inference time of ECO-GNN is measured on a machine with 2.40 GHz CPU and a NVIDIA RTX 2070 graphics card with 16GB memory, where *Synopsys PrimeTime* is ran on a machine with 2.50 GHz CPU and 8 cores enabled.

Table 8.6 also reports the micro F1-score as the evaluation metric of the classification task, owing to the fact that our framework ECO-GNN is performing supervised learning that we take the V_{th}-assignments from *Synopsys PrimeTime* as ground-truths in the training process. Note that micro F1-score represents the accuracy of multi-class classification. In the table, we observe that ECO-GNN performs the the V_{th}-assignments in high fidelity as *PrimeTime*.

Finally, due to the fact that V_{th}-assignments directly optimize the design leakage power, Figure 8.8 further shows the instance-based leakage power consumption maps of the unseen designs, which are corresponding to the optimization results presented in Table 8.6. In the figure, we compare the leakage power consumption of each instance in the original designs

Table 8.6: V_{th} re-assignment impact on power, timing, and runtime between ECO-GNN and *Synopsys PrimeTime*. Selected designs are *unseen* during training. Note that both leakage and total power reduce from V_{th} re-assignment, because our initial designs before ECO optimization are using ultra-low V_{th} only as suggested in [43]. Timing also improves because of the gate capacitance reduction from higher V_{th}.

Design	Target Frequency	Optimization Engine	Leakage Power (mW)	Total Power (mW)	WNS (ps)	TNS (ps)	Runtime (sec)	F1-Score $(micro)$
TATE	1.2GHz	Before Opt.	38.3	345.0	-2.4	-3.2	-	0.90
		PrimeTime	1.84	282.7	-0.5	-0.6	141	
		ECO-GNN	1.72	280.6	-0.9	-1.8	16 (9X)	
JPEG	1.1GHz	Before Opt.	57.5	376.8	-13.1	-228.7	-	0.85
		PrimeTime	3.4	294.6	-5.7	-69.6	120	
		ECO-GNN	3.8	296.9	-11.4	-182.4	15 (8X)	
VGA_LCD	1.8GHz	Before Opt.	18.0	212.7	-3.4	-14.1	-	0.89
		PrimeTime	3.7	184.6	-2.6	-4.4	69	
		ECO-GNN	3.5	183.3	-3.2	-11.8	5 (14X)	
LEON3MP	700MHz	Before Opt.	101.4	576.6	-16.3	-246.0	-	0.88
		PrimeTime	15.1	459.8	-8.4	-76.7	341	
		ECO-GNN	12.2	454.9	-12.8	-209.3	28 (12X)	
NETCARD	1GHz	Before Opt.	78.1	651.5	-2.4	-4.2	-	0.86
		PrimeTime	9.9	544.3	-0.8	-1.1	302	
		ECO-GNN	6.9	537.6	-1.2	-2.7	26 (12X)	

Table 8.7: Sweeping experiments on maximum number of aggregation level (K) of GNN. The entry represents the F1 score of the classification results.

Designs (F1-score)	K=1	K=2	K=3	K=4	K=5
TATE	0.39	0.78	0.90	0.82	0.73
JPEG	0.33	0.74	0.85	0.81	0.70
VGA_LCD	0.42	0.81	0.89	0.85	0.74
LEON3MP	0.36	0.84	0.88	0.79	0.66
NETCARD	0.41	0.69	0.86	0.83	0.72

with the ones after using ECO-GNN to perform signoff power optimization. Across all designs, we observe that ECO-GNN effectively reduces the overall leakage power without introducing extra hotspots.

Sweeping Experiments on GNN Aggregation Level

In the realm of graph learning using GNNs, choosing a right number of the maximum aggregation level is critical to the success of representation learning. Nonetheless, it is known that GNNs tend to suffer from the over-smoothing problem [136], which is an issue that the representations among different nodes become indistinguishable and thus the prediction accuracy becomes worse. Therefore, for the majority of GNN applications, the number of aggregation level is empirically set to a number between 2 and 4 (inclusive) [137]. In this chapter, we validate the hypothesis by providing sweeping experiments over the GNN aggregation level as shown in Table 8.7, where we clearly observe that in the classification task of predicting final V_{th}-type for each design instance, the best accuracy occurs when the number of aggregation level is set to 3.

8.8.6 Discussion of Optimization Results

Power Perspective. As shown in Table 8.6, the optimizations through V_{th}-assignments achieved by our framework and *PrimeTime* improve both leakage power and total signoff power. This is because we follow the experimental settings from the ISPD-2012 contest [43] as many previous works [122, 120, 123, 124, 131, 109, 118]. The setting

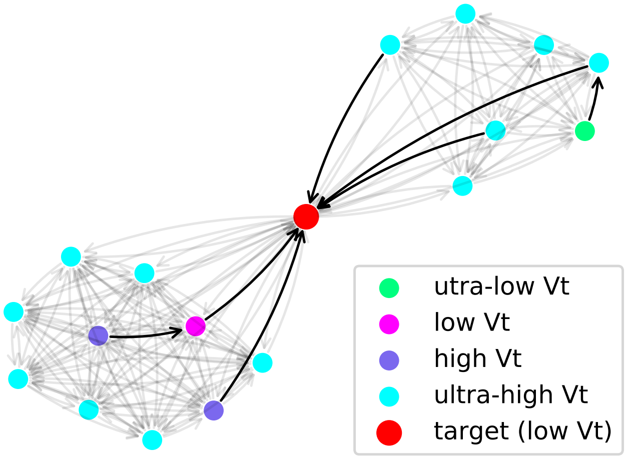


Figure 8.7: Graph learning explanation on b19 benchmark. The majority of the neighbors are ultra-high V_{th}, but cells with lower V_{th} types have higher importance to the target node. As a result, low V_{th} is assigned to the target node.

suggests all the cells to be in ultra-low V_{th} (tightest timing constraint) before the optimization. Therefore, for a design instance, a swap from ultra-low V_{th} to other V_{th} types in *TSMC 28nm* not only improves its static power (leakage) but also the dynamic power as the capacitance load is reduced.

Timing Perspective. As shown in the table, we observe that the WNS and TNS get improved as well. This comes from the fact that although *PrimeTime* will not upsize the V_{th} type of the cells that are on critical (negative slack) paths, the driving load of such cells may still be reduced if some of its fanout cells that are not on critical paths are swapped to higher V_{th} types, which in the end improves the overall timing as a by-product.

8.8.7 GNN Explanation

Instead of viewing our framework ECO-GNN as a blackbox, we validate our optimization results by explaining the V_{th}-assignments made by our framework. Figure 8.7 demonstrates the explanation results on the b19 design, where we plot the graph learning computational graph centered on the target node colored in red along with its neighbors using force-directed placement drawing [138]. Note that although we present single-instance ex-

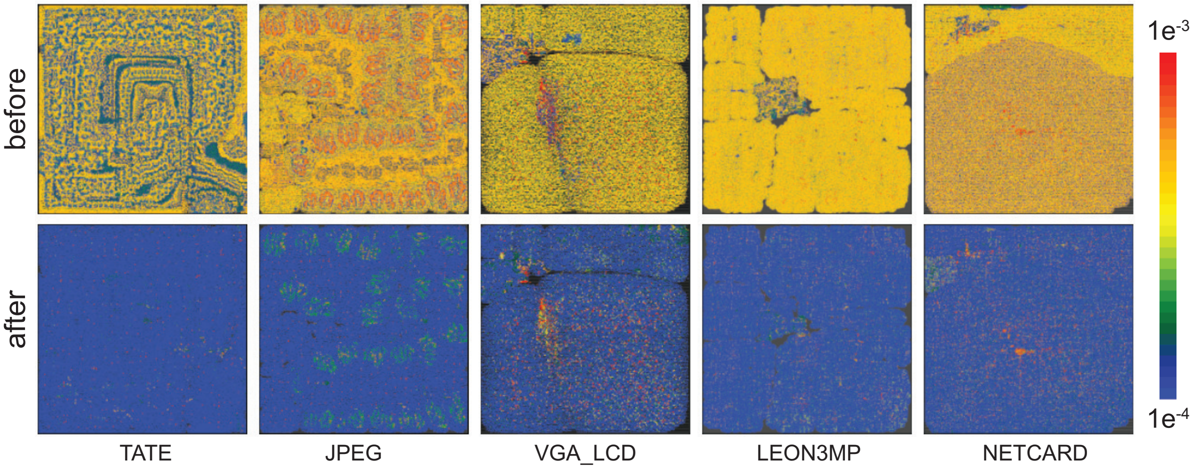


Figure 8.8: Leakage power consumption of each design instance before and after using ECO-GNN to perform optimizations. The designs are *unseen* during training, and the unit is mW.

planation in this experiment for clarity, the proposed explanation method can be leveraged to perform the explanation of multi-instance as well.

To explain the V_{th}-assignment on the target node (red), we identify the important message passing flows within the local sub-graph. As mentioned in subsection 8.7.1, we constrain the explanation method to search within the one-hop neighborhood. Therefore, every neighboring node in Figure 8.7 is either the fanin, fanout, or sibling of the target node. However, as shown in the figure, the influential features may not be passed directly from the neighbors to the target even though they are one-hop neighbors. This is because the message passing scheme in graph learning is bi-directional. For better illustration, in Figure 8.7, we plot two directed edges for each bi-directional edge in the graph learning computational graph to show how the influential features are being passed.

