Congestion Prediction in Chip Design with Walks and Partitioning

Walter Wong wcw003@ucsd.edu
Jiesen Zhang jiz147@ucsd.edu
Oren Kaplan okaplan@ucsd.edu
David Moon damoon@ucsd.edu
Lindsey Kostas lkostas@qti.qualcomm.com
Alex Hagen alexhage@qti.qualcomm.com
Arnav Ballani aballani@qti.qualcomm.com

Abstract

Chip design has increased exponentially in terms of complexity, leading to significant challenges in predicting and reducing wire congestion. Congestion is the term used for when there is an excessive amount of wire routing in a constrained physical area, which negatively impacts performance through overheating, signal interference, and increased power consumption. Our methodology proposes enhancements to the existing DE-HNN model by introducing skip connections to improve long-range message passing and leveraging alternative partitioning algorithms. These modifications aim to improve the accuracy of congestion prediction and optimize chip design efficiency. By evaluating these approaches against the original model, this work contributes to advancing analysis and applications of graph machine learning in analog chip design.

Website: https://orenkgit.github.io/capstone-webpage/
Code: https://github.com/waltercywong/Congestion-Prediction-with-Walks-and-Paritioning

1 Introduction

1.1 Analog Chip Design

Silicon chips are the main foundations in the modern electronics. These chips are used in electronics such as smartphones, computers, automobiles, medical devices, and industrial automation systems, which are all necessary to daily life. The semiconductor industry has been developing rapidly due to the increasing computational demands and all that has led to the development of the need for higher performance, power efficiency and miniaturization.

As chip technology advances, the semiconductor design is becoming more and more complicated. The latest chips today have billions of transistors which are then arranged into logic gates, cells and functional blocks. These elements then form circuits that are in charge of processing data, storing information and controlling power consumption. It is key to design such circuits with considerations to power consumption, heat interference, and signal integrity in order to achieve optimal performance and reliability.

1.2 Placement and Routing, Demand and Congestion

In the semiconductor design workflow, Placement and Routing (PnR) is an important stage where logical circuits are mapped onto the physical layout of a chip. During placement, components such as standard cells and macros are positioned on the actual chip. Routing then establishes interconnections between these components to ensure proper signal transmission. The PnR process is iterative, expensive, and computationally intensive, as designers must balance the constraints of timing, power distribution, and congestion.

A netlist consists of cells (functional units like logic gates, flip-flops, and memory blocks) and nets (connections that define signal pathways between these cells). Once synthesized, this netlist serves as the input for the placement and routing (PnR) phase, where components are arranged within a fixed 2D chip area, and interconnections are established through multiple wiring layers. The objective of PnR is to create an efficient layout while optimizing key design parameters such as wirelength, timing, power consumption, and congestion.

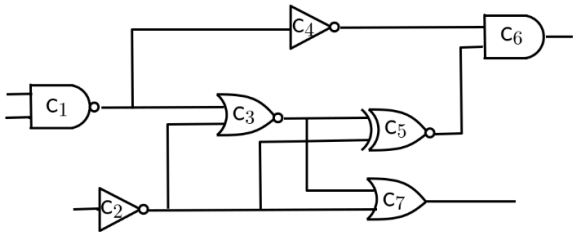

Figure 1: *A Netlist with 7 cells C1…C7 and 5 nets*

Once a netlist is finalized, the designers test multiple iterations of placement and routing to refine the design before mass production. However, due to the complexity of modern chips,

routing congestion can become a significant bottleneck, leading to delays, manufacturing difficulties, and wasted expenses.

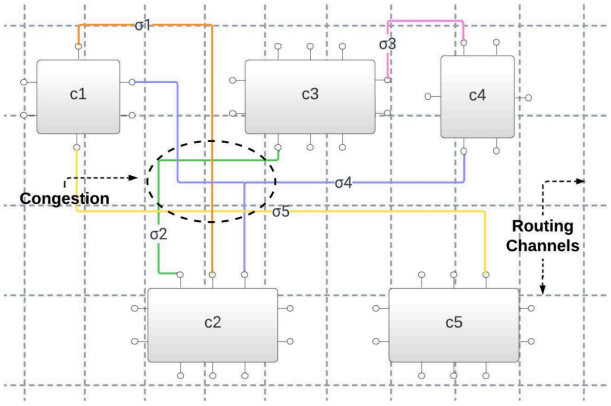

Figure 2: *Congestion in Chip Design*

When talking about congestion, it is referred to when the demand of wires exceeds the routing space available in a specific area of the chip. In short, each chip has a limited physical space for routing across multiple layers. When too many signals need to pass through a limited region, congestion occurs, leading to resource inefficiencies and costly routing detours.

2 Problem Statement

Due to the effects of the bottlenecks, finding congested areas early in the PnR timeline is important for any superconductor business. Traditionally, congestion is acknowledged post-routing, which means design teams may only detect major bottlenecks late in the workflow. At this stage, resolving congestion issues often requires substantial modifications to the circuit layout, increasing development time and cost.

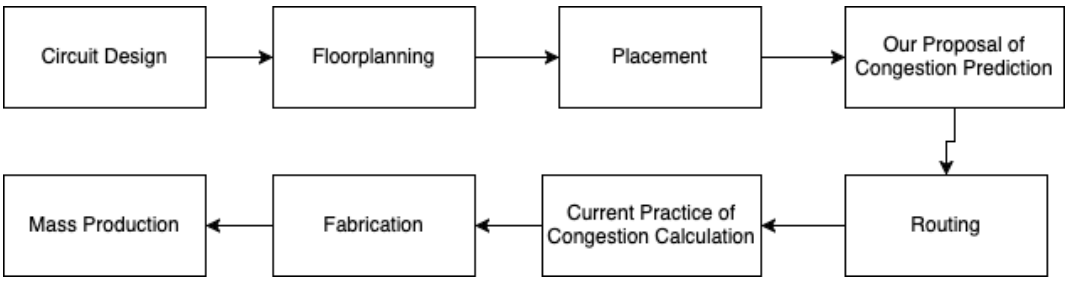

Figure 3: *Flowchart of Chip Design Process*

So, developing predictive methods to anticipate congestion before routing begins can help streamline the design process, reduce unnecessary rework, and improve overall chip efficiency. However, many state-of-the-art (SOTA) machine learning models and algorithms have been primarily developed for social network data, making them less effective for chip

design applications. This limitation arises because, in social networks, neighboring nodes typically represent entities with direct interactions, whereas in chip design, the physical proximity of components does not always correlate with logical connectivity. As a result, existing models struggle to capture the hierarchical and spatial dependencies inherent in netlist graphs.

Therefore, in our project, we aim to enhance potential enhancements to DE-HNN, an existing model to predict congestion, focusing on improving its message-passing mechanisms and refining partitioning strategies to better align with the data of circuit design. By improving these aspects, we aim to develop a more robust model for congestion prediction in modern chip layouts.

3 Data Overview

The data we used for this project was a dataset in collaboration with the University of California, San Diego, and North Carolina State University, with support from Qualcomm. The dataset named Superblue, has 12 unique circuits are one of the most publicly complex circuit data available that has been used in the past for VLSI (Very-large-scale integration) placement and routing.

Each circuit in the dataset is represented as a netlist, a logical structure that can be depicted as a graph. In this graph, instances are represented as nodes, and the connections between these instances (known as nets) are represented as edges. Within each netlist, the size can range from 400,000 to 1.3 million nodes with similar sizes for nets as well.

In addition, within this structure, there are global routing cells served as spatial divisions that organize various regions of the chip. Each global routing cell may contain instances, which are modeled as nodes within the graph. These instances feature multiple terminals, which act as input/output points for nets that connect to terminals on other instances.

