Graph Neural Networks for congestion prediction and logic optimization for synthesis of system-on-chips.

Lalit Prasad Peri, Benjamin Rice, Wilson Smith

{lalitprasad, bsrice33 ,wsmit22}@vt.edu
Course Project Report. ECE5424 Fall2024

Abstract. *As transistor scaling continues, the ability to accurately predict congestion during early design stages, particularly during logic synthesis, is essential for reducing the design cycle and enhancing System-on-Chip (SoC) performance. Congestion caused by inefficient logic combinations can lead to suboptimal physical implementations, increased development costs, and longer design times. Traditional manual optimization techniques in Very-Large-Scale Integration (VLSI) design rely heavily on expert intuition and iterative methods, which are both time-consuming and unsuitable for the complexity of modern SoC designs.*

In this project, we explore the application of Graph Neural Networks (GNNs) for logic optimization in SoC design. GNNs offer significant advantages by embedding prior knowledge and incorporating inductive biases relevant to VLSI tasks, making them well-suited for physics-informed optimization. These biases differ from those commonly applied in GNNs used for other structured data types, such as social or citation networks.

Our approach involves predicting congestion through direct learning of embeddings from netlist designs using matrix factorization techniques and inductive biasing, a distinct improvement over classical optimization methods. By integrating these predictions, we aim to improve key design tasks, such as floor-plan congestion, timing, and parasitic estimation, earlier in the design process. Our framework builds upon design automation tools like OPENROAD[1] and DREAMPLACE[2], and seeks to automate critical stages of SoC design, ultimately reducing design cycles and optimizing logic synthesis for improved performance.

1. Introduction

Design and logic optimization in VLSI domain is largely manual and iterative. As the chip size explodes these methods fails to catchup. GNNs have been applied to assist designer with scaling problem like in OPENROAD and DREAMPLACE. These works have proven to improve design times by over 4x and have predicted congestion with over 90% accuracy.

Recently, graph neural networks (GNNs), a subset of machine learning models designed to operate on graph-structured data, have garnered significant attention within the research community. This growing interest suggests that GNNs have the potential to achieve similar success in electronic design automation (EDA), particularly within the very-large-scale integration (VLSI) design automation domain. Many VLSI design data structures can be naturally represented as graphs. For instance, a circuit netlist can be modeled as a graph where devices are nodes and nets serve as edges. However, conventional GNNs, which are typically developed for other problem domains, are not directly applicable to challenges in VLSI design automation. This limitation arises because most GNN methods are designed to handle data generated by natural phenomena, such as biochemical graphs, whereas VLSI data are products of deliberate engineering processes. This distinction aligns with the concept of machine learning for scientific domains, such as physics-informed machine learning.

Physics-informed machine learning aims to apply machine learning techniques to model and predict the dynamics of multiphysics and multiscale systems, such as solving partial differential equations (PDEs). A recent survey on physics-informed machine learning [6] identifies three fundamental principles guiding this approach: (1) introducing observational biases directly through data that encapsulate the underlying physics; (2) incorporating inductive biases that reflect prior assumptions through tailored modifications to an ML model's architecture; and (3) employing suitable loss functions, constraints, and inference algorithms to serve as learning biases.

We propose that these principles are equally applicable to machine learning challenges in the VLSI design automation domain. Among these, we posit that the principle of inductive bias is particularly crucial for the successful application of GNNs in this field. Inductive biases embody the assumptions about the VLSI design automation problem, enabling learning algorithms to prioritize certain solutions over others and enhance generalization beyond the training data.

To effectively leverage this principle for graph learning challenges in the VLSI design automation domain, it is essential to carefully analyze the inductive biases inherent to the problem and design both the graph representations and the GNN architectures accordingly. The resulting graphs may not precisely mirror the structure of circuit graphs, and the architecture might differ from standard GNN architectures. In this paper, we first provide a concise overview of GNNs and the concept of inductive bias. Subsequently, we demonstrate, through various examples, how inductive bias can be utilized in different VLSI design analysis and optimization tasks.

