Congestion Prediction in Integrated Circuit Design Using Graph Neural Network

Yuyang Pang yupang@ucsd.edu

Vivian Chen vnchen@ucsd.edu

Michelle Tong m1tong@ucsd.edu

Hang Liu hal064@ucsd.edu

Lindsey Kostas lkostas@qti.qualcomm.com

Abstract

Congestion prediction is a crucial part of the design process of Integrated Circuits (IC). It allows chip designers to discover areas that have excessive routing demand and optimize the circuit accordingly. This paper presents insights into the use of Graph Neural Networks for early congestion prediction in IC designs. Utilizing the NCSU-DigIC-GraphData dataset, which includes 13 small netlists with distinct congestion characteristics, our methodology includes node feature engineering, multi-graph train-test split, and a comparison between model architectures, which features Graph Convolutional Networks (GCNs) and Graph Attention Networks (GATs). Results show that Graph Neural Networks do not scale well on small netlists that only contain a few thousand nodes. GATs can perform better than GCNs but have significantly longer training time. Moreover, GRC-based prediction models are inferior to node-based prediction models.

1 Introduction

Predicting congestion in IC design is a critical yet challenging task. Congestion occurs when excessive wiring or routing is required in certain regions of the chip layout, often resulting from suboptimal floor planning or high cell density. This can lead to significant issues, such as timing discrepancies, crosstalk, increased power consumption, reduced reliability, and high production costs. Early prediction and mitigation of congestion are thus essential to avoid these complications.

Current optimization tools for calculating congestion are extremely time-consuming given the complexity of modern chips. A machine learning model that utilizes the existing information of netlists and their congestion can accelerate the optimization process in IC design. Predicting congestion is generally complex due to the intricate nature of IC layouts. This project seeks to address this challenge by leveraging Graph Neural Networks (GNNs), which hold promise due to their ability to model complex relationships and interactions in graph-structured data.

Related Works The approach of using graph neural networks in congestion prediction has been explored in recent works. Kirby et al. utilized deep Graph Attention Networks on the hypergraph representation of IC designs to predict congestion [2]. This graph neural network-based approach has outperformed existing methods of congestion prediction by accuracy and speed. Moreover, GNNs can predict congestion on a per-cell level and don't require any placement information.

Given the advantages of GNNs in circuit congestion prediction, our project aims to explore similar GNN architectures from Kirby et al.'s work on the NCSU-DigIC-GraphData dataset and compare the performance of different GNN architectures. NCSU-DigIC-GraphData has a much smaller scale compared to the netlists used in Kirby et al.'s paper, which has roughly 50 million cells from tens of designs. Experimenting with graph neural networks on different sizes of netlists can give us insights into how well GNNs scale and how different GNN architectures perform on a smaller-scale dataset.

2 Methodology

2.1 Dataset

This project utilizes the NCSU-DigIC-GraphData, which comprises 13 netlists, each exhibiting unique placement and congestion characteristics. For each netlist, the dataset contains node features, node connectivity, and Global Route Cell (GRC) based demand and capacity. GRCs are organized in a rectangular grid (Figure 1), but not all GRCs have the same dimensions. Detailed information about the dataset can be found in the appendix A.

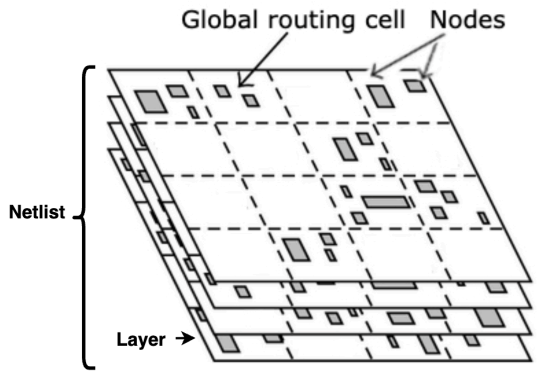


Figure 1: Nodes and GRCs

Table 1 presents the statistics of each netlist, including the numbers of cells/edges/layers, connection information, and congestion distribution. Congestion was calculated using demand minus capacity.