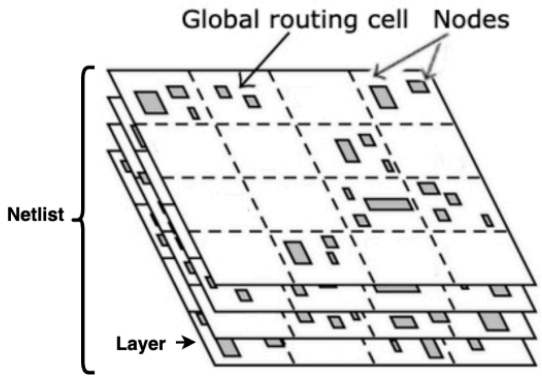

Figure 4: *Example Netlist with Global Routing Cell (GRC)*

4 DE-HNN

The Directional Equivariant Hypergraph Neural Networks (DE-HNN) model, was originally developed by Zhishang Luo and Yusu Wang (Luo et al. (2024)). The DE-HNN model utilizes hypergraph structures to model relationships in circuit designs, offering a better representation than traditional graph-based methods. In addition, by incorporating directional equivariance and virtual nodes, the DE-HNN model aimed to better capture the hierarchical dependencies in netlists and improve congestion prediction in comparison to other SOTA models available today.

4.1 Graph Representation

The DE-HNN model works with netlists as directed hypergraphs, where a net is a hyperedge connecting multiple cells. A net is modeled as a directed hyperedge (c, S), where c is the driver gate and S is the set of sink gates. This representation preserves net topology while distinguishing between driver and sink roles.

4.2 Model Architecture

Graph Neural Networks (GNNs) have been applied to netlists, but they face key challenges:

1. Large Size: Netlists contain hundreds of thousands to millions of nodes and edges.

2. Long-Range Dependencies: Critical properties (such as congestion) depend on distant node interactions.

3. Structural Complexity: Traditional graph representations struglle to encode circuit topology effectively.

Message-passing neural networks (MPNNs) suffer from:

1. Over-smoothing: losing node-level information.

2. Over-squashing: inefficient information propagation over long distances.

3. Limited motif-captureing ability: struggling to encode cycles and trees.

Transformer-based models offer some improvements but has high computational cost and unclear sensitivity to graph topology.

DE-HNN uses an iterative message-passing process to update node and net representations:

- **Node Update:** Each node aggregates information from all incident nets:

$$m^{(\ell)}(v) = \mathrm{Agg}_{\sigma \to v}\left(\left\{M^{(\ell-1)}(\sigma')\right\}_{\sigma' \in I(v)}\right) \tag{1}$$

- **Net Update:** Each net updates its representation based on its *driver and sinks*:

$$M^{(\ell)}(\sigma) = \mathrm{Agg}_{v \to \sigma}\left(m^{(\ell)}(v_\sigma), \left\{m^{(\ell)}(v')\right\}_{v' \in S_\sigma}\right) \tag{2}$$

This direction-aware message passing allows DE-HNN to capture interactions in netlists more effectively than standard GNNs.

To address long-range dependencies, DE-HNN employs a hierarchy of virtual nodes (VNs):

1. **First-Level Virtual Nodes (VNs):** Each VN aggregates information from a subset of nodes.
2. **Super Virtual Node (Super-VN):** Connects all first-level VNs, allowing for efficient global message propagation.

This mechanism extitenables DE-HNN to capture long-range interactions while maintaining a computationally efficient framework.

4.3 Baseline Results

In our baseline experiment, we replicated the setup from the original paper, using Superblue circuits 1, 2, 3, 5, 6, 7, 9, 11, 14, and 16 for training, circuit 18 for validation, and circuit 19 for testing. The focus of our experiment was to model congestion through two distinct loss functions: node demand loss and net demand loss.

- Node demand loss refers to predicting the demand at individual nodes of the circuit. Congestion at a node occurs when the demand at that specific node exceeds its available capacity.
- Net demand loss involves predicting the total demand across a net, which may consist of multiple nodes or cells. Congestion in this case arises when the cumulative demand across the net exceeds the overall capacity. This model takes into account the broader demand distribution over the net as a whole, rather than focusing on individual node congestion.

As shown in the graph, for node demand loss, the model demonstrates learning during the first 7 epochs, after which the RMSE of the test set stabilizes around 10. Similarly, for net demand loss, the model shows initial learning but reaches a plateau in performance, with the RMSE stabilizing at around 6 after approximately 10 epochs. In our work, we aimed to further decrease the loss on both of these types of demand.

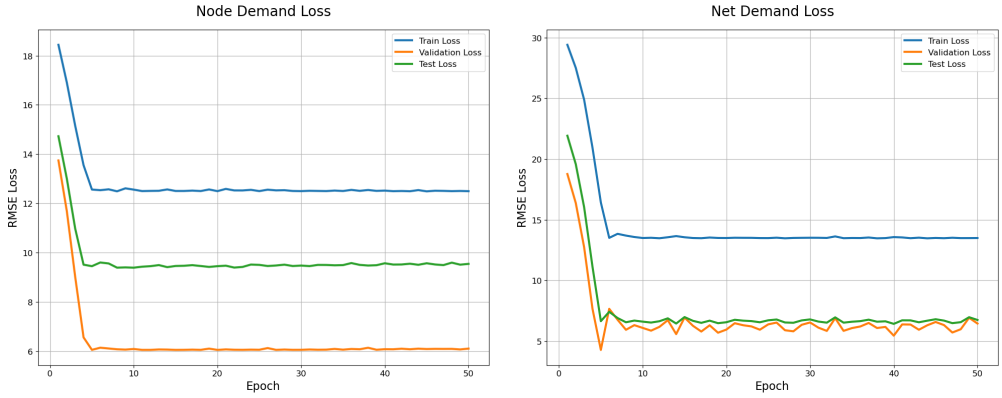

Figure 5: *Baseline DE-HNN Performance*

5 Walks

Walks in graphs, or networks, involve starting at a source node, traversing to one of its neighbors via a connection, or edge, and repeating this process. The goals can range from discovering neighborhood structures, embedding node similarity, or finding shortest paths between nodes. Random walks is a form of walk that involves choosing which node to traverse to at each iteration, randomly. Although this seems like a fruitless algorithm, random walks have shown to naturally find meaning within networks, whether this is with neighborhood structures or node embeddings. Our intuition is to use random walks to find nodes that are topologically far but still have a meaningful physical effect on each other in regards to congestion and demand on the chip.

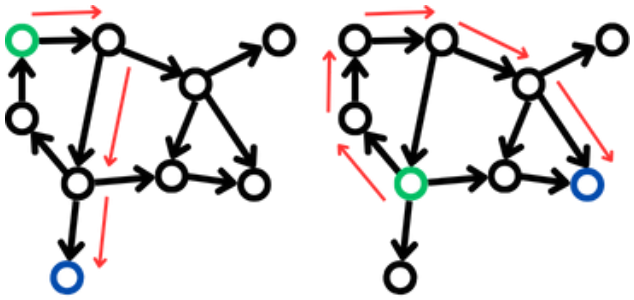

Figure 6: *Random Walk Algorithm with Different Sources*

5.1 Valid Node Pair Dataset

The main hallmark of message passing neural networks (MPNNs), or commonly just graph neural networks (GNNs), is that layers within the model architecture, regardless of whether they are graph convolutional layers or simply multi-perceptron message passing layers, represent the surrounding neighborhood of each node. A single layer represents the nodes one edge, or hop, away from the source. The message being passed is often the node features aggregated with a custom function. If a model has multiple layers, then the source node is receiving messages from nodes that are further away, with the breadth depending on the exact number of layers. For example, if a GNN has 4 layers, then a node will aggregate messages up to 4 hops away. While traditional neural networks can have hundreds of layers, GNNs are typically limited to a few layers. This is due to the fact that too many layers will result in nodes receiving too many messages, squashing the messages that are actually meaningful and resulting in the overall graph to be over-smooth. This is evident by the DE-HNN model, from which we are building our experiments upon, only utilizing 3 layers.

