Chip Design and Graph Representation

Neil Damle
ndamle@ucsd.edu

Wai Siu Lai
wslai@ucsd.edu

Pranav Rebala
prebala@ucsd.edu

Sahith Cherumandanda
scherumandanda@ucsd.edu

Lindsey Kostas
lkostas@qti.qualcomm.com

Elahe Rezaei
erezaei@qti.qualcomm.com

Masoud Sadeghian
msadeghi@qti.qualcomm.com

Abstract

Chip design optimizations have become increasingly challenging due to rising complexity, resulting in recent research towards machine learning methods and other techniques to improve the design process. A recently proposed model is the DE-HNN model, which uses a directed hyper-graph approach to improve demand and congestion predictions, given only the connectivity of the chip's architecture (Luo et al.). We build upon the results and architecture of this model to identify specific features in a netlist that are highly related to demand and congestion through ablation studies and the implementation of explainable AI tools, such as SHAP (SHapley Additive exPlanations) values. We also explore alternatives to the data processing step, such as downsampling, to improve generalizability. We also add capacity as a feature, explore tree-based modeling, and use hyper-vectors to modify the existing architecture. Finally, we use alternative partitioning approaches to target local structures and improve predictive performance.

Website: https://prebala123.github.io/chip-design/
Code: https://github.com/prebala123/chip-design

1 Introduction

Chips are key components of many applications and tools, from mobile phones to self-driving cars. Chip design involves defining the product requirements for a chip's architecture and system, as well as the physical layout of the chip's individual circuits, a task which is becoming increasingly challenging as these technologies continue to develop. Rising complexity is pushing the limits of existing chip design techniques, and machine learning offers a possible avenue to new progress. One specific area where chip designers face problems is predicting demand; often there are certain areas in a chip through which large amounts of information must pass, creating bottlenecks and reducing efficiency. Although electronic design automation tools have been useful to ensure scalability, reliability, and time to market, our group aims to improve the process by exploring new machine learning techniques.

Given the relatively unexplored domain of machine learning for chip design, many recent attempts have tried to apply tools from different fields, which may not perfectly fit this specific use-case. For example, chip circuits are often represented as graphs in machine learning, and researchers must choose a specific kind of graph and the features within the graph. Our project aims to identify possible shortcomings in current chip representations and suggest possible improvements, which will allow for deeper and more accurate research in the future.

2 Exploration

2.1 Data

We used 12 Superblue circuits from (Viswanathan et al. 2011). These circuits were used in the original DE-HNN paper, and will be used to remain consistent with our baseline model. Each netlist ranges from 400,000 to 1,300,000 nodes and is extremely complex.

For each chip, all cells (nodes) and nets (hyperedge) are associated with feature vectors. Node features contain cell type, dimensions, degree, local connectivity information such as degree distribution, top-10 Laplacian Eigenvectors, and a persistence diagram representation. Then, each net's feature is aggregated from its corresponding pins (eigenvectors and degree).

2.2 Baseline Model

We extended research into the DE-HNN (Directional Equivariant Hypergraph Neural Network) (Luo et al. 2024) model architecture, a framework for learning circuit properties from netlist graphs. The DE-HNN model implemented by Luo et al. is as a message-passing neural network on a directed hypergraph. Importantly, it makes use of injected virtual nodes that simplify the graph structure by forcefully adding connections between topologically nearby nodes. The Superblue dataset depicts the logic gates (nodes) and the wires connecting them (edges), which intuitively leads to a graph representation. Luo et al. represent the designs as hypergraphs, where nodes correspond to circuit cells (logic gates or blocks), and nets are modeled as hyperedges that connect one driver cell to multiple sink cells.

Our work was benchmarked against, and later built upon, a two layer DE-HNN model, with a node dimension of 16 in each layer, and virtual nodes. The dataset for this model used Superblue 18 and 19 as validation/test chips, and the remaining 10 chips served as training data, although we conducted an analysis that showed that most configurations of a train/test split had similar results. This baseline model had validation and test RMSE converging at around 6.4.