Table 1: Data Summary

Netlists	Cells	Edges	Layers	Edges Per Cell				Congestions			
				Min	Max	Mean	STD	Min	Max	Mean	STD
XBar_1	3952	4482	4	15	862	583.39	330.37	-48.0	14.0	-9.42	6.12
XBar_2	6872	7404	7	8	1253	166.06	253.38	-86.0	3.0	-28.55	12.52
XBar_3	6913	7445	7	8	1277	166.45	251.10	-86.0	-11.0	-35.14	10.01
XBar_4	7323	7855	7	8	1274	144.56	238.81	-86.0	-11.0	-34.64	9.73
XBar_5	7258	7790	7	8	1254	145.72	233.62	-86.0	-13.0	-35.23	9.73
XBar_6	7120	7652	6	8	1253	136.50	187.04	-78.0	-8.0	-28.69	9.13
XBar_7	7879	8411	5	7	1277	128.23	212.35	-65.0	20.0	-15.78	9.55
XBar_8	7626	8158	5	7	1274	135.84	224.04	-67.0	18.0	-18.43	9.22
XBar_9	7620	8153	5	6	1229	132.03	206.79	-76.0	-8.0	-29.67	9.23
XBar_10	7772	8304	6	8	1292	130.48	216.23	-78.0	-11.0	-31.68	8.36
XBar_11	7814	8346	5	8	1283	118.99	174.60	-67.0	22.0	-16.33	8.73
XBar_12	6529	7061	5	8	1280	187.29	271.56	-67.0	15.0	-14.10	9.83
XBar_13	6548	7082	5	6	1283	178.85	248.45	-65.0	29.0	-11.19	9.76

Given that the dataset doesn't contain layer information for nodes, GRC-based congestion is summed across layers and mapped to all the nodes inside each GRC for the prediction task. Figure 2 shows the heatmaps of congestion of layers in Xbar_1 and the combined heatmap of congestion of Xbar_1.

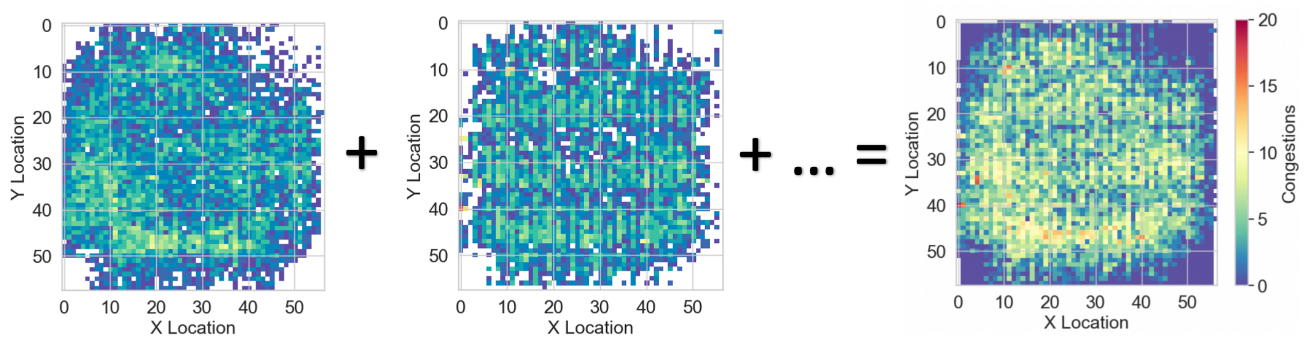


Figure 2: Congestion Heatmaps

To examine the effect of mapping GRC-based congestion to cell-based congestion, we can compare the histograms of per GRC congestion and per cell congestion. Figure 3 shows that we lost some lower-end congestion, mainly due to fewer numbers of cells in lower-end congestion GRCs. Since we care more about the congested regions in this project, predicting congestion per cell is reasonable, as it gives more node features.