2. Project Objectives & Research Contributions

The anticipated outcomes of this project are as follows:

- **Evaluation GNNs for predicting routing congestion** by quantifying the reduction in design parasitics and timing violations compared to classical electronic design automation (EDA) methods.
- **Optimization in logic synthesis**, specifically in terms of gate-count minimization and area reduction (mm^2) through the application of GNN-based prediction models.
- **Assessment of performance improvements** by measuring the reduction in netlist synthesis time, thereby contributing to a shorter overall chip design cycle.
- **Expansion of the dataset** to include custom netlists that reflect contemporary system-on-chip (SoC) designs, thus enhancing the generalizability of the proposed framework.

3. Background

3.1. EDA flow for System-on-Chip design

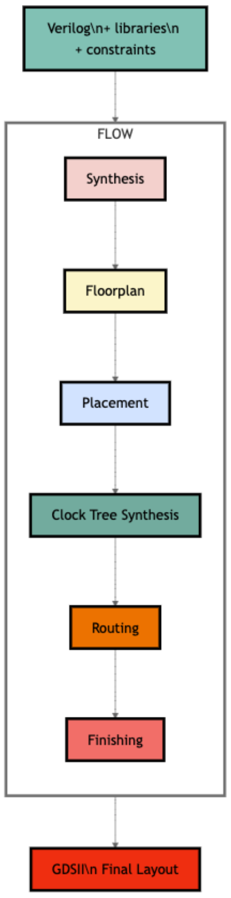


Figure 1. EDA-Flow to design System-on-Chip. Source[1]

The System-on-Chip (SoC) design methodology is a multi-phase process that converts a high-level hardware description into a silicon image. It begins with **RTL synthesis**, where a high-level description written in hardware description languages (HDLs) like Verilog or VHDL is translated into a gate-level netlist, optimized for constraints such as timing, area, and power.

This is followed by **design verification**, which ensures the synthesized netlist meets functional and timing requirements through methods like static timing analysis, functional simulation, and formal verification. The goal is to validate that the synthesized netlist aligns with the intended design.

Next is **physical design (place and route)**, where the gate-level netlist is converted into a physical layout. This involves floor planning, placement of standard cells, clock tree synthesis, and routing of interconnections, while ensuring compliance with design rules through Design Rule Checks (DRC) and Layout vs. Schematic (LVS) verification.

The final phase, **GDSII generation**, includes power analysis, timing closure, and parasitic extraction to ensure the layout's accuracy and performance. The design undergoes final checks before being translated into a GDSII file, the standard format used for chip manufacturing. This comprehensive process ensures the chip meets its specified performance, power, and area targets before fabrication.

These design steps are sequential and represented flow diagram in figure.1.

3.2. Issue of Routing Congestion in SoCs

Routing congestion in System-on-Chip (SoC) design occurs when the demand for routing resources, such as metal layers and wiring paths, exceeds the available capacity in certain areas of the chip. These congestions are driven by physics limitaions of nano-metric scaling of transistors and wires which could lead to critical issues affecting SoC performance .eg.

- **Increased Wire Delays**: High routing congestion can force signals to take longer or less direct paths, increasing the length of wires. This can introduce greater signal delays, making it difficult to meet the desired timing requirements of the design, especially for high-frequency circuits. Consequently, critical paths may fail to meet setup and hold times, leading to timing violations.
- **Higher Power Consumption**: Longer routing paths and increased capacitance due to congested areas can result in higher dynamic and leakage power consumption. The additional capacitance from longer wires requires more energy to drive signals across the chip, which can contribute to overall power inefficiencies in the SoC.
- **Signal Integrity Issues**: Dense routing can exacerbate problems like crosstalk and electromagnetic interference (EMI). Closely packed wires can induce unwanted interference between neighboring signals, leading to glitches or noise in the signals. This can cause functional errors or the need for additional buffers, which further complicate timing and area requirements.
- **Area Expansion**: To mitigate congestion, designers may be forced to enlarge the chip area, creating more space for routing channels. While this can alleviate congestion, it increases the silicon die size, leading to higher manufacturing costs and potentially lower yield due to a larger chip footprint.