Figure 8.7 shows that the V_{th}-assignment made by ECO-GNN on the target node is reliable, because we observe that the final V_{th} type of the target node is more influenced by its minority neighbors who are in lower V_{th} types rather than the majority neighbors that are in the ultra-high V_{th} type. This aligns well with common design knowledge. Since cells in lower V_{th} types have larger capacitance, tighter constraints will be imposed on their drivers compared with cells in high-level V_{th} types. Therefore, we conclude that the

V_{th}-assignment made by ECO-GNN on the target cell is reliable.

8.8.8 Why Does ECO-GNN Work?

In the experiments, we demonstrate that ECO-GNN achieves commercial quality signoff power optimization results with negligible runtime compared with *Synopsys PrimeTime*. The achievements of our framework can be accounted by two reasons. First, the initial modeling features (Table 9.1) accurately capture the underlying characteristics of each design instance that are related to the signoff power optimization. Specifically, the timing related features provide solid information for our framework to select appropriate V_{th}-assignments that optimize signoff power with the consideration of timing budget. Second, GNNs are highly powerful for solving the optimization problems on graphs. The final V_{th}-assignment of an instance highly depends on the information of its neighborhood structure. Therefore, unlike previous works [118, 132] who use traditional machine learning techniques to predict the V_{th}-assignment of an instance solely based on its handcrafted features, our framework acts as a graph filter that aggregates an instance's neighboring information to more accurately determine its final V_{th}-assignment through the classification model. Finally, with the validations from the explanation method, we conclude that this work successfully presents a solution to the long lasting V_{th}-assignment problem.

In spite of the superior performance achieved, we still see some limitations of the proposed framework. We observe that *Synopsys PrimeTime* consistently delivers better timing results, and ECO-GNN does not consistently improve the signoff power from the commercial tool. In fact, this is resulted from the modeling errors occurred in the learning process. Although we take the V_{th}-assignments from *Synopsys PrimeTime* as the ground-truths, there always exists a gap between the predictions of our framework and the actual assignments made by the tool. Nonetheless, the goal of this work is *not to replace* commercial signoff tools, but to provide PD engineers a fast, accurate, and reliable estimation of the amount of power recovery to expect from the signoff tools.

8.8.9 Related Works on Improving Chip Design Turnaround Time

Although the main focus of this work is to improve the turn-around time of signoff power optimization, related works have been developed to improve chip design productivity in different endeavors. Recently, as the hardware resources become more powerful, GPU-accelerated algorithms have been developed to significantly improve runtime of different tasks [139]. Previous work [140] develops a novel static timing analysis (STA) engine on a GPU-CPU hybrid system that greatly achieves a 3.6X speed-up on designs with over million of cells. Another work [24] further leverages GPU and deep learning toolkit, PyTorch, to advance placement, where the runtime is improved by 30X without degrading solution quality. As the technology scaling continuously increases the design complexity, in the future, methodologies to improve design turnaround time will be ever-critical.

8.9 Conclusion

In this chapter, we have proposed two different ML modeling approaches in classification and regression aspects to overcome the inherent challenges of the current signoff power optimization flow. The proposed framework, ECO-GNN, can not only provides commercial-quality tool-accurate signoff power optimization results *instantly* on unseen designs, but also has the ability to estimate the power savings of targeted instances using information obtained from local graph structure. Furthermore, we present a GNN-based explanation method to demonstrate the reliability of our framework, which gives designers better reasonings about the predictions of the models.

CHAPTER 9
DOOMED RUN PREDICTION IN PHYSICAL DESIGN BY EXPLOITING SEQUENTIAL FLOW AND GRAPH LEARNING

9.1 Background and Motivation

Modern low-power physical design (PD) implementation flows require designers to perform design space exploration (DSE) in search of the tool configurations (i.e., input parameters of each design stage) that lead to desired end-of-flow power targets [141]. However, with the ever-increasing design complexity driven by Moore's Law, leading-edge industrial designs in advanced technology nodes are pushing PD full-flow runtime into several weeks, which prohibits designers from performing effective power, performance, and area (PPA) exploration. Therefore, a methodology that performs accurate end-of-flow PPA predictions in early design stages is urgently needed, which allows designers to perform efficient DSE by terminating the runs that are doomed to fail early [142].

Recently, the authors of [17] have attempted to tackle the PD doomed run prediction problem by predicting end-of-flow design total negative slack (TNS). However, the literature only focuses on building a prediction model to capture the effect of sweeping target frequency and utilization rate (i.e., two parameters) that are set at the beginning of a flow, where all the other tool parameters are fixed. This makes previous work [17] impractical because modern PD implementation tools such as *Cadence Innovus* and *Synopsys IC Compiler II (ICC2)* offer hundreds of tool parameters throughout the PD flow for designers to explore.

To overcome the above issue and truly build a doomed run prediction framework that will benefit PD engineers, in this chapter, we specifically focus on the aspect of power and develop an end-to-end learning-based model named PD-LSTM using graph neural net-

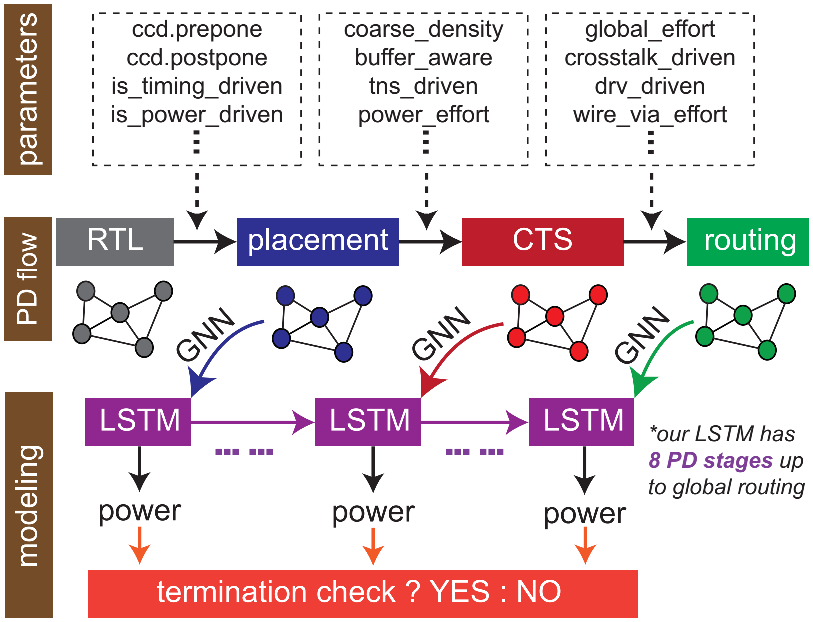


Figure 9.1: Overview of our sequential modeling approach.

works (GNNs) and long short-term memory (LSTM) networks [116]. Our framework predicts end-of-flow total power consumption in early design stages by sequentially encoding the input parameters specified for each intermediate PD stage. The goal of this work is to develop an early-stage power prediction framework that outputs an end-of-flow total power prediction accurately at each intermediate PD stage by incorporating DSE through sequential modeling.

Figure 9.1 demonstrates a high-level view of the proposed modeling approach based on a reference commercial PD tool *Synopsys ICC2*. As shown in the figure, unlike previous work [17] assuming the underlying implementation is fixed, in this work, we accept the fact that designers may explore various parameters at each intermediate PD stage. Note that due to the space limit, Figure 9.1 does not show all intermediate PD stages that we select for modeling. In this work, we select 8 design stages offered publicly by *ICC2* to perform sequential modeling using GNNs and LSTM.

At each modeling stage, our framework PD-LSTM will strive to directly predict the end-of-flow total power value by leveraging all the information obtained up to the current stage along with the tool parameters that designers plan to explore in the future stages

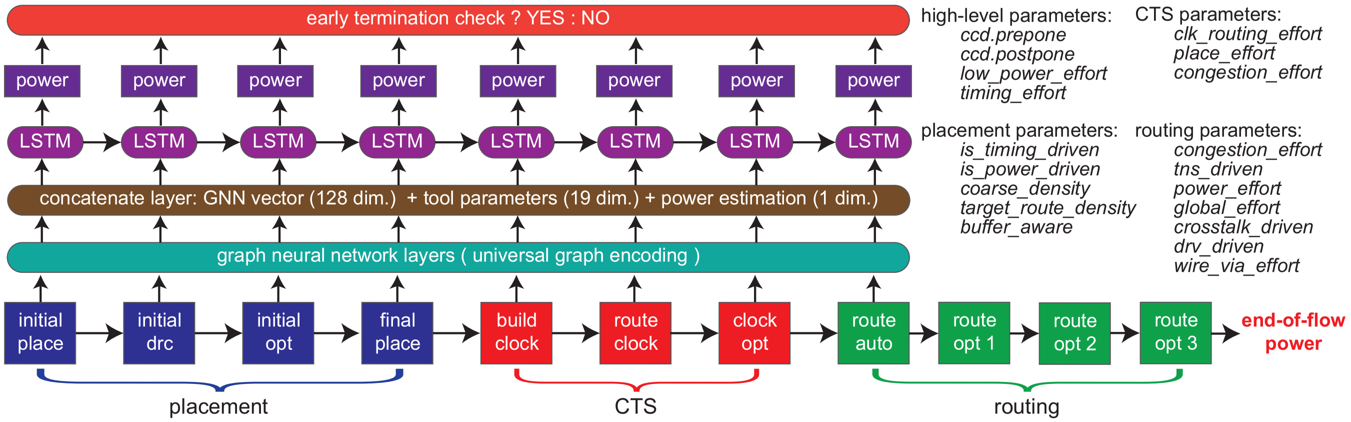


Figure 9.2: Overview of our PD-LSTM framework and the modeling parameters offered by *Synopsys ICC2*. For each stage, our framework will output an end-of-flow total power estimation which can be taken as the criterion of a doomed PD run.