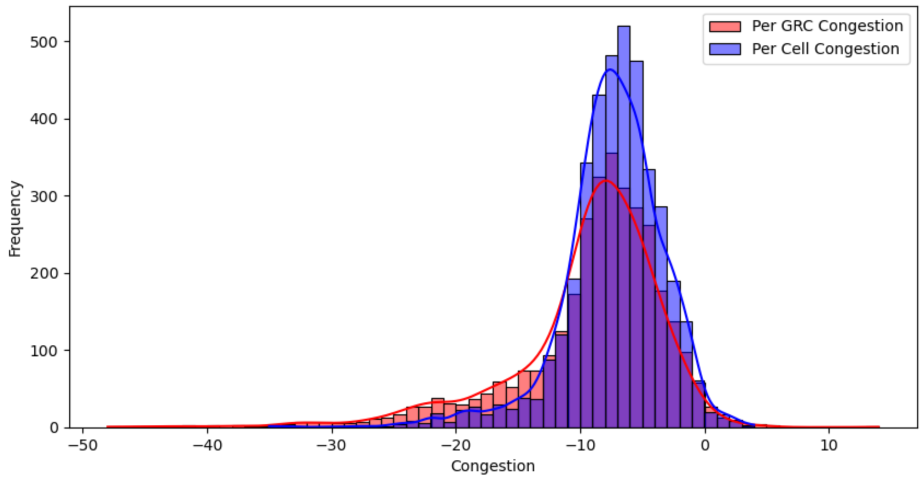


Figure 3: Xbar_1 Congestion Distribution

2.2 Feature Selection and Engineering

Selecting node features is essential for training the Graph Neural Network (GNN). The dataset gives basic node features, including the locations of nodes, the dimensions of the cells, and their orientations, as detailed below:

- **xloc and yloc**: The locations of nodes (`xloc` and `yloc`) provide the relative positions for the model to learn where congestion typically occurs within the chip layout.
- **width and height**: Utilizing the size of the nodes, specifically the width and height of the cells, serves as a convenient method to encode the cell type directly into the model, offering insights into the physical space occupied by each element.
- **orient**: The orientation of the cells might indicate the preferred direction of routing, offering a hint towards potential congestion areas.

Besides the existing node features, some related features are calculated and assigned to each node.

- **cell density**: The number of cells inside each GRC. By assigning cell densities to nodes, the model can use them to reference how congested different areas are.
- **input and output degrees**: The number of connections a node has to other nodes. It serves as critical routing information and helps with the prediction task.
- **GRC Indexes**: Unique x and y index for each GRC. Even though xloc and yloc already encode the spatial information of nodes, GRC indexes can provide extra references for the model to understand the spatial relationships of nodes.

To examine the correlation between these features and demand/congestion. We calculated the Pearson correlation coefficients (Table 2). In the table, we removed height and orientation because all cells have the same height and orientation is not ordinal. The correlation coefficients indicate an association between selected features and targeted congestion.

As one can observe, the correlation between congestion and demand is very high (0.778). As stated before, congestion is defined as (demand - capacity). In other words, the existence

Table 2: Correlation Table

	congestion	xloc	yloc	width	GRC_x	GRC_y	GRC_density	node_degree
demand	0.778	−0.121	0.115	−0.556	0.115	−0.122	0.301	−0.146
congestion	1.000	−0.151	0.118	−0.531	0.118	−0.153	0.299	−0.121

of demand is what causes congestion, which explains why the correlation between these two variables is notably high. Therefore, we will consider the idea of constructing a model that predicts demand instead of congestion.