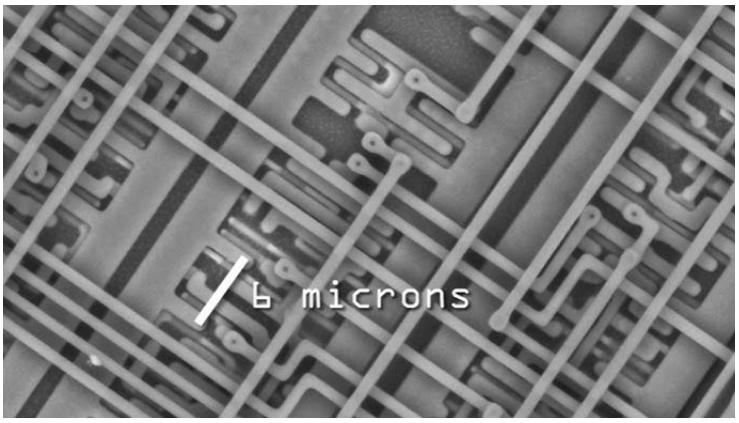


Figure 2. System-on-chip wire routing under a microscope. Source[1]

3.3. GNN

In this section, we will provide a concise discussion of the fundamental principles of graph neural networks (GNNs) and their applications in various machine learning tasks. Our aim is not to establish a detailed taxonomy or offer an extensive overview of GNNs; readers seeking a comprehensive understanding can refer to several in-depth surveys, such as [3]. Traditionally, when dealing with structured data represented as graphs, data scientists have needed to manually encode features to address prediction tasks. However, this process is often complex, involving resource-intensive queries, and is prone to errors. Manual feature engineering tends to be suboptimal and requires significant effort.

Machine learning techniques have the potential to automate feature extraction, but conventional methods are typically limited to operating on sequence or grid-structured data. The core principle of GNNs is to learn graph features, or representations, by mapping nodes to d-dimensional embeddings in such a way that similar nodes within the network are placed close to each other in the embedding space. GNNs can be trained directly using supervised learning, with predicted targets typically focusing on properties at the node level, link level, or entire graph level. Additionally, GNNs can utilize self-supervised learning to derive embeddings for downstream predictive tasks. They can also be integrated into reinforcement learning frameworks to optimize specific reward functions.

A recent survey on GNNs [3] identifies key computational modules integral to their operation: Propagation Module, which facilitates information exchange between nodes, allowing the aggregated information to capture both feature and topological characteristics; Sampling Module, which selects neighboring nodes when each node has a large number of connections; and the Pooling Module, which aggregates information at the subgraph or entire graph level from individual nodes.

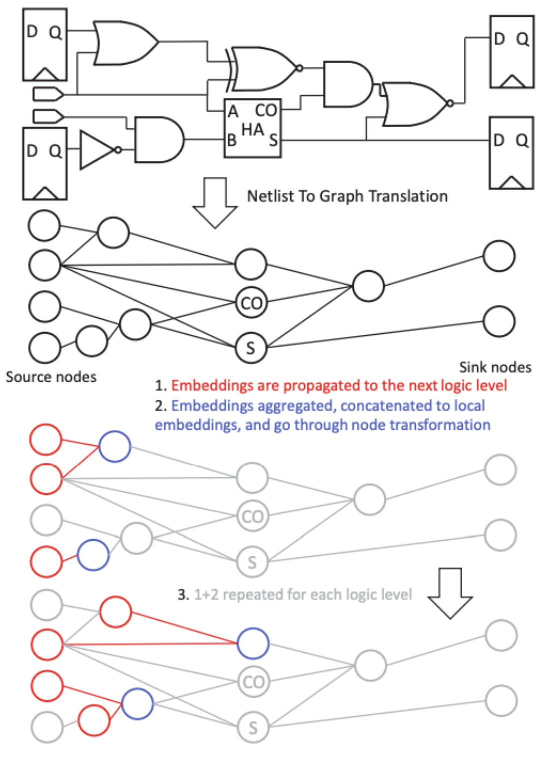

D Q
A CO
HA
B S
Netlist To Graph Translation
CO
S
Source nodes
Sink nodes
1. Embeddings are propagated to the next logic level
2. Embeddings aggregated, concatenated to local embeddings, and go through node transformation
3. 1+2 repeated for each logic level