2.3 Exploratory Data Analysis

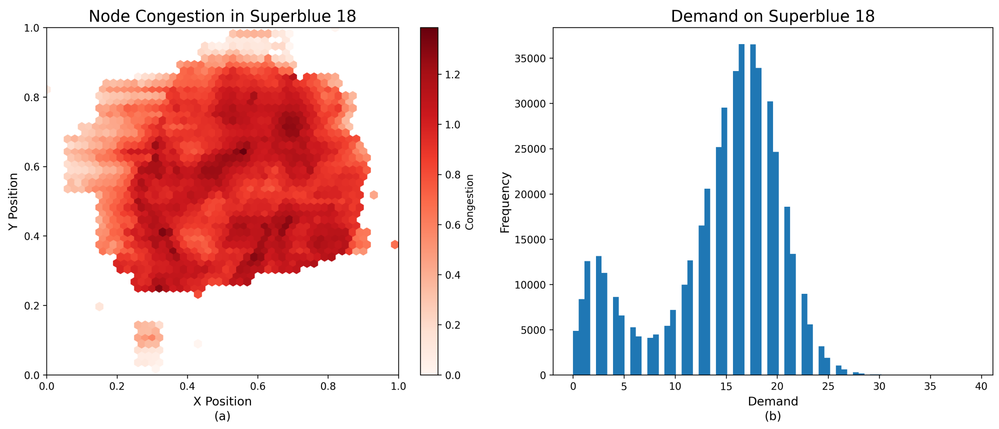


Figure 1: (a) A heatmap depicting node congestion on Superblue 18. (b) A histogram depicting node and net demand across Superblue 18.

Visualizing routing congestion is a key early step in the physical design process. We started by creating a heatmap of congestion across the chip to illustrate where the design is overstressed. Congestion is mathematically defined as the ratio of demand/capacity where values ≥ 0.9 are labeled *congested*. The darker areas in Figure 1a indicate regions where the number of required routing tracks exceed availability, i.e. *congested*. In practice, these maps can highlight issues such as too many cells being bunched near a macro pin orchannel, with multiple long nets running through. It is also relevant in machine learning contexts when considering the importance of identifying repeating graph substructures.

Demand is the amount of routing resources that need to pass through a GRC, and is a target variable in our congestion prediction tasks. We visualized the routing demand distribution – the number of routing tracks necessary in each region of the chip — with a histogram (Figure 1b). In the Superblue 18 circuit, demand seems to be bimodal, with most demand concentrated around 3 and 17. The demand histogram for this particular chip is also narrow and somewhat centered on one of the peaks, indicating that the routing load is relatively balanced across the chip. In contrast, demand distributions on other chips followed a wide or skewed distribution with significant outliers, indicating uneven resource utilization or poor design.

These analyses reveal how routing resources are utilized across the design, identifying patterns and outliers. Most Superblue chips have a small fraction of regions with sizable demand; these regions have disproportionately large routing loads, making them an important target for future machine learning models.

3 Methods

3.1 Explanation

3.1.1 Ablation Study

We wanted to determine the relative importance of each feature in the model to better understand how the DE-HNN model makes predictions. We first implement an ablation study. By inspecting the performance of the model by dropping one feature at a time, we can rank the features according to their impact on the model.

We present our results below, with a focus on the following features:

- Cell Type - the type of logic gate that the instance is
- Width and Height - physical dimensions of the cell
- Orientation - based on the rotation of the cell
- Degree - The number of nets a cell is a part of
- Laplacian Eigenvectors - Spatial embeddings that reveal clusters in the graph
- Degree Distribution - A local view of the graph, counting the number of neighbors at increasing distances
- Persistence Diagram - A summary of topological features encoding growing neighborhood around a node

Table 1: Ablation Study with DE-HNN Model

Feature	RMSE	% Change
Baseline	6.405	-
Cell Type	6.405	+0
Height	6.403	+0
Width	6.448	+0.7
Orientation	6.401	-0.1
Degree	6.410	+0.1
Eigenvectors	6.519	+1.8
Degree Distribution	6.415	+0.2
Persistence Diagram	6.385	-0.3
All Noise	6.419	+0.2

The ablation study on the DE-HNN model found that removing features had almost no effect on the performance of the model, with the maximum change being +1.8% from removing eigenvectors. This suggests that the model may only be learning from the graph structure itself, not the actual features (most of which encode the graph structure in some way). To test this, we reduced all features to noise, and tested the performance of the model again. (the final row in Table 1).