To ensure our model's robustness across various designs, feature normalization is performed. The `xloc`, `yloc`, `width`, and `height` values are normalized before being fed into the model, allowing it to generalize across designs with differing dimensions. The orientation feature (`orient`) is encoded using one-hot encoding to capture its categorical nature effectively.

Additionally, connectivity data is transformed into an Edge Index using the adjacency matrix, an important step for enabling the GNN to understand the connections between nodes. This Edge Index serves as a key input, allowing the network to learn the intricate relationships and pathways within the graph structure.

Congestion/Demand data, combined from all layers, is mapped to each node based on the Global Route Cell (GRC) boundaries. Congestion can be computed as the difference between demand and capacity, providing a quantitative measure of congestion at each node.

2.3 Model Architecture

Our project employs Graph Convolutional Network (GCN) and Graph Attention Network (GAT) architectures.

2.3.1 Graph Convolutional Network (GCN)

GCN models utilize the graph-based structure of circuit designs, where nodes represent circuit components, and edges denote connections between these components. The model utilizes spatial dependencies between components to predict congestion by encoding both local and global structures of the circuit.

Architecture Details:

- **Node Feature Encoding:** Each node v_i in the circuit graph $G(V, E)$ is associated with a feature vector x_i that represents the physical properties of the circuit components, such as size, orientation, and connectivity.
- **Graph Convolution Layers:** The model comprises multiple graph convolution layers. Each layer transforms the node features into higher-level representations. A

graph convolution layer is defined as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

where $H^{(l)}$ is the matrix of node features at layer l, $\tilde{A} = A + I$ is the adjacency matrix A with added self-loops I, $\tilde{D}$ is the degree matrix of $\tilde{A}$, $W^{(l)}$ is the weight matrix for layer l, and σ denotes a nonlinear activation function such as ReLU. This equation is given by Kipf et al. [1]

- **Pooling and Readout:** After several graph convolution layers, a global pooling layer aggregates the node features to capture the overall graph structure. The readout layer maps these aggregated features to congestion predictions.

2.3.2 Graph Attention Network (GAT) for Enhanced Feature Learning

To refine the model's capability in capturing intricate dependencies between circuit components, we incorporate a Graph Attention Network (GAT) layer, which assigns varying importance to different nodes within a neighborhood. The equations below are derived from Veličković et al. [3].

Architecture Details:

- **Attention Mechanism:** The GAT layer computes attention coefficients for node pairs, indicating the importance of node j's features to node i:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^\top[Wh_i \| Wh_j]))}{\sum_{k \in N_i} \exp(\text{LeakyReLU}(\mathbf{a}^\top[Wh_i \| Wh_k]))}$$

where h_i and h_j are the feature vectors of nodes i and j, respectively, W is a shared weight matrix, $\mathbf{a}$ is the attention mechanism's weight vector, and N_i denotes the neighbors of node i.

- **Multi-head Attention:** To stabilize the learning process, we employ multi-head attention, where K independent attention mechanisms execute in parallel, and their features are concatenated. This leads to the following representation for each node:

$$\text{MultiHead}(h_i) = \|_{k=1}^{K} \sigma\left(\sum_{j \in N_i} \alpha_{ij}^k W^k h_j\right)$$

where $\|_{k=1}^{K}$ denotes concatenation of the outputs from K attention heads.

- **Integration into GCN:** The GAT layers are integrated with the GCN framework, allowing the model to leverage both the structured feature propagation of GCNs and the adaptive attention mechanism of GATs for enhanced congestion prediction.

2.3.3 GRC-Based Prediction Model

Since the demand and capacity data is stored per GRC, predicting congestion per GRC may have better performance compared to predicting congestion per node. Furthermore, per node prediction fails to predict congestion for GRCs that are empty. Therefore, we decide to test a GRC-based model architecture alongside with node-based model.

To utilize GNNs, we treat each GRC as a node and compute GRC features and GRC edge index. GRC features are the averages of node features in each GRC. Empty GRCs are imputed using their neighbors' features. GRC edge index is derived from the node edge index. Duplicate edges are retained to preserve the connectivity information between GRCs.