Figure 3. The netlist is translated from Verilog to a graph representation. The sub-graphs then used in OpenRoad-GNN model. Source[8]

3.4. Inductive Bias

Neural networks have the capacity to approximate any function; however, the curse of dimensionality necessitates a substantial amount of data for effective training. When working with a finite training set, it becomes essential to introduce certain assumptions about the solution space to achieve generalization to new input data. These assumptions are referred to as *inductive biases* [10], which guide the learning algorithm to prioritize solutions with specific properties. For instance, convolutional neural networks (CNNs) utilize two key inductive biases for computer vision tasks: spatial translation and composition. Spatial translation equivariance implies that a spatial shift in the image does not alter the function's output, while composition suggests that complex functions can be constructed from simpler, more basic functions.

Graph neural networks (GNNs) employ a critical inductive bias: equivariance over nodes and edges. This means that similar edges between similar nodes should produce similar function values. These functions are often *permutation invariant*, indicating that the ordering of neighboring nodes does not affect the function's output. To encode inductive biases into a machine learning model, it is common to impose architectural constraints. For example, the convolution operator used in CNNs inherently embodies the property of translation invariance, while the deep layered structure of modern CNNs adheres to the principle of composition. In the case of GNNs, the computational modules discussed earlier must satisfy the property of permutation invariance to properly reflect the inductive biases.

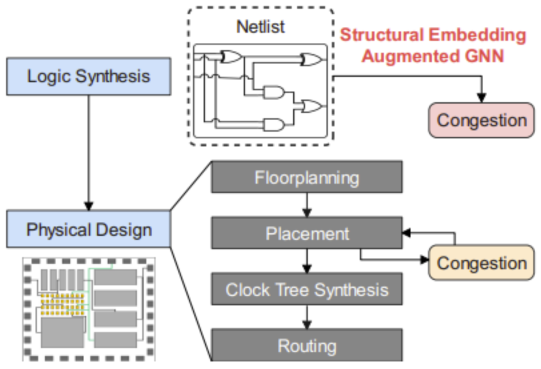


Figure 4. Congestion Prediction at different stages for Chip synthesis. Source[14]

3.5. GNN for Design Analysis & Optimization

GNN can help speedup design analysis or predict the analysis of future steps; it can also be used to produce configuration and guidance for future steps.

In the electronic design automation (EDA) workflow, the Register Transfer Level (RTL) design, described using hardware description languages such as VHDL or Verilog, is transformed into a physical layout suitable for manufacturing through processes like logic synthesis and physical design (Figure 2). During the physical design stage, circuit elements are positioned on the circuit board, followed by the routing process. While global routing results can provide an estimation of routing congestion, it is particularly valuable to identify