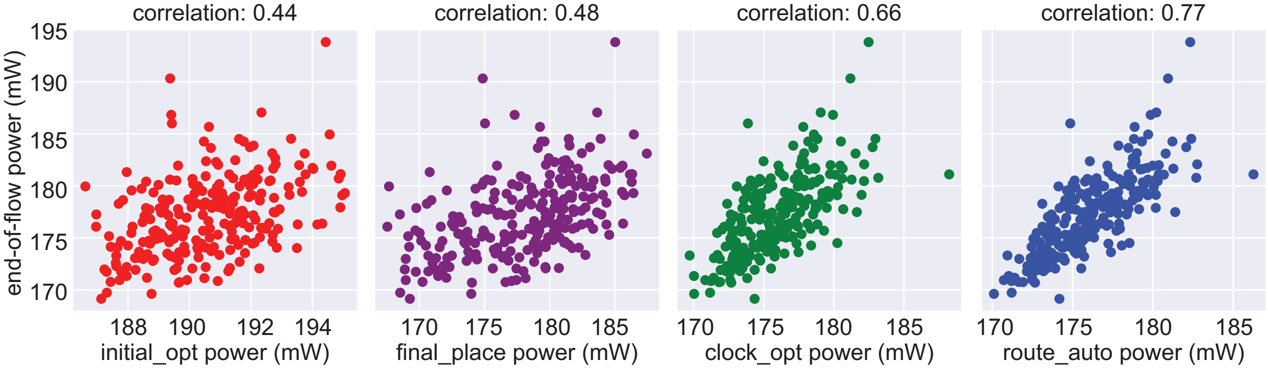


Figure 9.3: *ICC2* correlation analysis of sequential PD stages on a commercial CPU design. We select 4 intermediate PD stages and plot the power estimation of *ICC2* at each stage to the final achieved power value. We observe that power estimations in early design stages are poorly correlate with the end-of-flow power values.

(i.e., a look-ahead mechanism). With the proposed framework, we envision designers to easily perform the following two operations that are not imaginable before: (1) on-the-fly changing input parameters of future PD stages, which enables a more efficient PPA exploration, and (2) performing early termination on the implementations that are doomed to miss the power targets.

Ideally, we only want to perform the end-of-flow prediction as early as possible in the PD flow. However, there is an accuracy and runtime trade-off between a machine learning (ML) model's prediction and its input features collection. With more features collected from late stages, ML models are expected to make better predictions in terms of fidelity and correlation. In this work, we properly balance this trade-off with sequential modeling techniques by iteratively predicting power at each modeling stage. Note that although at each PD stage, the commercial tool will originally output a power prediction of the underlying design, this estimation from the tool is not accurate. In the experiments, we demonstrate that the proposed framework, PD-LSTM, consistently outperforms commercial tool's early stage power estimations and is generalizable to unseen netlists that are not utilized during the training process.

9.2 Overview: PD Flow Modeling

It has been widely acknowledged that GNNs are powerful ML models that encode graph knowledge into meaningful representations. Since VLSI netlists can be naturally represented as hypergraphs, in this work, we leverage GNNs to distill netlist information at each intermediate PD stage. Given that a netlist under a PD implementation is dynamically changing from stage to stage due to buffer insertion or removal, logic restructuring, fanout re-design etc., the goal of applying GNNs in this work is to encode these netlist updates effectively, where the encoded information is further taken as the input of the LSTM framework to perform power estimation.

Figure 9.2 presents a high-level overview of our PD-LSTM framework. The key idea behind is to model the PD flow as a sequential process, and perform on-the-fly end-of-flow total power estimation at each targeted modeling stage. Intuitively, with the proposed framework, designers can perform early termination of an implementation without waiting several weeks to obtain the end-of-flow power results. Furthermore, to enable a more fine-grained DSE for better PPA exploration, we train the proposed framework to incorporate the tool parameters specified by designers. That is, parameter configurations are taken as the inputs of the framework. Unlike previous work [17] which assumes the underlying parameters are fixed, in this work, the proposed framework accepts on-the-fly tuning of the input parameters at each intermediate PD stages.

Numerous parameters are offered by *ICC2* for PPA exploration. In this work, we select 19 parameters by design expertise to perform PD flow modeling which are shown in the right of Figure 9.2. The high-level parameters are known to have profound impact to the overall PD flow. Specifically, “CCD” stands for “concurrent clock data optimization”, which is a key feature of *ICC2* that optimizes clock and data paths for PPA optimization. Finally, we would like to mention that one of our input features to the LSTM framework is the power estimation made by the commercial tool *ICC2*. Although it is known that this power estimation made by the tool is not accurate, we reckon that it may act as a baseline for the model to improve from. In this work, the main objective of PD-LSTM is to provide better end-of-flow power estimations than the commercial tool *ICC2* in early design stages.

9.3 Design of Experiments

9.3.1 Database Construction

The proposed framework adopts supervised learning, which requires a database to be pre-generated for the model to be trained upon. To build the database, we leverage *Synopsys Design Compiler* to synthesize RTL into gate-level netlists, and utilize *Synopsys ICC2* to perform physical implementations. In this work, we utilize 2 commercial multi-core CPU designs and 5 OpenCore designs to perform the experiments. All the designs are synthesized under *TSMC 28nm* technology node at their best achievable frequency. For each synthesized netlist, we generate 200 PD implementations by randomly sampling 19 tool parameters as shown in Figure 9.2. These parameters govern the tool behaviour of various PD stages such as placement, clock tree synthesis (CTS), and routing, which directly impact the final design quality-of-results (QoR).

9.3.2 Database Analysis

Correlation Analysis

Since the goal of this work is to perform high-fidelity end-of-flow power estimation in early stages of the PD flow, the power estimation from the commercial tool of each intermediate PD stage becomes a natural and meaningful baseline for us. Figure 9.3 demonstrates a correlation analysis between the tool estimated power value at selected PD stage and the end-of-flow achieved power value. Note that each dot in the figure represents an actual PD implementation. We observe that as moving toward the end of the design flow, the tool provides more accurate power estimations with higher correlation coefficient (Pearson).

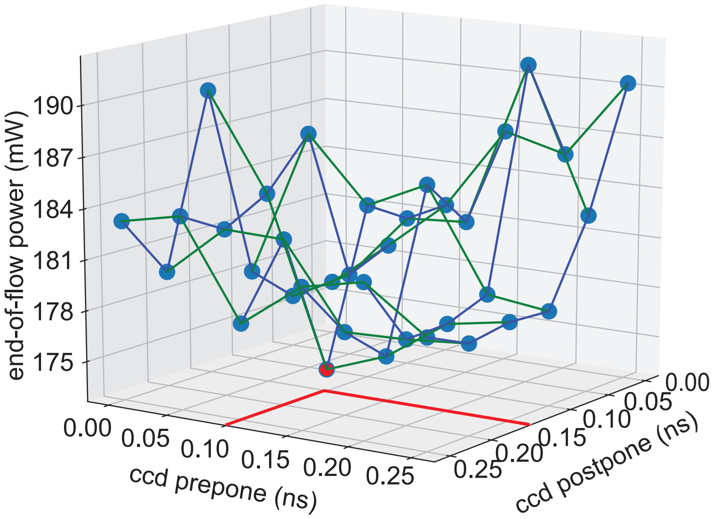


Figure 9.4: CCD parameter sweeping experiment. We observe that the end-of-flow power values can vary as much as 15% by only sweeping two parameters, and the best achieved power occurs under a non-intuitive combination.

However, in the early stages of the design flow (e.g., placement), the power estimations of the tool correlate poorly with the final achieved value. This motivates us to build a framework that can provide accurate power estimation in early stages of the design flow.

CCD Parameter Sweeping

As aforementioned, in *ICC2*, CCD optimization is a key methodology to improve design PPA during many optimization phases throughout the entire design flow. In this work, we select two CCD related parameters: "prepone" and "postpone", which are numerical numbers denoting the maximum reduction and increment, respectively, of the clock latency to registers. These two values are extremely critical, since the change of clock latency will have a direct impact on clock skew that governs the setup and hold margins of timing paths. Ultimately, the power consumption will be affected by the tightness or looseness of the timing margins. For example, buffer insertion is usually applied to fix timing violations, which inevitably increase the internal power.

Figure 9.4 demonstrates a CCD parameter sweeping experiment on a commercial multi-core CPU design. In this experiment, we only sweep around the two CCD parameters

Table 9.1: Initial node features defined for each design instance.

feature name	description
min slack	min_data_delay - max_clock_delay
max slack	max_data_delay - min_clock_delay
wst output slew	maximum transition of output pin
wst input slew	maximum transition of input pin
drv net power	switching power of driving net
switching power	cell switching power
int power	cell internal power
leakage	cell leakage power

introduced earlier: prepone and postpone, from 0 (ns) to 0.25 (ns) with an interval of 0.05, while fixing all other input parameters (shown in Figure 9.2). We observe that the best achieved power occurs when prepone and postpone values are set to 0.1 and 0.15 respectively, which is non-intuitive. Also, it is shown that the achieved power can vary as much as 15% only by simply sweeping these two parameters. Therefore, we believe a framework that predicts end-of-flow power estimation while considering the effect of tool parameters sweeping is highly needed to perform efficient PPA exploration.