With pure noise, the model performed only 0.2% worse than with the actual features, which shows that the baseline DE-HNN learns from the structure of the graph, and not the features of each node and net. Figure 2 shows that the performance of the randomized dataset mimics that of the baseline.

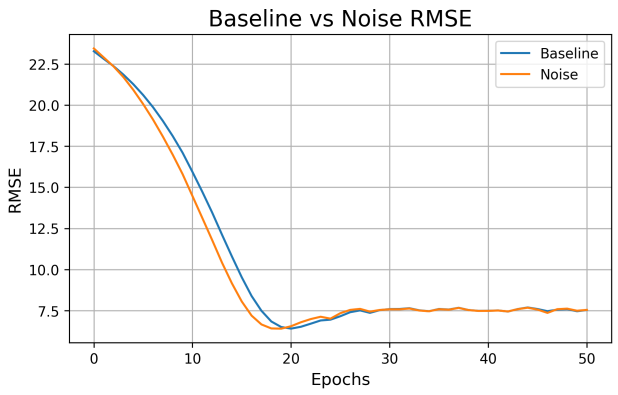


Figure 2: RMSE of the test split of both the baseline model and the random features over 50 epochs.

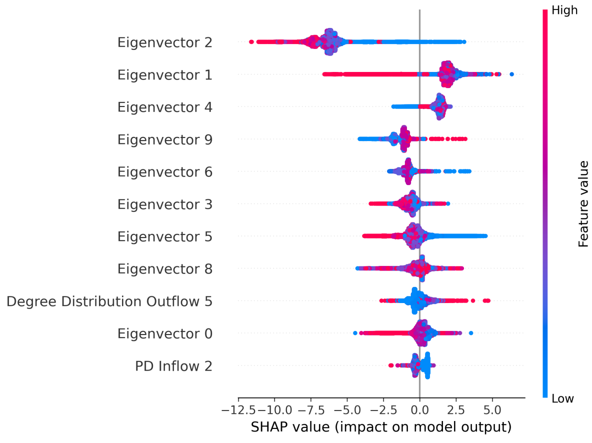
However, we believed that an understanding of feature importance would help guide future work, so we additionally conducted an ablation study using a Random Forest, as it can only use the features and not the connections. Our results are presented below:

Table 2: Ablation Study with Random Forest

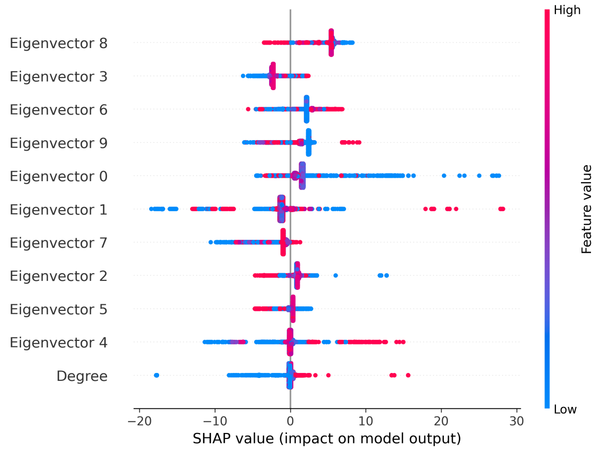
Feature	RMSE	% Change
Baseline	5.313	-
Cell Type	5.354	+0.8
Height	5.379	+1.2
Width	5.374	+1.1
Orientation	5.366	+1.0
Degree	5.322	+0.2
Eigenvectors	6.234	+17.3
Degree Distribution	5.653	+6.4
Persistence Diagram	5.425	+2.1
All topological features	9.267	+74.4

This ablation study showed that the most important features are the eigenvectors, degree distribution, and persistence diagram. Since these are all the topological features that relate to graph structure it makes sense that the connections themselves may be more important than the cell features. If we remove all of the topological features at once, the performance worsens by 74.4%, further demonstrating that the graph structure is crucial to predicting demand.