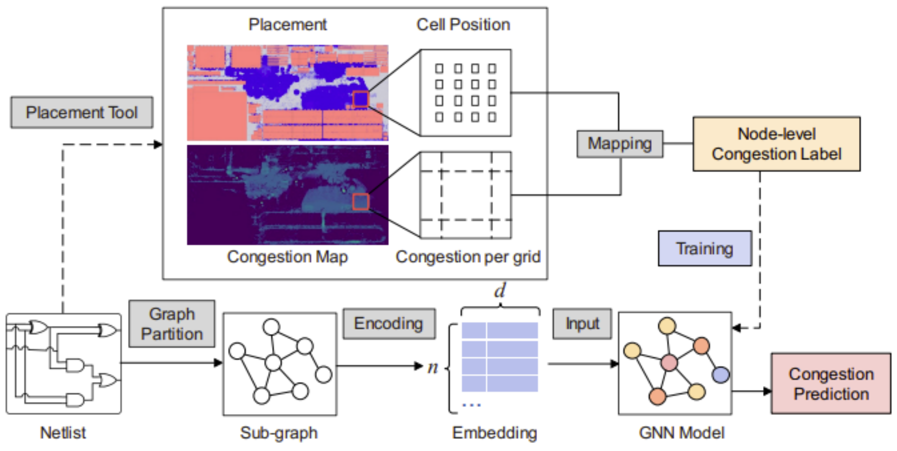


Figure 5. Flow-Diagram for netlist-to-graphs. Training Steps(dashed-arrow) and inference (solid-arrows).Source[4]

areas of high congestion early in the design process. This early awareness facilitates rapid feedback and helps to shorten design cycles.

To circumvent the high computational costs associated with placement, it is advantageous to predict congestion during the logic synthesis phase. Several studies have attempted to identify network structures within a synthesized netlist that may indicate localized areas of high routing congestion [7].

A recent approach, termed *CongestionNet*, employs a Graph Neural Network (GNN) to predict congestion based on a synthesized netlist [8]. We first introduce the concept of similarity in the context of embedding learning on graphs.

The simplest notion of similarity between two vertices in a graph is based on their proximity. Neighboring vertices are considered highly similar, with the similarity diminishing as the number of edges required to traverse between vertices increases (see Figure 1). Beyond this, one can consider *structural similarity*, which relates to intrinsic properties of a node, such as its degree and spectral properties. Notably, two nodes may exhibit structural similarity even if they are part of different graphs. In our approach, we predict the probability of congestion by evaluating the likelihood of overlap between subgraphs during the layout phase of these sub-netlists.

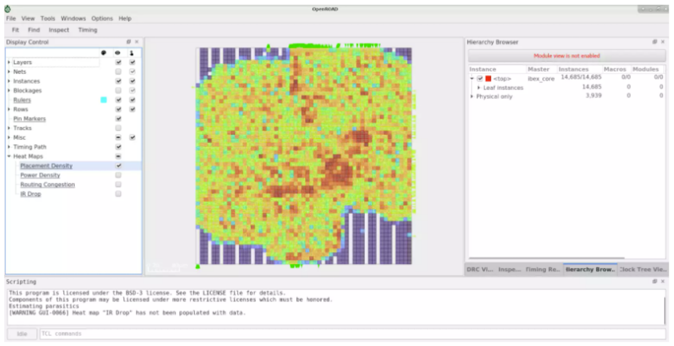
In addition to predictive tasks in VLSI design, we propose utilizing these prediction results to guide design optimization processes. By iteratively applying design constraints and adjusting inductive biases for subgraphs, it is possible to produce a netlist that is optimized for factors such as area and gate count, while also being free of routing congestion issues. A flow diagram illustrating the process of iterative training and prediction is shown in Figure 3.

4. Experiment and evaluation method

4.1. Dataset for training

As a first step, we frame the congestion prediction problem as a problem solvable by GNNs — the node regression problem, where continuous-valued labels are provided on some nodes (the training set) and GNNs predict this label on other nodes (the test set) with a held-out node set (validation set).

We extract two publicly available netlist sets: Superblue circuit line which we place via DREAMPLACE [2] as well as a



```
This program is licensed under the BSD-3 license. See the LICENSE file for details.
Components of this program may be licensed under more restrictive licenses which must be honored.
Estimating parasitics
[WARNING GUI-0066] Heat map "IR Drop" has not been populated with data.
```

Figure 6. Chip Floorplan and Congestion Prediction Visualization with OpenROAD:EDA.Source[1]

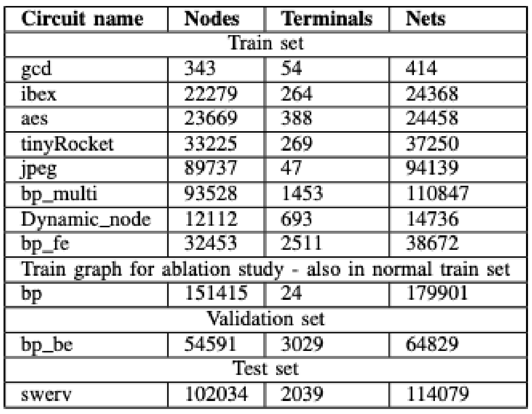
collection of circuits provided with the OPENROAD framework [1].