9.4 PD-LSTM Algorithms

In this work, we develop a flow-based ML-powered framework named PD-LSTM that performs on-thy-fly end-of-flow total power prediction at each intermediate PD stage by incorporating the dynamic netlist evolution and various parameter specifications. The goal of our framework is to perform early and accurate power predictions by leveraging graph learning and sequential flow modeling. Specifically, there are two main components in our framework, which are the GNN model and the LSTM network that are responsible for netlist encoding and time-series modeling, respectively.

9.4.1 Initial Node Features for Graph Learning

To fully benefit from the graph representation learning conducted by GNNs, prior to the learning process, we have to hand-craft related features for each design instance. Table 9.1

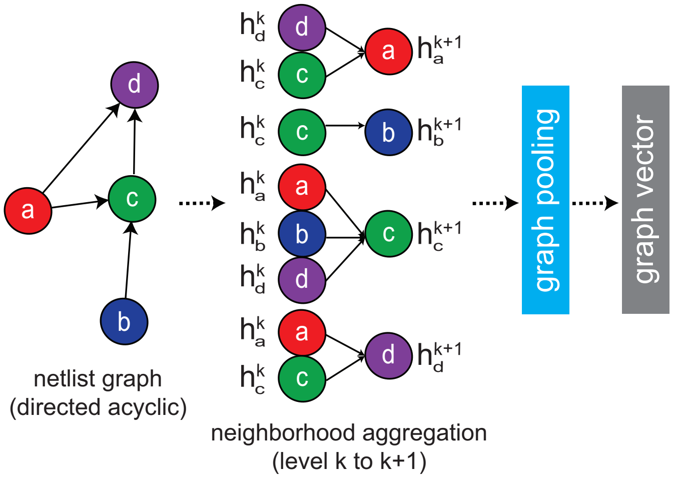


Figure 9.5: Netlist-to-vector encoding using GNNs. The GNN aggregation is performed from $k = 0$ (initial features) to $k = K$ (learned representations) where K is the number of layers.

summarizes the features we utilize for GNN modeling. As shown in the table, besides the power features that are directly related to our power prediction task, for each design instance, we also carefully monitor its timing information by introducing timing-related features. The key reason is that timing and power are highly-related with each other. As aforementioned, buffers or inverters often need to be inserted to fix timing violations, and on the other hand, if a design has enough timing budget, power can often be improved by relaxing timing margins [61]. During graph learning, these initial features of a cell will be transformed into meaningful high-dimensional representations by recursively aggregating neighborhood information.

9.4.2 Graph Embedding

The goal of graph representation learning is to obtain a graph embedding vector that accurately characterizes the underlying netlist. GNNs perform graph representation learning through a messaging passing scheme, where the initial representation vector of a node (i.e., a design instance) can be viewed as a message being recursively transformed and passed onto its neighboring nodes. This message passing process will capture the structural information of the graph and the complex interactions among nodes.

The GNN module is consisted of a set of neuron layers and each of them is responsible to perform aggregation at a specific level k. Figure 9.5 summarizes the netlist to graph vector encoding process using GNN, where for each node we recursively aggregate its neighborhood information from previous level k to obtain the representation in the next level $k+1$. Let h_v^k denote the representation vector of node v at level k, and h_v^0 represent the initial features defined in Table 9.1. Then, following from [37], we design our GNN model to transform the features from level k to level $k+1$ as:

$$\begin{aligned} h_{N_k(v)}^{k-1} &= reduce\_mean\left(\{\mathbf{W}_k^{agg} h_u^{k-1}, \forall u \in N_k(v)\}\right), \\ h_v^k &= \sigma\left(\mathbf{W}_k^{proj} \cdot \text{concat}\left[h_v^{k-1}, h_{N_k(v)}^{k-1}\right]\right), \end{aligned} \tag{9.1}$$

where σ denotes the sigmoid function, $N_k(v)$ denotes the neighboring nodes of node v, W_k^{agg} and W_k^{proj} denote the aggregation and projection matrices at level k respectively, which together represent the neuron layer at level k. Finally, after the last transformation at level $k = K$, we take global mean pooling of $h_v^{k=K}$ across every node in the graph to obtain the final vector g_t in graph-level at timestep t (i.e., an intermediate PD stage) as:

$$g_t = \text{concat}\left[mean\_pooling\left(\left\{h_v^{k=K}\right\}\right), estPower, params\right], \tag{9.2}$$

where "$estPower$" denotes the commercial tool's estimation power at the current stage, and "$params$" represents the tool parameters that the underlying PD implementation explores, which includes both past (i.e., up to timestep t) and future (i.e., after timestep t) exploration. The vectors $\{g_t|_{t=0}^{t=7}\}$ across 8 intermediate PD stages are further taken as the inputs of the downstream LSTM framework.

9.4.3 PD Sequential Modeling using LSTM

Since PD is a sequential process where the output of an intermediate stage is the input of the next stage, the encoded graph vectors $\{g_t|_{t=0}^{t=7}\}$ are highly related with each other. There-

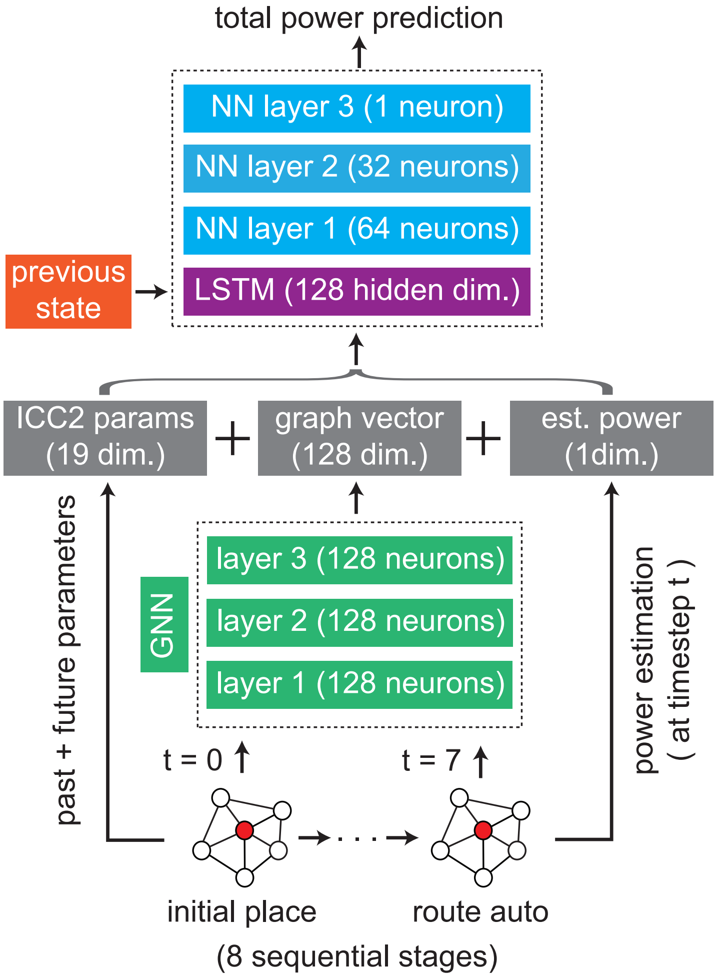


Figure 9.6: Architecture of the proposed PD-LSTM framework.

fore, in this work, we leverage a LSTM network [116] to model such dependency by considering the encoded vectors across various stages as time-domain dependent information. Figure 9.6 demonstrates the detailed architecture of our framework PD-LSTM.Combined with the GNN model presented above, here, we present an end-to-end framework that leverages LSTM architecture to predict end-of-flow total power value at each timestep t based on the encoded graph vectors.

Basically, the LSTM network is a variant of recurrent neural networks (RNNs) that has a backward connection. That is, at each timestep t, the LSTM network will receive not only the inputs from the current time step, but also the outputs from the previous timestep $t-1$ as shown in Figure 9.6. Note that since at the beginning there is no previous output, the state vector is usually set to $\mathbf{0}$. The key idea of LSTM is that the network possesses long-term and short-term memories to learn about which information to be disposed of and which to be kept track of. Specifically, a LSTM cell is consisted of three gates, which are input gate i, forget gate f, and output gate o subject to a timestep t. Given an input sequence g_t, the LSTM performs the encoding procedure as follows

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, a_{t-1}] + b_i), & f_t &= \sigma(W_f \cdot [h_{t-1}, a_{t-1}] + b_f), \\ o_t &= \sigma(W_o \cdot [h_{t-1}, a_{t-1}] + b_o), & \tilde{c_t} &= tanh(W_c[h_{t-1}, x_t] + b_c), \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c_t}, & h_t &= o_t \odot tanh(c_t), \qquad q_t = h_t \end{aligned} \tag{9.3}$$

where $\{W\}$ and $\{b\}$ denote the weights and biases, σ denotes the sigmoid activation function, s_{t-1} denotes the output from the previous time step, and $\odot$ denotes the element-wise multiplication. As shown in Figure 9.6, unlike previous work [**lud1**] that trains the LSTM framework to predict the target value only at the final time step, in this work, our LSTM model outputs a prediction representing the end-of-flow total power estimation at every time step. Finally, in this work, we take the mean-squared-error (MSE) as the loss function to train the model. Note that the proposed framework PD-LSTM is end-to-end differentiable, which means the parameters in both GNN module and the LSTM network are jointly updated in the same computational graph by optimizing the MSE at each timestep t through a gradient descent optimizer.