3.1.2 SHAP Analysis



(a)



(b)

Figure 3: (a) A SHAP Summary Plot for the Node Features (b) A Shap Summary Plot for the net features. For both plots, the color represents the value for that feature, and the position on the axis represents the feature's impact on the model output for that datapoint.

Another way to quantify feature importance is with SHAP, or SHapley Additive exPlanations. SHAP is a game theory-based method for explaining predictions of a machine learning model by calculating the impact of each feature in every prediction and provides both a magnitude and direction for the effect of a feature. We ran the SHAP algorithm on a LightGBM model using the same train/test split as in the baseline, and generated the feature importance for both the node and net level predictions. Figure 3a shows the resulting summary plot for the node features.

This SHAP plot for the node predictions supports the findings from the previous ablation study; the eight most impactful features are eigenvectors, and all ten eigenvectors are within the top twelve out of the 45 total features. The SHAP summary plot for the net features (Figure 3b) also highlights the importance of the eigenvectors, which have a significantly greater impact than the degree. We are unable to perform significant analysis on the directionality of each feature's impact, as the eigenvectors are simply embeddings of the graph structure, but knowledge of the eigenvectors' relative importance can help guide future experiments.

3.2 Experimentation

Following our initial EDA and attempts at drawing insight on the baseline model through explainable AI methods, we constructed a set of informed experiments to tackle key issues with the baseline model: long runtime, lack of generalizability, and limited learning on the node and net features.

3.2.1 Tree-Modeling

As previously mentioned, we noticed that the baseline DE-HNN model was not learning from the node and net features, and was instead only making predictions based on the structure of the graph, formed by the millions of connections in the training dataset. From our ablation study and SHAP analysis, however, we knew that there is some value in the features, and they can also serve as a representation of the greater graph structure, such as through the eigenvectors. This led us to considering tree-based models as an alternative architecture for the net demand prediction task, specifically Random Forest and LightGBM. These models contained only net degree and eigenvectors as features, but we hypothesized that this approach would reduce training time, while still making relatively accurate predictions.

3.2.2 Feature Engineering (Adding Capacity)

Although the DE-HNN does well at capturing the graph structure, it does not learn much from the node and net features. Therefore, we wanted to try adding a new feature called capacity into the model to see if it could perform better. We observed that capacity is correlated with demand, showcased in Figure 4. Capacity is a rough estimate of the amount of resources available for placing cells and routing wires in a given area. Importantly, capacity is *not* a product of the graph structure, rather, it is an (estimated) physical specification of an actual logic gate or net. We believe that the DE-HNN model can learn the interactions between capacity and dense topology, that it will learn that areas with low capacity but a large cluster of cells tend to be more congested. We hypothesize that including capacity will improve the performance of the model over the baseline. Further, we hypothesize that the LightGBM models will improve due to its reliance on the feature set.

3.2.3 Subpartitioning

In the DE-HNN model, virtual nodes are inserted into the graph to connect nodes within a given partition. This partition is identified using the METIS algorithm, well-regarded for its efficiency in handling large-scale graphs and producing balanced partitions while minimizing inter-partition edge cuts. There are two key parameters considered when initializing METIS: `nparts` and `ufactor`. `nparts` refers to the number of partitions the input graph should be divided into, and `ufactor` controls the permissible imbalance between partition sizes. A low `ufactor` enforces balance, but might hide natural variations in structure; a

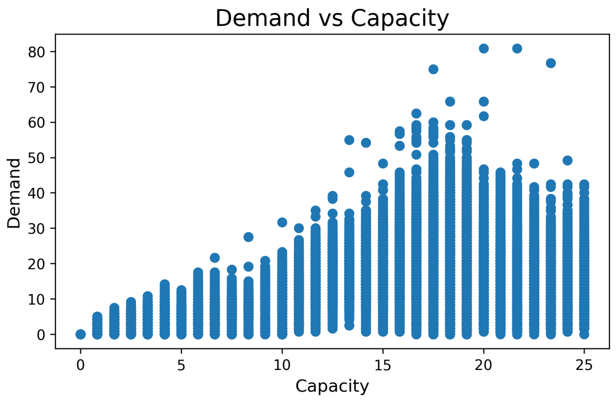


Figure 4: Capacity is positively correlated with demand with an R^2 value of 0.33.