2.4 Training and Optimization

The model utilizes the Mean Squared Error (MSE) loss function, apt for regression tasks aiming to predict a singular continuous value per node. The Adam optimizer, with a learning rate of 0.005 and a weight decay of 5e-4, is used for optimization, supplemented by a dropout rate of 0.6 in the `GATConv` layers and between layers to prevent overfitting.

2.4.1 Data Balancing

To match the of variance of predicted congestion, we implemented a weighed loss criterion, where our targets are split into bins and their weight is the inverse frequency of their respective bin. Due to this, the less frequent bins will have more weight and the more frequent bins will have less weight.

2.4.2 Train-Test Split

We adopt two training settings for our models:

Training within a single file. We randomly pick 80%/10%/10% of nodes within one netlist to serve as our training, validation and testing set.

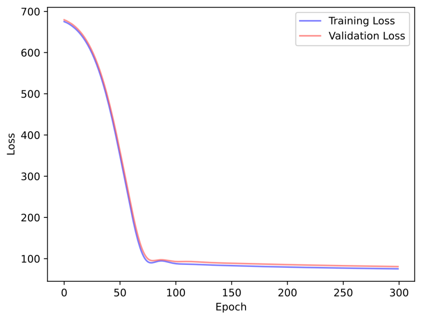
Training across files. We leave one netlist out and utilize the other 12 netlists as our training set. This cross-files training method broadens the scope of our model's learning to a wider array of data variations, which enhance its generalization capabilities.

3 Results

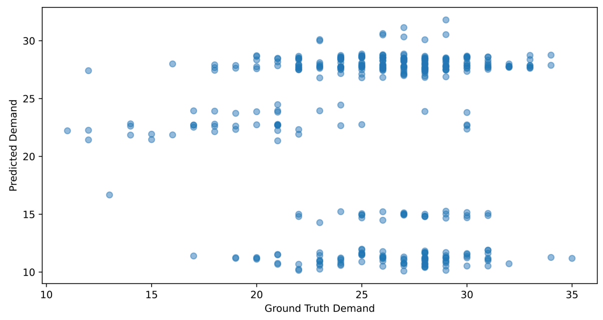
3.1 Node-based Models

By training the GCN model on one netlist (Xbar_1) that contains 3952 nodes, it reached a training loss of 75.368 and a test loss of 82.583. From figure 4a, we can tell that the training and validation loss decreased exponentially and are healthy.

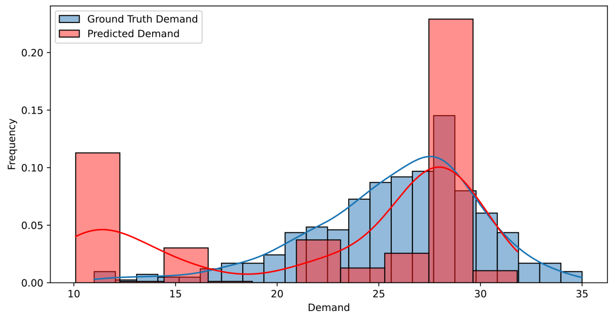
When we plotted the scatterplots of ground truth demand and predicted demand (Figure 4b), we found that our model tends to predict the mean of the congestion. This may be due



(a) GCN Loss



(b) GCN Prediction Scatterplot



(c) GCN Prediction Histogram

Figure 4: GCN Model

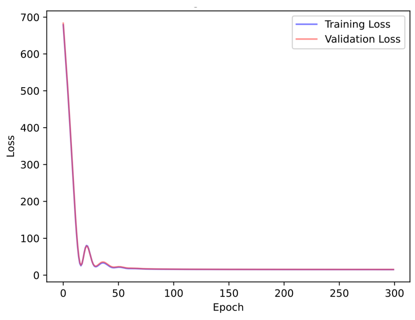
to the MSE loss function we used. It reflects that the model failed to capture the relationship between node features, connectivity, and demand.