9.5 Experimental Results

In this section, we demonstrate the achievements of our PD-LSTM framework, which is implemented in *Python3* and the *PyTorch* library. Specifically, we validate our framework on two commercial multi-core CPU designs and five OpenCore benchmarks with a train/test split ratio of 4:3. As aforementioned, for each design, we generate 200 complete PD imple-

Table 9.2: Our benchmarks and their attributes in *TSMC 28nm*.

Design Name	# Nets	# FFs	# Cells	Usage
CPU-A	206,224	22,366	202,791	training
ECG	85,058	14,018	84,127	
VGA	56,279	17,054	56,194	
JPEG	231,934	37,642	219,064	
CPU-B	542,391	47,552	597,085	testing
AES	90,905	10,688	113,168	
LDPC	42,018	2,048	39,377	

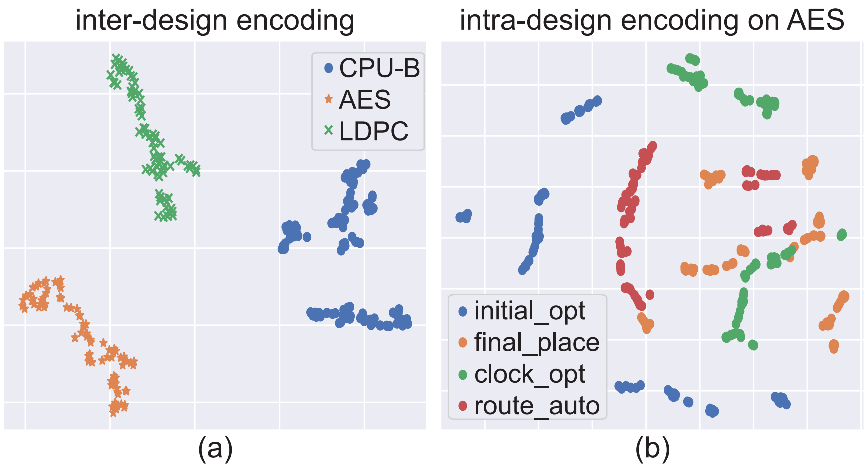


Figure 9.7: t-SNE visualization of GNN netlist encoding. (a) Each dot represents a complete PD run of an unseen netlist. (b) Each dot denotes a netlist graph at a specific PD stage.

mentations by randomly sampling the parameters shown in Figure 9.2. The characteristics of these designs after synthesizing under *TSMC 28nm* are shown in Table 9.2.

9.5.1 GNN Netlist Encoding Results

Graph encoding is the key to the success of the proposed PD-LSTM framework. Here, we leverage the t-distributed stochastic neighboring embedding [44] (t-SNE) algorithm to visualize the embedding results in R^2. The visualization results are shown in Figure 9.7. In Figure 9.7 (a), for each unseen design, we concatenate the encoded graph vector of each modeling stage and utilize t-SNE to visualize the concatenated vector ($128 * 8$ dimensions) in R^2. As for Figure 9.7 (b), within the AES benchmark, we visualize the distribution of the encoded graph vector in 128 dimensions extracted from four selected modeling stages.

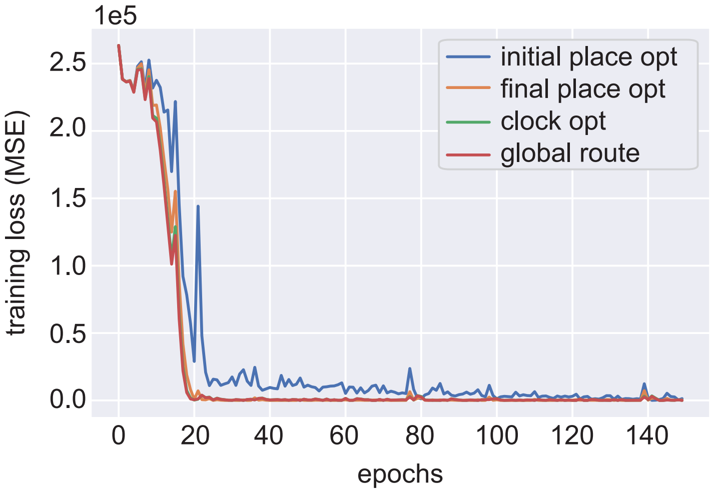


Figure 9.8: Training loss iteration.

In the figure, we observe that our GNN module not only clearly differentiates the characteristics of different designs, but also comprehends features from various modeling stages. Hence, we conclude that the proposed GNN model is generalizable.

9.5.2 Sequential Learning Results

Figure 9.8 demonstrates the training loss iteration of the selected modeling stages. We observe that the losses of early design stages require more iterations to reach convergence. Table 9.3 demonstrates the prediction results of the proposed PD-LSTM framework on the unseen netlists that are not utilized in the training process. In this work, our PD-LSTM has 8 modeling stages and for each stage, the framework will output an end-of-flow power estimation as *ICC2*. NRMSE denotes the normalized root-mean-squared error and is calculated by normalizing the RMSE that inherently comes with a "unit" (e.g., mW) by the difference between the maximum and minimum ground truth values (i.e., $NRMSE = \frac{RMSE}{power_{max} - power_{min}}$). NRMSE is a popular comparison metric that removes the effect of unit scale. As shown in the table, we observe that the predictions made by PD-LSTM consistently outperform the ones made by *ICC2* starting from early stages of the design flow in terms of correlation coefficient (CC), which directly proves that the proposed framework delivers better end-of-flow total power estimation. Finally, as moving

Table 9.3: PD-LSTM end-of-flow prediction results on "unseen" designs. CC denotes the Pearson correlation coefficient. NRMSE denotes the accuracy of our model. All metrics are computed against end-of-flow total power values.

PD stage (avg time)	unseen design	NRMSE (%)	ICC2 CC	our CC
initial place (3%)	CPU-B	29.8	0.42	0.46
	AES	24.7	0.26	0.5
	LDPC	21.2	0.18	0.37
initial drc (4%)	CPU-B	22.1	0.43	0.58
	AES	28.6	0.25	0.52
	LDPC	27.3	0.18	0.38
initial opt (7%)	CPU-B	18.5	0.42	0.72
	AES	12.1	0.32	0.68
	LDPC	12.9	0.31	0.66
final place (22%)	CPU-B	11.2	0.45	0.81
	AES	9.7	0.35	0.86
	LDPC	9.2	0.32	0.72
build clock (6%)	CPU-B	8.2	0.41	0.89
	AES	7.1	0.47	0.9
	LDPC	8.7	0.43	0.88
route clock (7%)	CPU-B	5.9	0.42	0.94
	AES	6.4	0.76	0.92
	LDPC	5.8	0.74	0.93
clock opt (12%)	CPU-B	5.2	0.65	0.95
	AES	6.4	0.96	0.96
	LDPC	3.9	0.92	0.95
route auto (8%)	CPU-B	4.1	0.75	0.98
	AES	5.3	0.96	0.97
	LDPC	3.7	0.94	0.97

*remaining routing optimization stages take 31% of runtime.

closer to the end of the design flow, we see that the power predictions become more accurate for both *ICC2* and the proposed framework. This is expected because with more features collected from latter stages of the design flow, ML models are expected to make better predictions in terms of fidelity and correlation.

9.6 Conclusion

In this work, we have proposed PD-LSTM, a flow-based framework that leverages graph learning and sequential modeling to perform end-of-flow total power estimation starting from early PD stages. The proposed framework consistently demonstrates better power estimation results across various intermediate modeling stages than the reference commercial PD tool *ICC2*. In spite of the superior prediction results achieved, in the future, we aim to explore the possibilities to leverage PD-LSTM to perform on-the-fly PPA optimization by dynamically searching for optimized parameters.

CHAPTER 10
CONCLUSION

10.1 A Machine Learning Powered Tier Partitioning Framework for Monolithic 3D ICs

In this study, we have presented an unsupervised, graph-learning-based, tier partitioning framework named TP-GNN to mitigate the significant drawbacks of the existing tier partitioning algorithm, the bin-based min-cut algorithm, in the state-of-the-art M3D implementation flows. First, we proposed a hierarchy-aware edge contraction algorithm to reduce 3D routing overhead occurred in the bin-based min-cut partitioning algorithm by merging chosen nodes into super-nodes. Then, we consider the classic tier partitioning problem as a clustering problem and solved it using GNNs. The graph representation learning provides the freedom for designers to deal with various partitioning objectives, and the unsupervised learning promises the generality of our framework. We validate our framework using various styles of M3D design flows, including partitioning-first, partitioning-last, and heterogeneous. We observe significant PPA improvements over the bin-based min-cut algorithm across numerous industrial designs in the TSMC $28nm$ technology.

10.2 VLSI Placement Optimization via PPA-Directed Self-Supervised Deep Graph Clustering

In this study, we have developed the first PPA-directed, self-supervised placement optimization framework that directly formulates PPA metrics as ML loss functions, and optimizes them through gradient descent. Given a globally-placed netlist as input, our framework, PPA-GNN, generates the cell clustering constraints that can be taken as additional constrains to a commercial placer in order to improve the underlying placement during the placement optimization phase. Unlike previous works that rely on a two-step process to generate such constraints, PPA-GNN is end-to-end differentiable. That is, the node representation learning and the clustering assignments are jointly updated through optimization of PPA metrics. We validate our framework using industrial million-scale design in advanced technologies, where we not only demonstrate that PPA-GNN produces immediate PPA improvements at the placement stage, but also show that the improvements last firmly to the post-route stage.