- During the global routing phase, we save the position of a cell during an iteration along with the 2-D congestion label and convert the grid congestion value to cell label.
- The dataset is then divided into test and train graphs. We train a GNN to create accurate predictions (in correlation terms) for the test graphs' congestion.

4.2. Error and Accuracy

We begin with a publicly available, calibrated training set characterized by known levels of error and accuracy. By expanding the set of netlists and graphs (see Figure 5) to include a broader collection of VLSI designs, we aim to iteratively benchmark the accuracy of predictions made by the OpenRoad-GNN model.

To achieve this, we plan to incorporate netlist samples such as the RISC-V Boom core [22], which includes a large collection of netlists, with samples numbering on the order of 10^6. These netlists are known for their optimized structure and congestion-free characteristics. Integrating these samples into the training dataset serves as a control measure for mitigating inefficiencies in prediction errors, which may arise from issues such as sub-optimal graph conversion, subgraph creation, or the application of biased constraints.



Circuit name	Nodes	Terminals	Nets
Train set			
gcd	343	54	414
ibex	22279	264	24368
aes	23669	388	24458
tinyRocket	33225	269	37250
jpeg	89737	47	94139
bp_multi	93528	1453	110847
Dynamic_node	12112	693	14736
bp_fe	32453	2511	38672
Train graph for ablation study - also in normal train set			
bp	151415	24	179901
Validation set			
bp_be	54591	3029	64829
Test set			
swerv	102034	2039	114079

Figure 7. OpenROAD[1] Public dataset for collection of netlists. We plan to use this dataset to quantify Error and Accuracy of GNN predictions.

Milestone	ETA
Milestone 1	October 18th
Milestone 2	November 1st
Milestone 3	November 15th
Milestone 4	November 29th
Final Report	December 6th

Table 1. Project Timelines and Milestones

5. Project Timelines

Here is the project plan and tentative list of milestones we propose to achieve project objectives:

1. **Milestone #1, October 18th:** Complete the collection and preparation of datasets, and ensure all netlists are converted into graph representations.
2. **Milestone #2, November 1st:** Finish the design and implementation of the GNN model and prepare it for training.
3. **Milestone #3, November 15th:** Complete the evaluation of the GNN model by testing its predictions on the datasets, and calibrating the accuracy using OpenROAD.
4. **Milestone #4, November 29th:** Finish analyzing prediction results, examining both successful predictions and failures and calibrate the predictions to improve accuracy.
5. **Final Report December 6th:** Finish writing the final paper that reports methods and findings based on results.

6. Conclusion

In conclusion, in this project we intend to demonstrate the potential of Graph Neural Networks (GNNs) to predict routing congestion early in the design process of complex chips, offering a significant advantage over traditional methods. By leveraging these predictions, we aim to iteratively optimize the high-level synthesis of netlists, facilitating more efficient implementation of System-on-Chip (SoC) designs. The integration of GNN-based predictions into the design flow promises to reduce design cycle time and enhance the overall performance and manufacturability of SoCs.