The tradeoff that comes with so few layers is that topologically far nodes do not get their messages passed to each other. This might be fine in more traditional graphs like social networks, where far away nodes are usually people one does not know and therefore do not have important messages to pass. However, in analog chip graph representations, topologically far nodes can have a great effect on each other in the actual physical chip. They

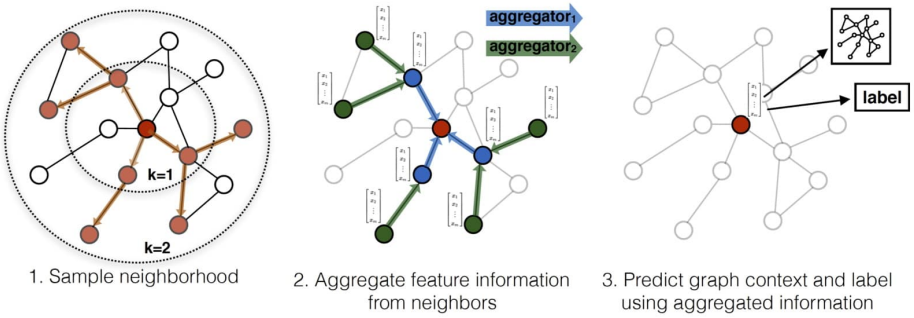

Figure 7: *General GNN Architecture and Message Passing*

can possibly be right next to each other physically, or be part of the same logical or combinatorial process that spans many nodes and nets/edges. The DE-HNN model attempts to address this by partitioning neighborhoods to virtual nodes that message-pass with each other on a higher level. However, we believe that there is not enough meaningful message passing through this method, as messages still get smoothed out through the aggregation methods and multiple layers of separation from going up to the virtual node level and coming back down to the original graph level. This sets up our problem statement and goal with walks, identifying these neglected node pairs and facilitating more direct message passing between them.

To begin tackling this problem, we need a way to properly define whether two nodes have an effect on each other or to the same congested area. We decide that physical location is the most direct indication of this, as nodes in the same physical space will likely contribute to the same power bottlenecks and possibly have crosstalk and signal interference between each other. We construct a dataset of node pairs that are topologically far, defined by being at least 5 hops away from each other and, with the help of physical placement data, are physically close, which we define as being within 0.001 normalized distance units from each other. We define these pairs of nodes as valid pairs and this dataset as the valid pair dataset. The process of constructing this dataset involves choosing random source nodes and traversing from them. Once topologically far away enough, we start checking whether the current node is physically close enough. This resulting dataset comprises approximately 137,000 observations across 10 designs, with each being a source and destination node that meets the above requirements. This dataset is the foundation for which we do the bulk of our data analysis, find feature correlation and fine-tune our experiments.

Valid Pair Dataset					
Source	Destination	Path	Source Feature 1	...	Destination Feature 45
353995	354640	[...]	200.0	...	2.863574e-05
380021	597064	[...]	200.0	...	2.863574e-05
216485	216756	[...]	64.0	...	-7.438903e-07

Table 1: *Portion of Valid Pair Dataset*

5.2 Random Walks

In this baseline experiment, we explore the effectiveness of purely random walks at extracting meaning and related node pairs through purely random traversals of the graph. We sample hundreds of thousands of random source nodes throughout the different designs and do traversals starting from them without revisiting previous nodes or nets/edges. In a directed hypergraph, we first choose at random one of the hyperedges for which the current node is the source for. Then, we select at random one of the many sink nodes for the chosen hyperedge. After 5 hops, we add a skip connection between the original source node and the current node the random walk ends up at. These skip dummy connections are initialized with random numerical features, similar to how the original hyperedge feature tensors are generated in the DE-HNN architecture. There is little risk with this practice as throughout the training epochs, the model learns the features of the net from in and out flow nodes, shifting it significantly from the random feature tensor it started out as. With this resulting modified graph representation, we pass it through the original DE-HNN model architecture.

5.3 XGBoost Proximity Validation

Some problems we observe in the baseline random walks experiment is that limiting the traversals to 5 hops prevents many significant pairs from being reached, not solving our problem with long-range dependencies. In the valid pairs dataset, some nodes are over 100 hops away from each other, highlighting the need for longer, unconstrained traversals. Additionally, simply due to the nature of the first experiment, there was no validation of whether or not the source-destination pairs for which the connections we create have meaningful physical effects on each other.

To solve the first problem, we remove the limit on the number of hops each random walk can make. This allows the traversals to reach nodes that are extremely far down the path of important logical processes in the netlist that have been separated into many cells and nets. To address the problem of validating the significance of source-destination relationships, we decide to build an XGBoost classifier to perform binary classifications on the features of the source-destination nodes to determine whether to add a connection or not. We do not simply use physical placement data like in the valid pairs dataset, because we intend for the model to perform inference on data before placement and routing occurs. This stems back to our motivation of catching possible problem areas early on in the process of chip design, saving time and resources. As a result, we only have access to the netlist and therefore topological connection data and corresponding node features.

To find a suitable proxy for physical proximity, we construct two more large datasets consisting of node pairs. However, these datasets do not have the same qualifiers and constraints as the valid pairs dataset. The first dataset contains simply random pairs within a design. These nodes might not even be connected to each other. The second dataset contains topologically far nodes (5 or more hops away) but without the constraint of having to be physically close. Therefore these node pairs are guaranteed to be connected and not

have their messages passed well in the original model architecture. However, there is no guarantee that they contribute to the same congested area in physical space. For all we know, they could be on opposite ends of the chip.

We proceed to compare the node feature correlations between these different datasets. The results show significantly higher feature correlations for the in-flow persistence diagrams (PD) embeddings in the valid pairs dataset than the other two. Persistence diagrams are embeddings of the local neighborhood of nodes and their structure. The random pairs dataset shows little to no correlation, meaning that it is near impossible to cluster or identify this group of random pairs from other pairs with their features. The second dataset of topologically far nodes had medium correlation, with a Pearson correlation coefficient of around 0.2 to 0.4 for the in-flow PDs. Finally, the valid pair dataset shows whopping correlation coefficients around 0.85 to approaching 1 for the same embeddings.