10.3 Bridging Open-Source and Commercial Placers using Generative Adversarial Networks and Transfer Learning

In this study, we have presented DREAM-GAN, which advances the renowned open-source placer DREAMPlace using generative adversarial learning. In our settings, we consider DREAMPlace as a generator, and develop two discriminators with one using CNNs and the other using GNNs to characterize bin-density maps and netlist information, respectively. At each placement iteration, we not only train DREAMPlace to optimize conventional metrics: wirelength and density, but also improve the underlying cell locations to optimize the similarity scores output by the proposed discriminators. Experimental results demonstrate DREAM-GAN outperforms the vanilla DREAMPlace across every critical PPA metric in 6 industrial designs using under advanced technologies.

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

In this study, we have presented GAN-CTS, which is a generative adversarial framework for clock tree prediction and optimization. First, to precisely characterize distinct designs, we leverage transfer learning to extract netlist features directly from placement images. Second, we perform regression learning using various methods to predict the target CTS outcomes and demonstrate that the proposed multi-task learning approach achieves better accuracy than the meta-modeling method adopted by previous works. To fully benefit from the predictions made by our framework, we further quantitatively interpret the importance of each CTS input parameter subject to various design objectives through attribution-based learning. Finally, generative adversarial learning is leveraged to optimize the target clock metrics with the guidance provided by the pre-trained regression model. To substantiate the generality of our framework, we perform validations on four unseen netlists that are not utilized in the training process. Experimental results conducted on real-world designs demonstrate that our framework significantly outperforms the default CTS process inherent in the commercial tool.

10.5 RL-Sizer: VLSI Gate Sizing for Timing Optimization using Deep Reinforcement Learning

In this study, we have developed RL-Sizer, an RL agent that performs gate sizing for timing optimization. Particularly, we re-formulate the traditional gate sizing process as a control problem, and solve it using RL algorithms. RL-sizer is integrated with an industry-leading commercial EDA tool, Synopsys ICC2. With a completely different approach, RL-Sizer achieves better, if not equal, performance as the tool's native sizing algorithm that has been developed relentlessly over the past decade. Experimental results on industrial designs in $5-12nm$ technologies demonstrate that RL-Sizer outperforms the native sizing algorithm in 4 out of 6 designs with a significant margin.

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

In this study, we investigate a brand new research problem. That is, balancing clock-path and data-path optimization strategies in commercial tools through endpoint prioritization, which is motivated by the fact that commercial tools always adopt fixed recipes to resolve violating endpoints, neglecting the fact that not all endpoints are equal as some are more easily fixed by data-path optimization techniques, while others, clock-path. We propose RL-CCD, an attention-based, self-supervised RL framework that prioritizes endpoints for useful skew optimization using margin. Experimental results on 19 industrial designs in $5 - 16nm$ technologies clearly demonstrate that our framework significantly outperforms the default CCD engine across every benchmark, with TNS reduction up to 64%.

10.7 ECO-GNN: Signoff Power Prediction using Graph Neural Networks with Subgraph Approximation

In this study, we explore supervised learning techniques to improve the signoff power optimization process. Particularly, we have proposed two different modeling approaches in terms of classification and regression to improve the turn-around time of the optimization step. We demonstrate that the proposed framework, ECO-GNN, can not only provides commercial-quality tool-accurate signoff power optimization results *instantly* on unseen designs, but also has the ability to estimate the power savings of targeted instances using information obtained from local subgraph. Furthermore, instead of blindly relying on the predictions, we explore explanation method to interpret the rationales behind the predictions, which makes the proposed framework more trustworthy.

10.8 Doomed Run Prediction in Physical Design by Exploiting Sequential Flow and Graph Learning

In this study, we present PD-LSTM, a PD doomed run prediction framework that performs end-of-flow total power prediction from early stages of the design flow. As PD is a sequential process where netlists undergo several optimizations and physical updates from stage to stage, PD-LSTM leverages GNNs and LSTM for effective sequential modeling. Utilizing t-SNE, a dimension reduction technique, we demonstrate that the graph embeddings produced by our GNN module not only have the capability to differentiate between diverse netlists, but also to distinguish the same netlist at varying stages. In the experiments, we show that PD-LSTM consistently achieves better power estimations compared with the default estimation of the reference commercial tool.

10.9 Concluding Remarks

We believe this research have demonstrated the promising potentials of advancing modern PD implementations flows for 2D and 3D ICs. Particularly, we show that ML algorithms can not only be used as PPA predictors but directly as optimizers. Undoubtedly, more efforts to further improve the chip design productivity and the final outcomes in the post-Moore era, however, we reckon that we have shown the new horizons of revolutionizing PD with ML.

REFERENCES

[1] A. B. Kahng and Z. Wang, “Ml for design qor prediction,” in *Machine Learning Applications in Electronic Design Automation*, Springer, 2022, pp. 3–33.

[2] S. Sahni and A. Bhatt, “The complexity of design automation problems,” in *Proceedings of the 17th Design Automation Conference*, 1980, pp. 402–411.

[3] G. Huang *et al.*, “Machine learning for electronic design automation: A survey,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 26, no. 5, pp. 1–46, 2021.

[4] Y.-C. Lu and S. K. Lim, “On advancing physical design using graph neural networks,” in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–7.

[5] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, “A comprehensive survey on graph neural networks,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[6] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, “Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[7] Y.-C. Lu, S. Pentapati, and S. K. Lim, “The law of attraction: Affinity-aware placement optimization using graph neural networks,” in *Proceedings of the 2021 International Symposium on Physical Design*, 2021, pp. 7–14.

[8] Y.-C. Lu, T. Yang, S. K. Lim, and H. Ren, “Placement optimization via ppa-directed graph clustering,” in *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, 2022.

[9] A. Mirhoseini *et al.*, “A graph placement methodology for fast chip design,” *Nature*, vol. 594, no. 7862, pp. 207–212, 2021.

[10] A. Agnesina, K. Chang, and S. K. Lim, “Vlsi placement parameter optimization using deep reinforcement learning,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[11] 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)*, IEEE, 2021, pp. 733–738.

[12] Y.-C. Lu, S. Nath, S. S. K. Pentapati, and S. K. Lim, “A fast learning-driven signoff power optimization framework,” in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2020, pp. 1–9.

[13] U. Mallappa and C.-K. Cheng, “Gra-lpo: Graph convolution based leakage power optimization,” in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2021, pp. 697–702.

[14] Y. Zhang, H. Ren, and B. Khailany, “Grannite: Graph neural network inference for transferable power estimation,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[15] K. K.-C. Chang, C.-Y. Chiang, P.-Y. Lee, and I. H.-R. Jiang, “Timing macro modeling with graph neural networks,” in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1219–1224.

[16] 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 *Proceedings of the 59th Annual Design Automation Conference 2022*, ACM, 2022.

[17] Y.-C. Lu, S. Nath, V. Khandelwal, and S. K. Lim, “Doomed run prediction in physical design by exploiting sequential flow and graph learning,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2021, pp. 1–9.

[18] Y.-C. Lu, W.-T. Chan, V. Khandelwal, and S. K. Lim, “Driving early physical synthesis exploration through end-of-flow total power prediction,” in *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*, 2022.

[19] Z. Xie, R. Liang, X. Xu, J. Hu, Y. Duan, and Y. Chen, “Net 2: A graph attention network method customized for pre-placement net length estimation,” in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2021, pp. 671–677.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[21] I. Goodfellow *et al.*, “Generative adversarial networks,” *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[22] P. Dhariwal and A. Nichol, “Diffusion models beat gans on image synthesis,” *Advances in Neural Information Processing Systems*, vol. 34, pp. 8780–8794, 2021.

[23] D. Silver *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[24] Y. Lin *et al.*, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 4, pp. 748–761, 2020.

[25] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[26] J. Knechtel and J. Lienig, "Physical design automation for 3d chip stacks: Challenges and solutions," in *Proceedings of the 2016 on International Symposium on Physical Design*, ACM, 2016.

[27] K. Arabi, K. Samadi, and Y. Du, "3d vlsi: A scalable integration beyond 2d," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ACM, 2015.

[28] M. Vinet *et al.*, "Monolithic 3d integration: A powerful alternative to classical 2d scaling," in *2014 SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, IEEE, 2014, pp. 1–3.

[29] Y.-C. Lu, S. Pentapati, L. Zhu, G. Murali, K. Samadi, and S. K. Lim, "A machine learning-powered tier partitioning methodology for monolithic 3-d ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4575–4586, 2021.

[30] O. Billoint *et al.*, "A comprehensive study of monolithic 3d cell on cell design using commercial 2d tool," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2015.

[31] S. Panth, K. Samadi, Y. Du, and S. K. Lim, "Shrunk-2-d: A physical design methodology to build commercial-quality monolithic 3-d ics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 10, 2017.

[32] B. W. Ku, K. Chang, and S. K. Lim, "Compact-2d: A physical design methodology to build commercial-quality face-to-face-bonded 3d ics," in *Proceedings of the 2018 International Symposium on Physical Design*, ACM, 2018.

[33] P. Vanna-Iampikul, C. Shao, Y.-C. Lu, S. Pentapati, and S. K. Lim, "Snap-3d: A constrained placement-driven physical design methodology for face-to-face-bonded 3d ics," in *Proceedings of the 2021 International Symposium on Physical Design*, 2021, pp. 39–46.