higher `ufactor` might let natural clusters emerge even if they are of unequal sizes. While the DE-HNN model prioritized balanced partitions, our intuition regarding chip design suggested a higher `ufactor` may reveal clusters of nodes representing real, cohesive units within the chip.

We ran a parameter sweep that allowed us to evaluate partition quality in terms of conductance. A lower conductance value indicates that a partition is well-separated from the rest of the graph, suggesting a strong internal cohesion. The results of our parameter sweep are depicted in Figure 5. Based on our evaluations, a setting of `ufactor`=600 and 10 partitions was identified as optimal for the full graph.

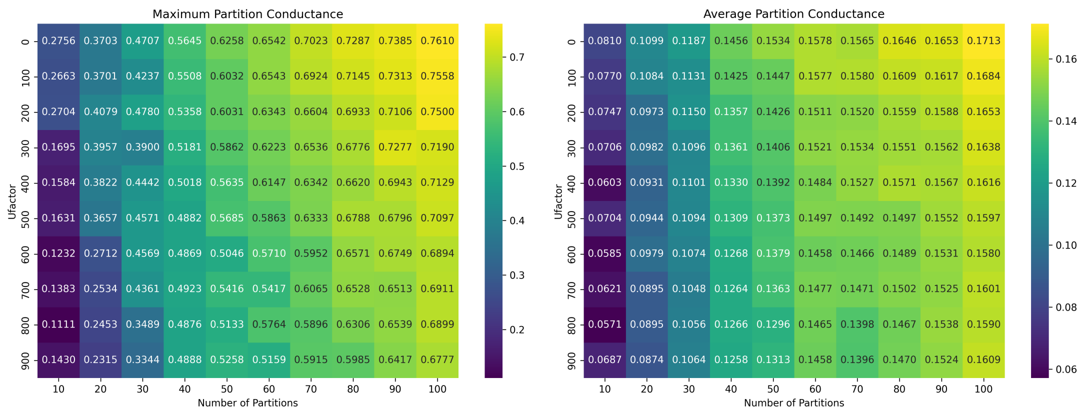


Figure 5: We varied the `ufactor` from 0 to 900 in steps of 100, and the number of partitions was varied from 10 to 100 in steps of 10. Both the maximum and average conductance values across partitions are computed. The maximum conductance reflects the worst-case quality among all partitions, whereas the average provides an overall measure of separability.

Recognizing that the global partitioning might obscure local substructures, we implemented a hierarchical refinement strategy, dubbed *Subpartitioning*. Following the initial partitioning, each of the 10 partitions was treated as an independent subgraph. For each subgraph, we reapplied the same heatmap-based parameter sweep and assessed local partition quality. The same parameter settings that were optimal for the global graph were identified as optimal for the subgraphs. This multi-level approach allowed us to capture nested modular structures that are characteristic of chip designs.

This strategy resulted in 1000 “new” graphs, potentially representing functionally cohesive blocks within the chip. To mimic the original set of node and net features, we recompute the degree and Laplacian eigenvector for each node and net. Though the training set has increased 100-fold, each graph is, on average, 1/100th the size of the original graph. This will dramatically cut the computational resources required to train the model. Further, we hypothesized that this strategy would allow the model to better understand the common substructures present across different designs, aiding the models generalizability on unseen designs.

3.2.4 Downsampling

In our EDA, we noticed that the demand variable is bimodal, with a smaller peak between 0 and 5 and a much larger peak between 15 and 20 (refer to Figure 1b). In conducting an error analysis on the baseline model, we noticed that it generally did well at predicting values around the larger peak, but failed to recognize the second common range of values.