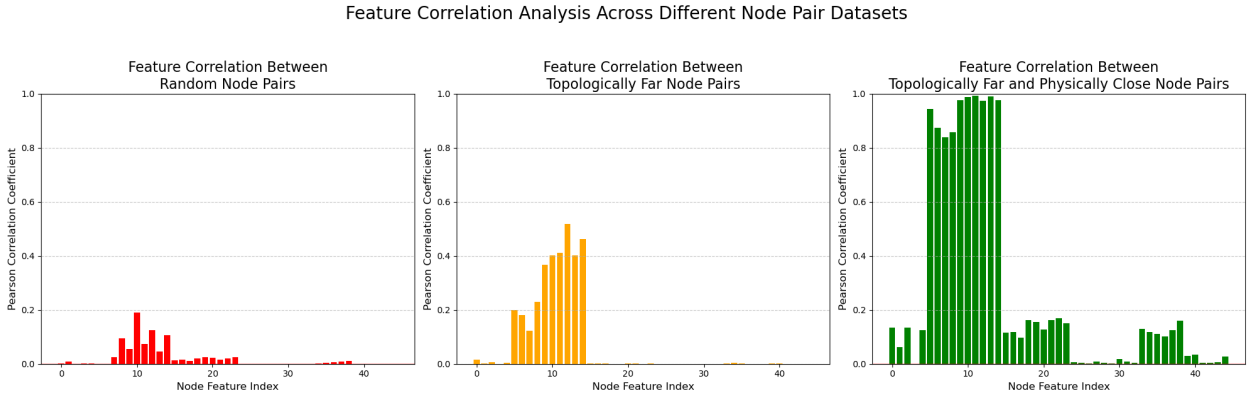

Figure 8: *Node Feature Correlation of Three Node Pair Datasets*

With these significant results, we decide that the in-flow PDs serve as a good alternative, or proxy, for physical proximity from placement data. Therefore, we build our XGBoost classifier from only this set of features, paring down and combining the three node pair dataset for training, validation, and testing. The resulting experiment is an algorithm that samples random source nodes from each graph and performs non-revisiting random walks traversals. After the traversals have gone 5 hops, each current node's in-flow PDs are concatenated with the original source node's and run through the XGBoost classifier. If the classifier returns True, then a skip dummy connection is added; otherwise we keep traversing. Like in the baseline experiment, we input our modified graph representation into the original DE-HNN model architecture.

5.4 Weighted Walks, Early Stopping

Building upon our previous 2 experiments, we implement a weighting algorithm so that the walks are less random and more able to replicate paths that lead to valid pairs. Additionally, there is a slight worry around adding too much noise in the form of too many connections coming from a singular source node. This might cause over-smoothing of the source node's messages and its surrounding neighbors as the nearby neighborhood becomes too dense

and interconnected. The previous experiment also is quite computationally expensive in terms of runtime, as each traversal can go deep through the graph. To address this we implement early stopping to limit the number of connections added per source node. We set this limit to 3 as this is the average degree of most nodes. Therefore, we do not often go beyond doubling the degree of source nodes.

In regards to the weighted walks, the weighting algorithm mainly consists of weighting 2 different factors: overall feature entropy and gradient consistency within valid paths. Feature entropy in a path refers to how much node features change along the path. Looking at the paths within valid paths, we operate with the logic that features that do not change much along these paths must be important towards discovering valid pairs. On the contrary, features that are constantly oscillating or show very random patterns along these paths likely contribute little towards finding patterns in paths between valid source and destination nodes. Gradient strength and consistency refer to the overall patterns and changes of features within these paths. In some way this is a complement to feature entropy, but it more so looks for defined and consistent patterns for features, not just whether features are changing much or not. This involves identifying possible patterns in the gradients of plots of feature evolution, whether they are consistently increasing, decreasing, or oscillating. The overall R-squared correlation of feature evolution over the valid paths validates these patterns in the gradients.

The overall weighting formula for the weighted walks algorithm is approximately 40% of the score towards feature entropy, 30% towards gradient strength, and 30% towards R-squared consistency scores. The final experiment is one that still has the XGBoost model for validation, but now includes a more feature-based, probabilistic algorithm for weighted traversals as well as early stopping to prevent adding too much noise and saving computational power and runtime.

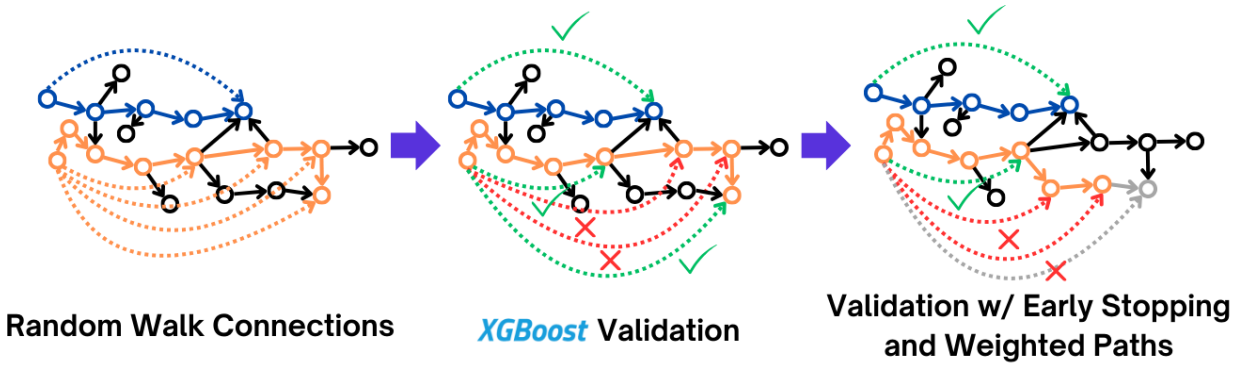

Figure 9: *Experiment Process with Walks*

6 Partitioning

6.1 Louvain Method

The Louvain method is a hierarchical clustering approach that iteratively optimizes modularity, a quality function that measures the density of connections within communities compared to connections between them. We use this approach to identify communities of cells in each chip design, allowing us to identify naturally occurring partitions within the graph, which may correspond to functionally related circuit components or design blocks.

After applying the Louvain algorithm, we observe an average graph modularity of 0.34, indicating a moderate level of community separability within the chip design graphs. While this value suggests the presence of meaningful partitions, it also reflects the complex and interconnected nature of the circuit structures, where clear-cut modular organization may be limited by design constraints and functional dependencies.

We choose to use node partitions obtained from the Louvain algorithm to replace the initial METIS partitions that were previously used. Unlike METIS, which performs multilevel graph partitioning based on edge cuts and balanced partitioning objectives, Louvain provides a data-driven approach to grouping nodes based on their connectivity patterns rather than enforcing predefined balance constraints. This substitution allows us to leverage intrinsic structural properties of the chip design graphs, potentially leading to more semantically meaningful subgraph divisions that align with real-world circuit functionality.

Louvain Community Modularity	
Design	Graph Modularity
1	0.35145
2	0.35422
3	0.35913
5	0.35168
6	0.35123
7	0.35426
9	0.32673
11	0.33364
14	0.33799
16	0.33541
18	0.34653
19	0.33259

Table 2: *Graph Modularity of Each Individual Design in the Dataset*

6.2 Approximating Placement Based Partitions

DE-HNN demonstrated a 10% improvement in Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) when incorporating physical placement data as node features and partitioning criteria. This follows our intuition that physically close nodes will contribute to each other's congestion.

To construct placement-based partitions, the chip was uniformly divided into 81 bounding boxes, with all nodes within a given box assigned to the same partition.

Our objective is to approximate these placement-based partitions by clustering nodes based on features that exhibit a strong correlation with physical placement.

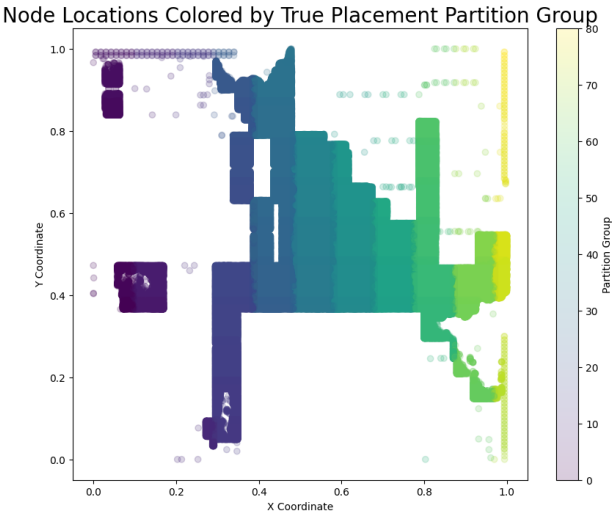

Figure 10: *Nodes Colored by Placement Based Partition*

6.2.1 Feature Analysis

Utilizing placement-based partitions, we conducted a comprehensive analysis of node features within each partition to identify features that exhibit significant correlation with the partitioning scheme.