[34] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, 1982.

[35] K. Chang *et al.*, “Cascade2d: A design-aware partitioning approach to monolithic 3d ic with 2d commercial tools,” in *Proceedings of the 35th International Conference on Computer-Aided Design*, ACM, 2016.

[36] S. S. K. Pentapati, K. Chang, V. Gerousis, R. Sengupta, and S. K. Lim, “Pin-3d: A physical synthesis and post-layout optimization flow for heterogeneous monolithic 3d ics,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[37] W. Hamilton, Z. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *Advances in Neural Information Processing Systems*, 2017.

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

[39] R. C. De Amorim and B. Mirkin, “Minkowski metric, feature weighting and anomalous cluster initializing in k-means clustering,” *Pattern Recognition*, 2012.

[40] S. K. Samal, D. Nayak, M. Ichihashi, S. Banna, and S. K. Lim, “Tier partitioning strategy to mitigate beol degradation and cost issues in monolithic 3d ics,” in *Proceedings of the 35th International Conference on Computer-Aided Design*, ACM, 2016, p. 129.

[41] J. Balkind *et al.*, “Openpiton: An open source manycore research framework,” in *ACM SIGARCH Computer Architecture News*, ACM, 2016.

[42] K. Asanovic *et al.*, “The rocket chip generator,” *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17*, 2016.

[43] M. M. Ozdal, C. Amin, A. Ayupov, S. Burns, G. Wilke, and C. Zhuo, “The ispd-2012 discrete cell sizing contest and benchmark suite,” in *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, ACM, 2012, pp. 161–164.

[44] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[45] J. Cong and G. Luo, “A multilevel analytical placement for 3d ics,” in *2009 Asia and South Pacific Design Automation Conference*, IEEE, 2009, pp. 361–366.

[46] G. Luo, Y. Shi, and J. Cong, “An analytical placement framework for 3-d ics and its extension on thermal awareness,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 4, 2013.

[47] M.-K. Hsu, V. Balabanov, and Y.-W. Chang, "Tsv-aware analytical placement for 3-d ic designs based on a novel weighted-average wirelength model," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2013.

[48] H. Sarhan, S. Thuries, O. Billoint, and F. Clermidy, "An unbalanced area ratio study for high performance monolithic 3d integrated circuits," in *2015 IEEE Computer Society Annual Symposium on VLSI.*

[49] A. B. Kahng, "Advancing placement," in *Proceedings of the 2021 International Symposium on Physical Design*, 2021, pp. 15–22.

[50] X. Gao *et al.*, "Congestion and timing aware macro placement using machine learning predictions from different data sources: Cross-design model applicability and the discerning ensemble," in *Proceedings of the 2022 International Symposium on Physical Design*, 2022, pp. 195–202.

[51] D. Vashisht *et al.*, "Placement in integrated circuits using cyclic reinforcement learning and simulated annealing," *arXiv preprint arXiv:2011.07577*, 2020.

[52] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International conference on machine learning*, PMLR, 2016.

[53] M.-K. Hsu *et al.*, "Ntuplace4h: A novel routability-driven placement algorithm for hierarchical mixed-size circuit designs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 12, pp. 1914–1927, 2014.

[54] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 9, pp. 1717–1730, 2018.

[55] X. He, Y. Wang, Y. Guo, and E. F. Young, "Ripple 2.0: Improved movement of cells in routability-driven placement," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 22, no. 1, pp. 1–26, 2016.

[56] D. Z. Pan, B. Halpin, and H. Ren, "Timing-driven placement," *Handbook of Algorithms for Physical Design Automation*, pp. 423–446, 2008.

[57] L. Hagen and A. B. Kahng, "A new approach to effective circuit clustering," in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 1992, pp. 422–427.

[58] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[59] Z. Xie *et al.*, “Routenet: Routability prediction for mixed-size designs using convolutional neural network,” in *2018 IEEE/ACM International Conference on Computer-Aided Design*.

[60] 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)*, IEEE, 2022.

[61] 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*, 2021, pp. 83–90.

[62] W. Jin, L. Chen, S. Sadiqbatcha, S. Peng, and S. X.-D. Tan, “Emgraph: Fast learning-based electromigration analysis for multi-segment interconnect using graph convolution networks,” in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2021, pp. 919–924.

[63] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[64] X. Liu *et al.*, “Self-supervised learning: Generative or contrastive,” *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[65] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[66] C.-C. Huang, C.-H. Chiou, K.-H. Tseng, and Y.-W. Chang, “Detailed-routing-driven analytical standard-cell placement,” in *The 20th Asia and South Pacific Design Automation Conference*, IEEE, 2015, pp. 378–383.

[67] C. E. Shannon, “A mathematical theory of communication,” *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[68] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini, “Gap: Generalizable approximate graph partitioning framework,” *arXiv:1903.00614*, 2019.

[69] Y. Cheon, P.-H. Ho, A. B. Kahng, S. Reda, and Q. Wang, “Power-aware placement,” in *Proceedings of the 42nd annual Design Automation Conference*, 2005, pp. 795–800.

[70] C. Inc., *Innovus user guide*, 2022.

[71] S. Inc., *Icc2 user guide*, 2022.

[72] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems'19*,

[73] X. He *et al.*, "Ripple 2.0: High quality routability-driven placement via global router integration," in *Proceedings of the 50th Annual Design Automation Conference*, 2013, pp. 1–6.

[74] L. Liu, B. Fu, M. D. Wong, and E. F. Young, "Xplace: An extremely fast and extensible global placement framework," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1309–1314.

[75] Y.-C. Lu, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, "Gan-cts: A generative adversarial framework for clock tree prediction and optimization," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2019, pp. 1–8.

[76] Z. Ying *et al.*, "Hierarchical graph representation learning with differentiable pooling," *2018 Advances in neural information processing systems*,

[77] 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.

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

[79] 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.

[80] 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.

[81] 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.

[82] S. Koh, Y. Kwon, and Y. Shin, "Pre-layout clock tree estimation and optimization using artificial neural network," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 193–198.

[83] 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.

[84] 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*, vol. 4, no. 4, pp. 588–598, 2017.

[85] 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.

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

[87] 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.

[88] D. Broomhead and D. Lowe, "Multivariable functional interpolation and adaptive networks, complex systems, vol. 2," 1988.

[89] N. Cressie, "Fitting variogram models by weighted least squares," *Journal of the international Association for mathematical Geology*, vol. 17, no. 5, pp. 563–586, 1985.

[90] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, ACM, 2016.

[91] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," in *Advances in neural information processing systems*, 2018, pp. 6638–6648.

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

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

[94] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009.

[95] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Advances in neural information processing systems*, vol. 25, pp. 2951–2959, 2012.

[96] Y. Ma, Z. Yu, and B. Yu, "Cad tool design space exploration via bayesian optimization," *arXiv preprint arXiv:1912.06460*, 2019.

[97] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.

[98] N. Stander and K. Craig, "On the robustness of a simple domain reduction scheme for simulation-based optimization," *Engineering Computations*, 2002.

[99] F. Chollet *et al.*, *Keras*, 2015.

[100] G. V. Trunk, "A problem of dimensionality: A simple example," *IEEE Transactions on pattern analysis and machine intelligence*, no. 3, pp. 306–307, 1979.

[101] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[102] W. Ning, "Strongly np-hard discrete gate-sizing problems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 8, pp. 1045–1051, 1994.

[103] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[104] A. Mirhoseini *et al.*, "Chip placement with deep reinforcement learning," *arXiv preprint arXiv:2004.10746*, 2020.

[105] H. Wang, K. Wang, J. Yang, N. Sun, H. Lee, and S. Han, "Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *ACM/IEEE 57th Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[106] H. Liao, W. Zhang, X. Dong, B. Poczos, K. Shimada, and L. Burak Kara, "A deep reinforcement learning approach for global routing," *Journal of Mechanical Design*, vol. 142, no. 6, 2020.

[107] G. Pasandi, S. Nazarian, and M. Pedram, "Approximate logic synthesis: A reinforcement learning-based technology mapping approach," in *20th International Symposium on Quality Electronic Design (ISQED)*, IEEE, 2019, pp. 26–32.

[108] C.-P. Chen, C. C. Chu, and D. Wong, "Fast and exact simultaneous gate and wire sizing by lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 7, pp. 1014–1025, 1999.

[109] S. Roy, D. Liu, J. Um, and D. Z. Pan, “Osfa: A new paradigm of gate-sizing for power/performance optimizations under multiple operating conditions,” in *Design Automation Conference*, ACM, 2015, p. 129.

[110] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, “Paragraph: Layout parasitics and device parameter prediction using graph neural networks,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, pp. 1–6.

[111] A. Vaswani *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.

[112] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” *Machine learning*, vol. 8, no. 3-4, 1992.

[113] 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)*.

[114] 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*, IEEE, 2014, pp. 504–509.

[115] 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.

[116] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[117] O. Vinyals, M. Fortunato, and N. Jaitly, “Pointer networks,” *Advances in neural information processing systems*, vol. 28, 2015.

[118] S. Patanjali, M. Patnaik, S. Potluri, and V. Kamakoti, “Mltimer: Leakage power minimization in digital circuits using machine learning and adaptive lazy timing analysis,” *Journal of Low Power Electronics*, vol. 14, no. 2, pp. 285–301, 2018.

[119] S. Sirichotiyakul, T. Edwards, C. Oh, and J. Zuo, *Primetime user guide: Fundamentals*, 2005.