To try and create a more representative model that derives learnings relevant to the full dataset, we decided to implement *downsampling* - a process in which the training data is binned by the target variable, and each bin is randomly sampled to ensure equal sizes. This ensures that the dataset is balanced across the distribution of the target variable, and will ideally allow us to improve the model’s performance. Choosing the number of bins and the bounds of each bin was an important task, as these outputs could significantly alter the final model’s predictions. To ensure a balanced dataset, while minimizing the amount of data removed, we performed a parameter sweep over the bin count and size. We also ensured that the edges also matched the new, smaller dataset by removing connections that involved a removed node, which may have damaged local structures, but helped shrink the overall dataset.

From this downsampling process, we hypothesized that we would be able to improve test and validation accuracy, while preventing overfitting. We also expected that a reduced dataset size would improve the runtime.

3.2.5 Hypervectors

The core idea of a GNN is to learn how to represent each node in the graph by considering its own features, the features of its neighbors, and the structure of the graph. The GNN allows information to propagate through the graph with message passing, or neighborhood aggregation, a process that is repeated in every layer of the model. This is a very complex model, especially when using virtual nodes to capture long-range interactions.

We believe that there may be limitations of representing data as nodes and edges in a graph, which essentially reduces the dimensionality of the data. Intuitively, AI models thrive on high-dimensional data, and GNNs, by their very nature, restrict the dimensionality. We decided to implement Hypervector Feature Encoding: either replace or augment DE-HNN's input layer to use hypervector representations for nodes and nets, as they allow for a much richer representation of data with higher dimensionality. We started by encoding each node's features into a 10,000-bit random vector. To mitigate accuracy loss, we can introduce lightweight learned components on top of hypervector outputs. This can recover some of the fine-grained optimization capability of deep networks while still keeping the bulk of computation in the ultra-efficient HDC domain. We hypothesize that this change will help simplify the DE-HNN architecture, drastically reduce memory overhead (as there is no need to learn/store huge embedding matrices), and improve cross-design generalization.

4 Results

4.1 Tree-Modeling

We experimented with two tree-based modeling architectures, LightGBM and Random Forest. For each model, we ran a grid search to optimize for Validation RMSE + Test RMSE, identified optimal hyperparameters, and produced the results in Table 3.

Table 3: Tree-Modeling Losses

Model	Training RMSE	Validation RMSE	Test RMSE	Training Time (min)
Baseline GNN	17.23	5.01	6.40	41.97
LightGBM	10.89	8.52	13.45	0.46
Random Forest	10.73	7.83	11.03	10.81

Overall, these new approaches did not have as much success as the baseline DE-HNN model, likely because there is importance in the connections between nodes that simply isn't captured by the feature set. However, the tree-models trained significantly faster, with the LightGBM model training nearly 100x faster than the baseline DE-HNN, and the Random Forest model about 4x faster. This suggests that there may be some value in further exploring this approach, especially with further optimizations or feature enhancements.

4.2 Feature Engineering (Adding Capacity)

Adding capacity only improved the performance of the DE-HNN model by 0.6%, showing that it did not have much of an impact on the model (showcased in Figure 6). This result follows our previous finding that the DE-HNN only learns from the graph structure and not the features. Since high capacity areas are correlated with densely connected areas in the graph, the DE-HNN may already be estimating capacity without needing an explicit feature for it.

However, we see an improvement of 41.6% in the LightGBM model upon including capacity (Table 4). Since a tree model has no way to model the graph structure, it needs to be

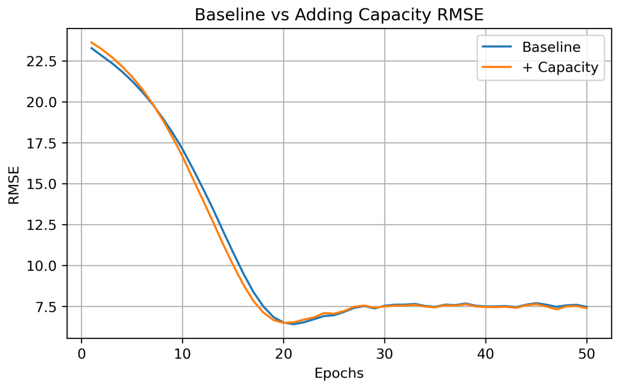


Figure 6: RMSE of the test split of both the baseline model and the baseline with capacity over 50 epochs.