For each of the 81 partitions, we ranked the importance of individual features based on their correlation with partition membership and their contribution to explained variance. These rankings were then aggregated to determine the overall most influential features in the placement-based partitioning of the given design.

Our analysis revealed that node features corresponding to the top-ten eigenvectors, along with node type, were the most indicative factors in determining the co-partitioning of nodes.

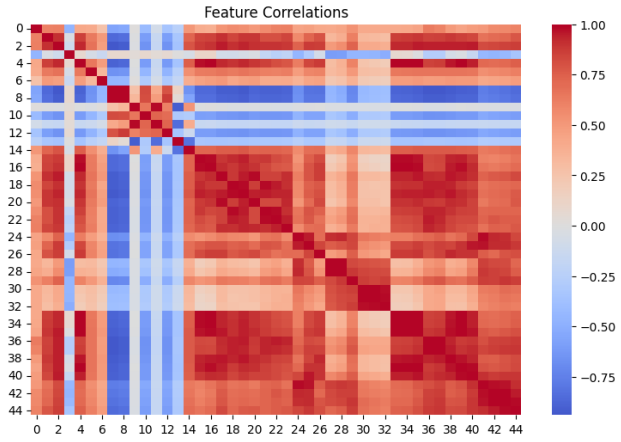

Figure 11: *Example Partition Feature Correlation Matrix*

6.2.2 Standard K-Means

K-Means clustering was selected as an efficient method for grouping nodes based on their feature representations, particularly for large datasets.

For each design, 81 centroids were initialized to correspond to the 81 placement partitions. The K-Means algorithm was then applied, iteratively adjusting the centroids based on their proximity to the nearest nodes until convergence was achieved.

Each node was subsequently assigned a partition ID corresponding to its closest centroid. These partition IDs were then utilized by the model to construct virtual node structures for message passing.

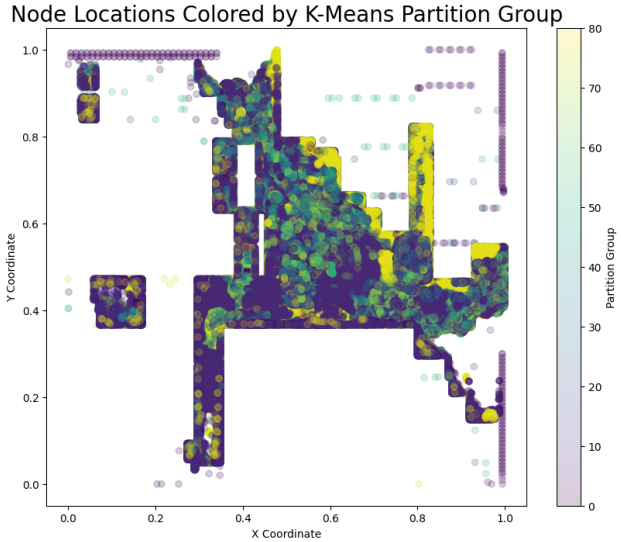

Figure 12: *Nodes Colored by Standard K-Means Partition*

6.2.3 Bisecting K-Means

DE-HNN employs METIS partitioning to ensure that virtual node partitions are balanced in terms of their size, specifically the number of nodes within each partition. To evaluate the impact of partition balancing on model performance, we examine the effect of balancing our K-means partitions.

The bisecting K-means algorithm represents a hybrid approach that combines Hierarchical Clustering with K-means clustering. Rather than initializing K centroids and then converging, the bisecting K-means algorithm sequentially splits a single cluster into two sub-clusters using K-means, until the desired number of clusters is achieved.

This approach results in partitions that are more balanced, aligning closely with the partitioning strategy employed by the base DE-HNN model.

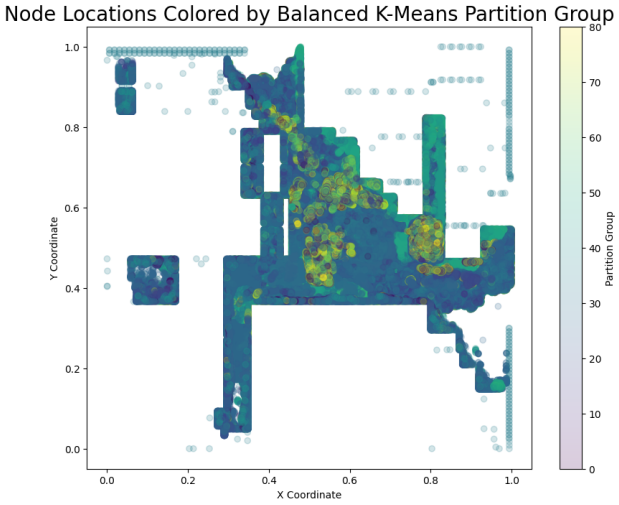

Figure 13: *Nodes Colored by Bisecting K-Means Partition*

7 Results

We evaluate all our experiments against the original DE-HNN model's node and net demand RMSE loss.

7.1 Walks

7.1.1 Random Walks

Evaluating the effectiveness of adding skip connections from the results of purely random walks, the RMSE loss for both node and net demand increased. Node demand increased by a marginal 1.6% while net demand loss increased significantly by 13.5%. These results

point to the fact that random walks do not do well in extracting meaning from graph representations of chip designs, unlike more traditional networks. While this may be due to the specific DE-HNN model architecture, these results are not completely surprising, as completely random walks and skip connections serve as a foundation for our experiments with walks.

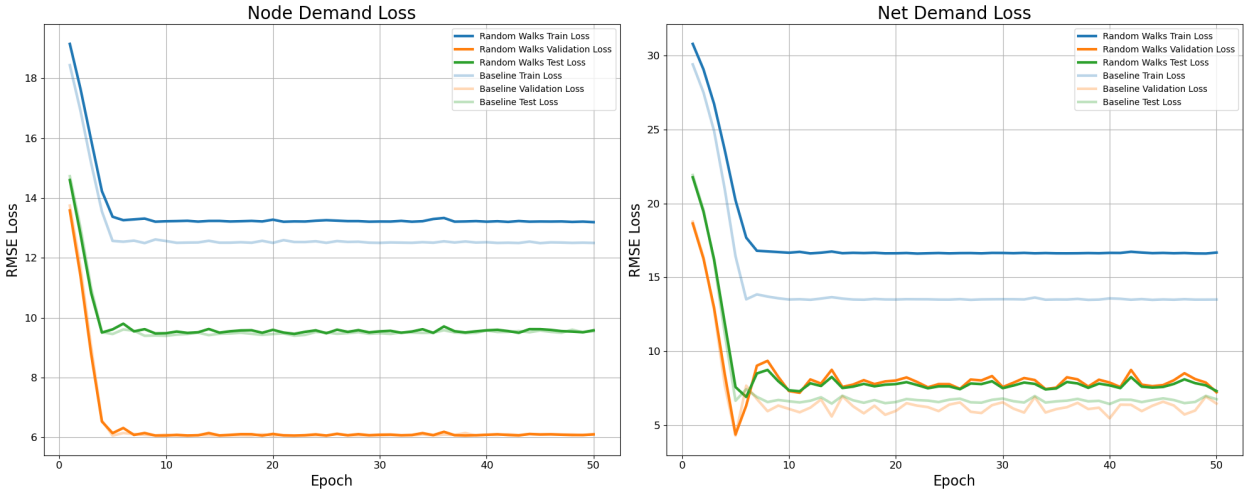

Figure 14: *Random Walk Performance*

7.1.2 XGBoost Proximity Validation

Evaluating the effectiveness of incorporating an XGBoost classifier to validate any skip connections being added along deep traversals, the RMSE loss for node demand decreased by 0.6% while the net demand loss increased by 16.6%. The slight improvement in performance regarding node demand points to the fact we are able to identify some important relationships from topologically far nodes and facilitate their message passing. However, the small number leads us to believe that either we are not adding enough connections or that there do not exist many meaningful pairs to begin with. It is possible that we are not adding enough connections due to our sampling method, which involves taking random source nodes and starting traversals from them. A more comprehensive method such as using a sparse matrix to find the pairwise relationships between every node in a graph and draw connections between all that are positive for the XGBoost classifier. The increase in net demand leads us to believe that the connections we are adding are possibly over-smoothing neighborhoods the edges message-pass from.