[120] S. Mok, J. Lee, and P. Gupta, “Discrete sizing for leakage power optimization in physical design: A comparative study,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 18, no. 1, p. 15, 2013.

[121] R. Ying, D. Bourgeois, J. You, M. Zitnik, and J. Leskovec, "Gnn explainer: A tool for post-hoc explanation of graph neural networks," *arXiv preprint arXiv:1903.03894*, 2019.

[122] J. Hu, A. B. Kahng, S. Kang, M.-C. Kim, and I. L. Markov, "Sensitivity-guided metaheuristics for accurate discrete gate sizing," in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 233–239.

[123] M. Rahman and C. Sechen, "Post-synthesis leakage power minimization," in *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2012, pp. 99–104.

[124] T. Reimann, G. Posser, G. Flach, M. Johann, and R. Reis, "Simultaneous gate sizing and vt assignment using fanin/fanout ratio and simulated annealing," in *2013 IEEE International Symposium on Circuits and Systems (ISCAS2013)*, IEEE, 2013, pp. 2549–2552.

[125] M. Hashimoto, H. Onodera, and K. Tamaru, "A power optimization method considering glitch reduction by gate sizing," in *Proceedings of the 1998 international symposium on Low power electronics and design*, 1998, pp. 221–226.

[126] Y. Liu and J. Hu, "A new algorithm for simultaneous gate sizing and threshold voltage assignment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 2, pp. 223–234, 2010.

[127] L. Li, P. Kang, Y. Lu, and H. Zhou, "An efficient algorithm for library-based cell-type selection in high-performance low-power designs," in *Proceedings of the International Conference on Computer-Aided Design*, ACM, 2012, pp. 226–232.

[128] J. Singh, V. Nookala, Z.-Q. Luo, and S. Sapatnekar, "Robust gate sizing by geometric programming," in *Proceedings. 42nd Design Automation Conference, 2005.*, IEEE, 2005, pp. 315–320.

[129] S. Roy, W. Chen, C. C.-P. Chen, and Y. H. Hu, "Numerically convex forms and their application in gate sizing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 9, pp. 1637–1647, 2007.

[130] M. M. Ozdal, S. Burns, and J. Hu, "Gate sizing and device technology selection algorithms for high-performance industrial designs," in *Proceedings of the International Conference on Computer-Aided Design*, IEEE Press, 2011, pp. 724–731.

[131] A. Sharma, D. Chinnery, S. Bhardwaj, and C. Chu, "Fast lagrangian relaxation based gate sizing using multi-threading," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2015, pp. 426–433.

[132] S. Bao, “Optimizing leakage power using machine learning,” *CS Department, Stanford University*, 2010.

[133] K. Wang and P. Cao, “A graph neural network method for fast eco leakage power optimization,” in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2022, pp. 196–201.

[134] W. Lee, Y. Kwon, and Y. Shin, “Fast eco leakage optimization using graph convolutional network,” in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, 2020, pp. 187–192.

[135] J. B. White and P. Sadayappan, “On improving the performance of sparse matrix-vector multiplication,” in *Proceedings Fourth International Conference on High-Performance Computing*, IEEE, 1997, pp. 66–71.

[136] D. Chen, Y. Lin, W. Li, P. Li, J. Zhou, and X. Sun, “Measuring and relieving the over-smoothing problem for graph neural networks from the topological view,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, 2020, pp. 3438–3445.

[137] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, “Graph neural networks: A review of methods and applications,” *arXiv preprint arXiv:1812.08434*, 2018.

[138] T. M. Fruchterman and E. M. Reingold, “Graph drawing by force-directed placement,” *Software: Practice and experience*, vol. 21, no. 11, pp. 1129–1164, 1991.

[139] T.-W. Huang, D.-L. Lin, C.-X. Lin, and Y. Lin, “Taskflow: A lightweight parallel and heterogeneous task graph computing system,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 6, pp. 1303–1320, 2021.

[140] Z. Guo, T.-W. Huang, and Y. Lin, “Gpu-accelerated static timing analysis,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[141] D. MacMillen, R. Camposano, D. Hill, and T. W. Williams, “An industrial view of electronic design automation,” *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 19, no. 12, pp. 1428–1448, 2000.

[142] A. B. Kahng, “New directions for learning-based ic design tools and methodologies,” in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2018, pp. 405–410.

PUBLICATIONS

This dissertation is based on and/or related to the works and results presented in the following publications in print:

[1] Y.-C. **Lu**, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, “Gan-cts: A generative adversarial framework for clock tree prediction and optimization,” in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2019, 1–8 (**Best Paper Nomination**).

[2] Y.-C. **Lu**, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim, “Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics,” in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, 2020, 1–6 (**Best Paper Nomination**).

[3] Y.-C. **Lu**, S. Nath, S. S. K. Pentapati, and S. K. Lim, “A fast learning-driven signoff power optimization framework,” in *2020 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, 2020, pp. 1–9.

[4] Y.-C. **Lu**, S. Pentapati, and S. K. Lim, “Vlsi placement optimization using graph neural networks,” in *Proceedings of the 34th Advances in Neural Information Processing Systems (NeurIPS) Workshop on ML for Systems, Virtual*, 2020, pp. 6–12.

[5] Y.-C. **Lu**, S. Pentapati, and S. K. Lim, “The law of attraction: Affinity-aware placement optimization using graph neural networks,” in *Proceedings of the 2021 International Symposium on Physical Design*, 2021, 7–14 (**Best Paper Nomination**).

[6] 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)*, IEEE, 2021, pp. 733–738.

[7] Y.-C. **Lu**, S. Nath, V. Khandelwal, and S. K. Lim, “Doomed run prediction in physical design by exploiting sequential flow and graph learning,” in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, IEEE, 2021, pp. 1–9.

[8] Y.-C. **Lu**, J. Lee, A. Agnesina, K. Samadi, and S. K. Lim, “A clock tree prediction and optimization framework using generative adversarial learning,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 9, pp. 3104–3117, 2021.

[9] Y.-C. **Lu**, S. Pentapati, L. Zhu, G. Murali, K. Samadi, and S. K. Lim, “A machine learning-powered tier partitioning methodology for monolithic 3-d ics,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4575–4586, 2021.

[10] Y.-C. **Lu** and S. K. Lim, “On advancing physical design using graph neural networks,” in *2022 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2022, pp. 1–7.

[11] Y.-C. **Lu**, W.-T. Chan, V. Khandelwal, and S. K. Lim, “Driving early physical synthesis exploration through end-of-flow total power prediction,” in *2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD)*, IEEE, 2022, pp. 97–102.

[12] Y.-C. **Lu**, T. Yang, S. K. Lim, and H. Ren, “Placement optimization via ppa-directed graph clustering,” in *2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD)*, IEEE, 2022, 1–6 (**Best Student Paper Award**).

[13] Y.-C. **Lu**, S. Nath, S. Pentapati, and S. K. Lim, “Eco-gnn: Signoff power prediction using graph neural networks with subgraph approximation,” *ACM Transactions on Design Automation of Electronic Systems*, 2022.

[14] Y.-C. **Lu**, H. Ren, H.-H. Hsiao, and S. K. Lim, “Dream-gan: Advancing dreamplace towards commercial-quality using generative adversarial learning,” in *Proceedings of the 2023 International Symposium on Physical Design*, 2023.

[15] Y.-C. **Lu**, W.-T. Chan, D. Guo, S. Kundu, V. Khandelwal, and S. K. Lim, “Rl-ccd: Concurrent clock and data optimization using attention-based self-supervised reinforcement learning,” in *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 2023.

In addition, the author has completed works unrelated to this dissertation presented in the following publications in print:

[1] S.-C. Hung, Y.-C. **Lu**, S. K. Lim, and K. Chakrabarty, “Power supply noise-aware scan test pattern reshaping for at-speed delay fault testing of monolithic 3d ics,” in *2020 IEEE 29th Asian Test Symposium (ATS)*, IEEE, 2020, pp. 1–6.

[2] P. Vanna-Iampikul, C. Shao, Y.-C. **Lu**, S. Pentapati, and S. K. Lim, “Snap-3d: A constrained placement-driven physical design methodology for face-to-face-bonded 3d ics,” in *Proceedings of the 2021 International Symposium on Physical Design*, 2021, pp. 39–46.

[3] S.-C. Hung, Y.-C. **Lu**, S. K. Lim, and K. Chakrabarty, “Power supply noise-aware at-speed delay fault testing of monolithic 3-d ics,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 11, pp. 1875–1888, 2021.

[4] P. Vanna-Iampikul *et al.*, “Snap-3d: A constrained placement-driven physical design methodology for high performance 3d ics,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[5] L. Zhu, N. E. Bethur, Y.-C. **Lu**, Y. Cho, Y. Im, and S. K. Lim, “3d ic tier partitioning of memory macros: Ppa vs. thermal tradeoffs,” in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2022, pp. 1–6.

VITA

Yi-Chen Lu was born in Taipei, Taiwan in 1994. He received the B.S. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, in 2017, and the M.S. degree in Electrical and Computer Engineering from Georgia Institute of Technology, Atlanta, GA, USA, in 2019, where is is currently a Ph.D. candidate.

He has been working as a graduate research assistant in the Georgia Tech Computer Aided Design (GTCAD) Laboratory since 2018 under the advisement of Dr. Sung Kyu Lim. His research focuses on devising machine learning (ML) and graph algorithms to improve physical design flows for 2D and 3D integrated circuits (ICs).