Table 4: Adding Capacity Losses

Model	Training RMSE	Validation RMSE	Test RMSE	Training Time (min)
Baseline GNN	17.23	5.01	6.40	41.97
GNN + Capacity	17.21	5.01	6.36	42.00
LightGBM	10.89	8.52	13.45	0.46
LightGBM + Capacity	10.19	5.98	7.86	0.49

explicitly given features like capacity. Capacity is more correlated with demand than other features, making the LightGBM's performance comparable to the DE-HNN. Although the loss is still worse than the baseline, the training time is almost 100x faster, giving a great tradeoff of accuracy for speed.

4.3 Subpartitioning

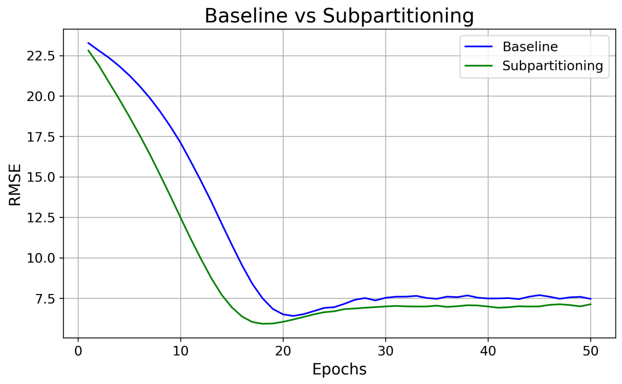


Figure 7: RMSE of the test split of both the baseline model and the subpartitioning model over 50 epochs.

Subpartitioning noticeably improves performance, while significantly reducing training time and required memory (Table 5). In initial experiments, we observed that the loss plots were highly erratic and failed to converge, due to the increased amount of training graphs.

Table 5: Subpartitioning Losses

Model	Training RMSE	Validation RMSE	Test RMSE	Training Time (min)
Baseline GNN	17.23	5.01	6.40	41.97
Subpartitioning	15.82	4.38	5.92	18.38

Reducing the learning rate by a factor of 100 allowed the model to quickly and smoothly converge. In fact, in Figure 7, we see the model achieve a lower test loss than the baseline model. These results suggest that reducing the problem was a viable strategy and allowed the GNN to better capture local dependencies. We believe the model learned to identify smaller common networks in the chip, aided by, importantly, training on smaller, cohesive graphs.

4.4 Downsampling

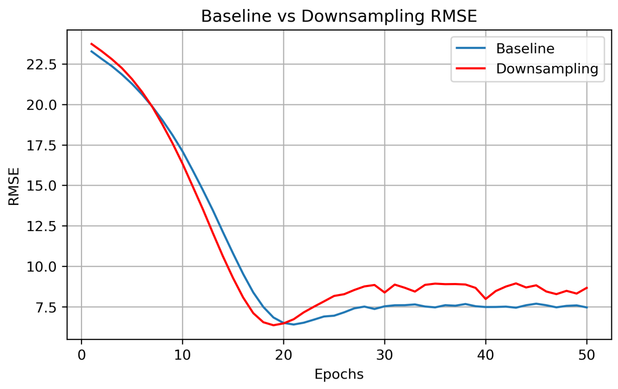


Figure 8: RMSE of the test split of both the baseline model and the downsampling model over 50 epochs.

Table 6: Downsampling Losses

Model	Training RMSE	Validation RMSE	Test RMSE	Training Time (min)
Baseline GNN	17.23	5.01	6.40	41.97
Downsampling	20.84	4.87	6.36	19.85

Downsampling slightly reduced both the validation and test RMSE, suggesting that this method did help the model generalize slightly better (Table 6). Even though the downsampled dataset was slightly different every time, due to random sampling, these results were relatively consistent. The larger improvement, however, was in terms of the training time, which was cut in half. This is driven by the significantly reduced dataset, not just in terms of the number of nodes, but also the connections between them. In addition, the model trained on the downsampled dataset converged slightly faster (Figure 8).