7.2 XGBoost Classifier

The XGBoost Classifier performs extremely well on the node pairs dataset, boasting a 93% classification accuracy of whether node pairs are physically close or not, using only the in-flow PDs as features. This performance is partially due to the fact that the dataset is

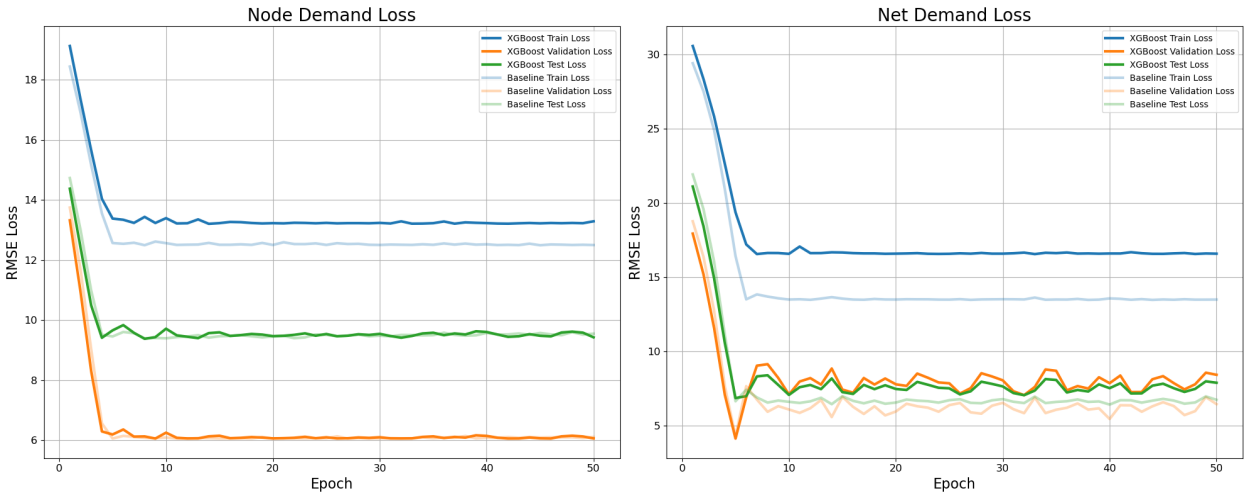

Figure 15: *XGBoost Validator Performance*

extremely skewed towards observations that are not valid pairs, with 90% of the pairs being in this group. However, after oversampling the positive group and fine-tuning the hyper-parameters, the performance only goes up in overall accuracy and significantly improves in regards to classification of valid pairs. The promise with this tree model on in-flow PDs suggests that persistence diagrams can be used in future experiments as a proxy for physical location, opening the gates to innovations before placement and routing.

Classification Report				
Label	**Precision**	**Recall**	**F1-Score**	**Support**
False	0.958	0.960	0.959	252053
True	0.625	0.613	0.619	27932
Accuracy		0.926		
Macro Avg	0.792	0.787	0.789	279445
Weighted Avg	0.925	0.926	0.925	279445

Table 3: *XGBoost Classifier Performance*

7.2.1 Weighted Walks, Early Stopping

Evaluating the effectiveness of adding a weighting algorithm for the walks and early stopping, the RMSE loss for node demand decreased by a marginal 0.2% while net demand increased by 15.7%. With the marginal improvement for predicting node demand, we conclude that adding more connections is a good step forwards, as early stopping actually worsened the performance in regards to node predictions. Additionally, this means that there are no meaningful patterns within the paths between valid pairs. The results give more clarity and assurance towards needing to add more connections to have a greater impact on the graph representation.

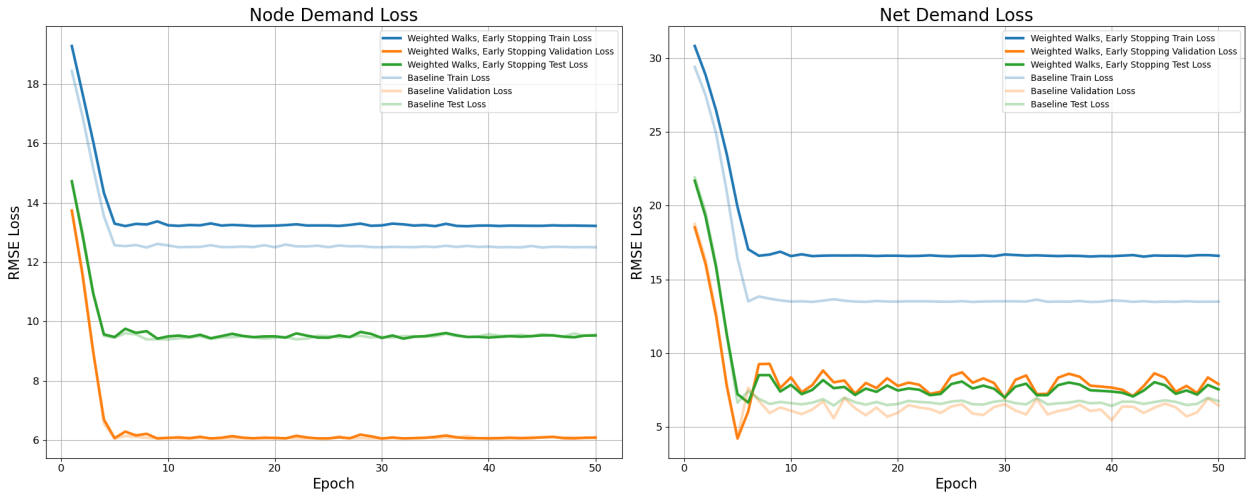

Figure 16: *Weighted Walk, Early Stopping Performance*

7.3 Partitioning

7.3.1 Louvain Method

Evaluating the effectiveness of modularity-based partitioning in chip design graphs, our results indicate a marginal improvement in node loss, with a reduction of 0.25%, suggesting that the revised community-based partitions maintain or slightly enhance the model's ability to predict node-level properties. However, we observe a 3.29% increase in net loss, indicating a slightly diminished performance when considering connectivity-based objectives, such as inter-cluster net connections. This suggests that while the Louvain-derived partitions capture local structural coherence, they may introduce inefficiencies in capturing global connectivity patterns crucial for net-level optimization.

Despite the trade-offs in loss metrics, the modularity-based partitioning approach demonstrates a notable improvement in computational efficiency, significantly reducing runtime compared to METIS-based partitioning. The faster convergence and reduced computational overhead highlight the practical advantages of leveraging unsupervised community detection for partitioning chip design graphs. These findings suggest that while modularity-based partitioning may not universally outperform traditional methods in all predictive tasks, it presents a viable alternative for accelerating large-scale chip design workflows, particularly in scenarios where rapid partitioning is prioritized over slight variations in predictive accuracy.