7. References

1. OpenRoads Project https://theopenroadproject.org/
2. Y. Lin et al., "DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 40, no. 4, pp. 748-761, April 2021
3. Guyue Huang. et al., "Machine Learning for Electronic Design Automation: A Survey". ACM Transactions on Design Automation of Electronic Systems, 2021.
4. Brucek Khailany, Haoxing Ren, Steve Dai, Saad Godil, Ben Keller, Robert Kirby, Alicia Klinefelter, Rangharajan Venkatesan, Yanqing Zhang, Bryan Catanzaro, William J. Dally."Accelerating Chip Design With Machine Learning". IEEE Micro, 2020. paper
5. Yuzhe Ma, Zhuolun He, Wei Li, Lu Zhang, Bei Yu., "Understanding Graphs in EDA: From Shallow to Deep Learning" International Symposium on Physical Design (ISPD), 2020. paper
6. Daniela Sanchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Souvik Hazra, Robert Wille, Wolfgang Ecker. "Survey of Graph Neural Networks for Electronic Design Automation" ACM/IEEE Workshop on Machine Learning for CAD (MLCAD), 2021.
7. Daniela Sánchez Lopera, Lorenzo Servadei, Gamze Naz Kiprit, Robert Wille, Wolfgang Ecker. "A Comprehensive Survey on Electronic Design Automation and Graph Neural Networks: Theory and Applications." ACM Transactions on Design Automation of Electronic Systems, 2022.0.
8. Haoxing Ren, Siddhartha Nath, Yanqing Zhang, Hao Chen, Mingjie Liu. "Why are Graph Neural Networks Effective for EDA Problems?". IEEE/ACM International Conference On Computer Aided Design (ICCAD) , 2022.0.
9. Nan Wu, Hang Yang, Yuan Xie, Pan Li, Cong Hao "High-Level Synthesis Performance Prediction using GNNs: Benchmarking, Modeling, and Advancing." ACM/IEEE Design Automation Conference (DAC), 2022.0.
10. Daniela Sánchez Lopera, Wolfgang Ecker. "GNNs for EDA Behavioral and Logic Design Applying GNNs to Timing Estimation at RTL" IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2022.
11. Ecenur Ustun, Chenhui Deng, Debjit Pal, Zhijing Li, Zhiru Zhang "Accurate Operation Delay Prediction for FPGA HLS Using Graph Neural Networks." IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2020.
12. Nan Wu, Yuan Xie, Cong Hao. "IRONMAN: GNN-assisted Design Space Exploration in High-Level Synthesis via Reinforcement Learning." Proceedings of the 2021 Great Lakes Symposium on VLSI (GLSVLSI), 2021.
13. Robert Kirby, Saad Godil, Rajarshi Roy, Bryan Catanzaro. "CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks." IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC), 2019. paper
14. Amur Ghose, Vincent Zhang, Yingxue Zhang, Dong Li, Wulong Liu, Mark Coates. "Generalizable Cross-Graph Embedding for GNN-based Congestion Predic-

tion." IEEE/ACM International Conference On Computer Aided Design (ICCAD), 2021. paper

15. Yi-Chen Lu, Siddhartha Nath, Vishal Khandelwal, Sung Kyu Lim. "RL-Sizer: VLSI Gate Sizing for Timing Optimization using Deep Reinforcement Learning." 58th ACM/IEEE Design Automation Conference (DAC), 2021. paper
16. Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, Jeff Dean. "A graph placement methodology for fast chip design" Nature, 2021.
17. Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, Jiang Hu, Yixiao Duan, Yiran Chen. "Net2: A Graph Attention Network Method Customized for Pre-Placement Net Length Estimation" Proceedings of the 26th Asia and South Pacific Design Automation Conference (ASPDAC), 2021.
18. Yi-Chen Lu, Sai Pentapati, Sung Kyu Lim. "The Law of Attraction: Affinity-Aware Placement Optimization using Graph Neural Networks" Proceedings of the 2021 International Symposium on Physical Design (ISPD), 2021.
19. Anthony Agnesina, Kyungwook Chang, Sung Kyu Lim. "VLSI placement parameter optimization using deep reinforcement learning" Proceedings of the 39th International Conference on Computer-Aided Design (ICCAD), 2020.
20. Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z. Pan, Yibo Lin. "A timing engine inspired graph neural network model for pre-routing slack prediction" Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC), 2022.
21. Ruoyu Cheng, Junchi Yan. "On Joint Learning for Solving Placement and Routing in Chip Design" Neural Information Processing Systems (NeurIPS), 2021.
22. RISC-V MegaBoom CPU [https://github.com/The-OpenROAD-Project/megaboom]