The fact that this approach was able to replicate the performance of the DE-HNN may suggest that not all the information in the dense graph is relevant, and that not much information is lost with a stratified downsampling approach. Using domain specific knowledge or a more informed approach to defining the bins for downsampling could lead to even further improvements.

4.5 Hypervectors

To incorporate the features in high-dimensional hypervectors to replace DE-HNN's learned node and net feature embeddings, each node or net is assigned a fixed hypervector (e.g. a 10,000-bit random vector which is easily initialized in python code) computed from its attributes, eliminating large embedding tables. We suspect this distributed representation is more highly generalizable – similar node features map to similar hypervectors, improving the model's ability to transfer to new circuits.

We found integrating hypervectors in place of embeddings is faster and more memory efficient, although it wasn't as accurate. More specifically, a model trained on hypervectors was able to identify congestion with 70-80% accuracy, which was lower than the baseline. There is thus a potential accuracy trade-off: hyperdimensional computing (HDC) might sacrifice a bit of peak accuracy in exchange for gains in speed and memory.

5 Discussion

The success of alternate representations of a chip design, through downsampling and subpartitioning, suggest that the full graph structure is too complex for the GNN to model. As it stands, we believe local and global information is still not easily propagated through the model. Our experiments validate two methods to reduce noise and improve generalizability on different levels of granularity. Downsampling "simplifies" the graph while mimicking its original characteristics (through effective stratified sampling). Each design likely shares similar overarching characteristics. Downsampling attempts to capture this larger structure while limiting the various sources of noise present throughout the dense network of each chip. Subpartitioning, on the other hand, more strictly adheres to the chip design, but prioritizes local dependencies with the belief that these structures are representative of smaller blocks common within all designs. Both of these methods reduce the complexity and overall size of the dataset, resulting in a drop in required computing resources and faster training.

Experiments with tree-based models also produced interesting results. Particularly, after adding capacity as a feature, LightGBM approached similar losses as the baseline DE-HNN. When looking at the SHAP analysis, this predictive success was in large part due to the eigenvectors, which serve as a representation of the graph structure. This approach wasn't as effective as a full GNN, however, suggesting that there is value to the edges that cannot be described by the eigenvectors alone.

As naive as our algorithms were, we believe that their performance can be enhanced with more thoughtful design choices suggested by engineers from various points in the chip design lifecycle. Such domain expertise will help identify what aspects of a design can be prioritized or ignored when determining these alternate representations. We believe that a close look into stratification strategies can create more expressive downsampled graphs. Similarly, we believe that even finer substructures can be extracted while subpartitioning, or, alternatively, some structures are better analyzed as larger, cohesive pieces.

Of course, with different constraints, from varying use-cases to time and compute restrictions, comes a trade-off. A cursory analysis early in a chip's design might not require the extensive training time of a GNN, and can be sufficiently modeled with a tree architecture. Certain chips may call for the preservation of local structures and a partitioning approach. Designing a new chip from scratch may benefit from building up a downsampled design of a similar chip.

Expanding on these experiments will likely depend on specific contexts, and we believe they can be further by methodical, domain-guided alterations to the algorithms we propose. It will be interesting to further explore this direction.

References

Luo, Zhishang, Truong Son Hy, Puoya Tabaghi, Donghyeon Koh, Michael Defferrard, Elahe Rezaei, Ryan Carey, Rhett Davis, Rajeev Jain, and Yusu Wang. 2024. "DE-HNN: An effective neural model for Circuit Netlist representation." [Link]

Viswanathan, Natarajan, Charles J. Alpert, Cliff Sze, Zhuo Li, Gi-Joon Nam, and Jarrod A. Roy. 2011. "The ISPD-2011 routability-driven placement contest and benchmark suite." In *Proceedings of the 2011 International Symposium on Physical Design*. New York, NY, USA Association for Computing Machinery. [Link]

Appendices

Previous Project Proposal