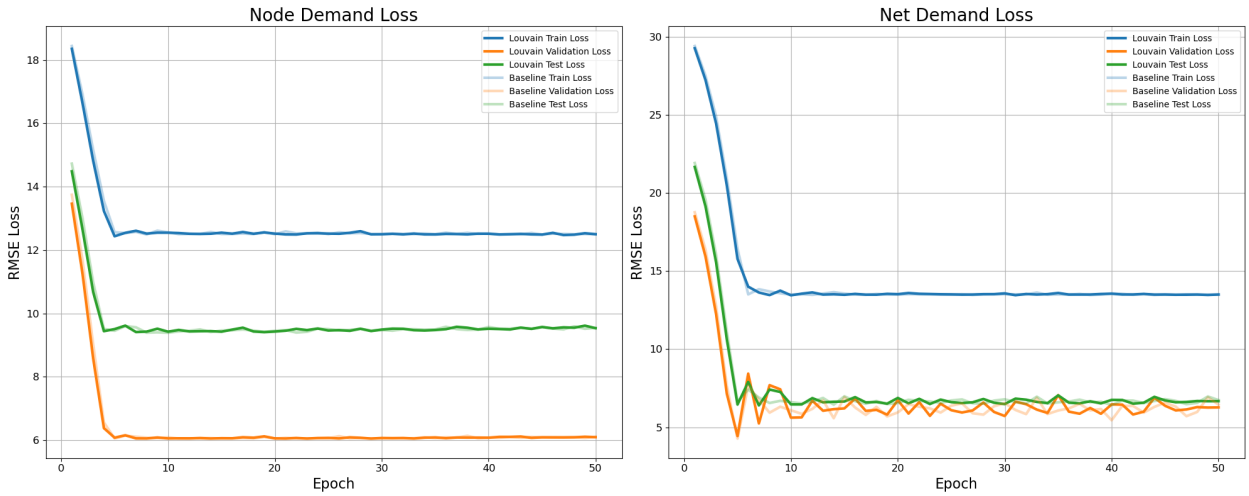

Figure 17: *Louvain Performance*

7.3.2 Standard K-Means

Standard K-Means partitioning demonstrated the most significant performance improvements among all methods in terms of net loss, with only marginal performance degradation observed in node loss. Specifically, net loss improved by 6.7%, while node loss worsened by 1.8%.

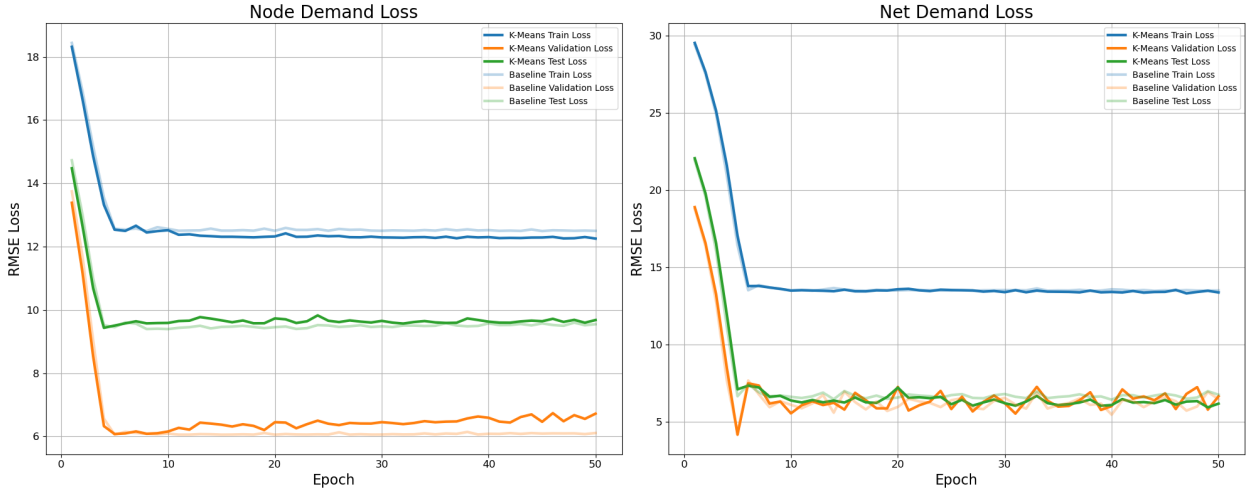

Figure 18: *Standard K-Means Performance*

7.3.3 Bisecting K-Means

Bisecting K-Means provides a compromise between the node and net loss results of standard K-Means clustering. While the net loss improvements exhibit a lower improvement of 4.4%, the reduction in node loss is slightly more favorable, with a decrease of 0.4%.

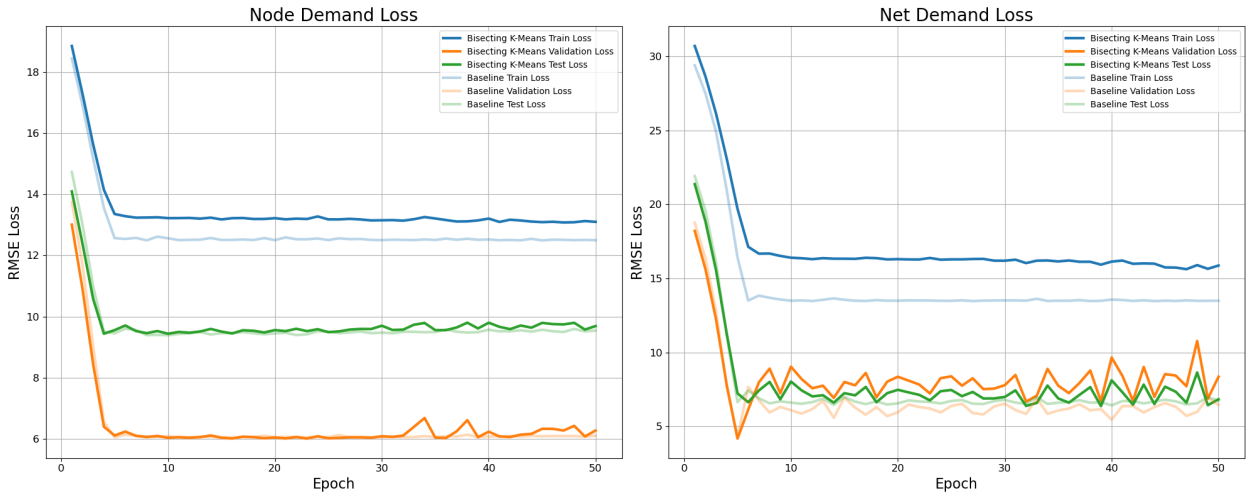

Figure 19: *Bisecting K-Means Performance*

7.4 Overall Results

The greatest improvement in loss for predicting node demand came from random walks with XGBoost validation at a 0.6% performance in loss. The greatest improvement in loss for predicting net demand came from the K-Means partitioning methods, with the standard and bisecting algorithms reducing loss by 6.7% and 4.4%, respectively. The Louvain partitioning method also yields intriguing results, as there is minimal loss in performance while greatly improving the computational runtime of the model.

Model	Node Loss	% Diff	Net Loss	% Diff
Baseline DE-HNN	9.533	---	6.573	---
Random Walks	9.687	1.6%	7.461	13.5%
XGBoost Validation	9.472	-0.6%	7.667	16.6%
Weighted Walk, Early Stopping	9.510	-0.2%	7.605	15.7%
Louvain Community Detection	9.509	-0.3%	6.790	3.3%
K-Means	9.704	1.8%	6.134	-6.7%
Bisecting K-Means	9.569	0.4%	6.285	-4.4%

Table 4: *Overall Loss Results*

8 Conclusion

The experiments we conducted provide many insights into the nature of graph representations and machine learning for analog chip design. From walks we gather that there is improvement in node demand prediction and therefore exist long-range messages that are meaningful towards congestion. Future work will be to add more connections, either through sampling more nodes or the use of pairwise sparse matrices to create and validate more connections. This larger scale experiment will allow the dummy connections to make a more significant impact on the graph representation that should be represented in the

loss metrics. Regarding Louvain community detection, the algorithm shows that modularity based unsupervised clustering and partitioning may not be the blanket solution but offers an alternative topological partitioning algorithm. It shows that there exists some modularity within the graph representation of chips that can possibly be extracted in the future, either to embed more features or to be compressed into one node to cut computational costs. Partitioning with K-Means on the in-flow PDs and yielding performance boosts to net demand prediction bolster the case that these embeddings are a good proxy for physical placement data. But since partitioning operates on larger groups, which is a similar scale to nets, which span out over many sink nodes from their sources, the net demand loss improves while the random walk algorithms cater towards node level predictions as the connections created are directed towards message passing between only nodes. This research project not only provides better understanding of how to analyze graph representations of analog chips, but also sprouts new motivations and angles towards developing new representations and model architectures that may generalize better for different circuits and netlist designs.

References

Luo, Zhishang, Truong Son Hy, Puoya Tabaghi, Michaël Defferrard, Elahe Rezaei, Ryan M Carey, Rhett Davis, Rajeev Jain, and Yusu Wang. 2024. "DE-HNN: An effective neural model for Circuit Netlist representation." In *International Conference on Artificial Intelligence and Statistics*. PMLR

Appendices

A.1 Latest Project Proposal

A.1.1 Background Information

Chip design has become exponentially complex as technological demands grow. This complexity creates bottlenecks in the chip development process, where tasks such as predicting wire congestion and optimizing placement have become critical yet resource-intensive and time consuming. Machine Learning models and algorithms have been demonstrated to be powerful tools for solving these challenges, but their potential in addressing chip congestion remains underexplored. Congestion, which occurs when too many wires are routed through narrow physical areas, leads to performance issues such as overheating, signal crosstalk, and increased power consumption. The ability to predict and mitigate congestion early in the chip design process could revolutionize the field, saving time, resources, and improving chip efficiency.

Successfully tackling chip congestion will impact the semiconductor industry significantly, enabling faster, more cost-effective chip development while improving the performance and reliability of electronic devices. By advancing ML techniques for predicting congestion, we can address a core challenge in chip design, benefiting engineers, researchers, and end consumers.

A.1.2 Prior Work

The DE-HNN (Deep Embedding with Hierarchical Neural Networks) research paper by Zhishang Luo and Yusu Wang (Luo et al. (2024)) attempts to tackle this problem by attempting to learn a graph representation of the chip and predict metrics like congestion and wirelength required from an input netlist without having to place cells and route wires. One of the main features of DE-HNN's representation is the inclusion of virtual nodes which aggregate graph partitions generated by METIS to enhance long-range message passing within the graph. The goal of these virtual nodes is to achieve better long range message passing between nodes, as a common problem in graph machine learning is getting far away nodes to have an effect on each other. However, the current implementation has limitations in accurately capturing long-range dependencies, especially for nodes that indirectly influence congestion but are not well-represented within the existing partitions.

A.1.3 Proposed Enhancements

1. Enhancing Long-Range Message Passing with Dummy Nodes

Modern chips have many interconnected cells within each other, leading to congestion and making long-range message passing important for effective congestion prediction. However, the current DE-HNN framework faces two major issues:

- **Message Over-Squashing:** Interconnected nodes often squash important messages from distant nodes due to the high volume of neighboring connections.
- **Under-Reaching Architecture:** The three-to-four layer architecture of DE-HNN limits its ability to propagate information across longer distances, even with the virtual node hierarchy.

To address these limitations, we propose adding dummy nodes to enhance long-range message passing. Our approach involves:

- Conducting random walks over the input netlist to identify pairs of nodes with frequent indirect connections.
- Adding dummy nodes between nodes that are commonly visited in these walks, creating new 2-hop neighbors.
- Reinforcing critical long-range connections to preserve messages and prevent over-squashing.

This enhancement strengthens the graph representation, enabling the model to capture meaningful relationships between distant nodes and improve congestion prediction accuracy.

2. Domain-Based Partitioning for Virtual Nodes

Secondly, we plan on implementing a more domain-based partitioning algorithm regarding the pool of nodes aggregated by a virtual node. Currently, the DE-HNN paper uses the METIS algorithm for determining these partitions. We were planning on looking into incorporating things like netlist features to achieve a more domain-based partitioning algorithm. We are planning on using deep learning models to learn the most salient netlist features to perform this partitioning to get the most accurately similar nodes under the same virtual node. Being more based in domain knowledge, this approach will address problems with taking neighborhood information, which is the general idea of the METIS algorithm. It will help avoid situations where nodes are clustered close together, a common phenomena in chips nowadays, although they deal with different functional aspects.

Currently we are focused on trying to create partitions based on identifying groups of cells driven by individual clocks. While we are still exploring which features to prioritize, these additions aim to create partitions that better align with the physical design constraints and

logical structure of the chip.

Conclusion

After applying the proposed modifications to the graph representation, we plan to retain the existing DE-HNN model architecture, with potential adjustments to the aggregation mechanism to account for the inclusion of dummy nodes. To evaluate the efficacy of our enhancements, we will:

- Evaluate models with the addition of dummy nodes and the modified partitioning algorithms independently to determine the significance and effectiveness of each modification.
- If both modifications perform well, synthesize them into a single final model for evaluation.

The final model will be benchmarked using the metrics outlined in the original DE-HNN paper, including node congestion classification and total wirelength prediction. We will compare our results against simpler baseline models as well as the original DE-HNN framework to assess improvements in performance.

Following the evaluation, we aim to analyze the broader implications of our proposed modifications across different scopes of chip design. By exploring the impact of our enhancements, we hope to provide insights into their potential applications beyond the specific use case addressed in this project. This comprehensive evaluation will contribute to advancing graph-based representations and machine learning approaches in addressing chip design challenges.

The primary output of our project will be a comprehensive report/paper documenting the methodology, results, and broader implications of our work. The report will include detailed analyses of the data produced by our model, supported by visualizations and statistical comparisons to baseline methods and the original DE-HNN framework.

B Contributions

- **Walter Wong:** Explored methods for possible methods to create skip connections between topologically far nodes for better long range message passing. Created multiple scripts for generating feature datasets and visualizations for analysis regarding node pairings. Implemented scripts for modifying the directed hypergraph representation passed into the DE-HNN model architecture through weighted and completely random walks. Created and managed Github repository structure for better code organization. Implemented scripts for all the random walk experiments and corresponding training scripts. Contributed to bulk of poster text, visual content.

- **Oren Kaplan:** Created scripts for K-Means partitioning and Bisecting K-Means partitioning. Explored additional KaHyPar partitioning. Performed data analysis on all partitioning methods. Additionally explored methods for creating domain based partitions. Created webpage for project showcasing. Created poster outline.
- **David Moon:** Benchmarked and debugged the original DE-HNN model to replicate the original paper's performance. Set up and optimized cloud computing resources, streamlining all experimental implementations to ensure an efficient workflow. Helped with creating and debugging all scripts necessary to run our experiments. Researched benchmarking metrics for evaluating our model and systematically gathered all results for a smooth analysis process. Maintained the majority of the GitHub repository to ensure reproducibility and clarity.
- **Jiesen Zhang:** Performed explorative data analysis on processed data and valid-pairs of node. Created manual script for calculating the graph modularity of different designs and researched state-of-the-art community-detection algorithms. Created and managed Github issues threads for group members to catchup with each other's progress.