On the other hand, the GAT model reached a training loss of 15.083 and a test loss of 13.418. Moreover, Pearson's correlation coefficient for the predicted demand and ground truth is 0.472 for the GAT model, much higher than the correlation coefficient of the GCN model (0.005). To test our models' robustness, we separately trained the GCN and GAT models on every design and calculated the RMSE, Pearson's r, Kendell's Tau B on the test sets.

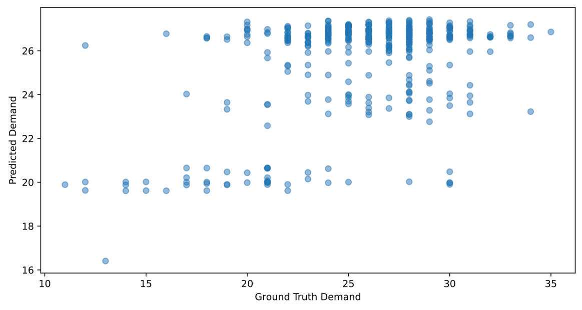
Table 3: Model Performance on Test Set

Model	Single-Design			Cross-Design		
	RMSE	Pearson	Kendell	RMSE	Pearson	Kendell
GCN	9.857	0.152	0.089	11.902	-0.170	-0.129
GAT	6.211	0.458	0.205	14.919	0.554	0.150
Naive Mean Predictor	6.995	NA	NA	4.137	NA	NA

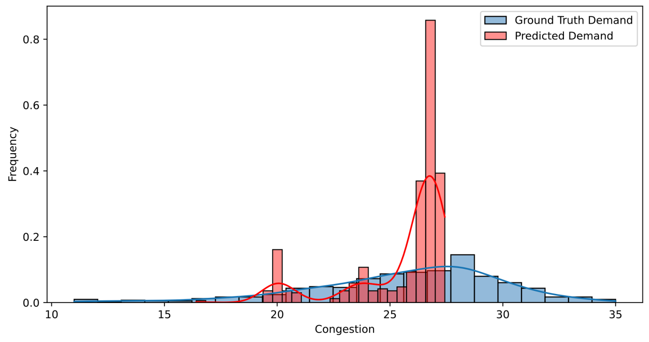
Table 3 contains model performance on single designs and cross designs settings. The performance of a naive mean predictor is also included to serve as a reference for model performance. Generally, GATs have lower RMSE and higher Pearson and Kendell in both single-design training and cross-design training compared to GCNs.



(a) GAT Loss



(b) GAT Prediction Scatterplot



(c) GAT Prediction Histogram

Figure 5: GAT Model

3.2 GRC-based Models

We experimented with a per GRC prediction model using a GAT model because we believed GRC-based models may have the lowest MSE given the nature of how demand is stored in the dataset. However, the prediction has the worst correlation coefficient with the ground truth (-0.0855) compared to cell-based models. Figure 6 shows the prediction has no patterns at all. This might be caused by a lack of node features in the GRC-based models.

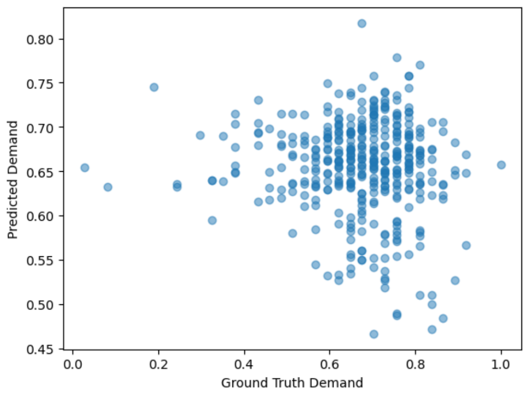


Figure 6: GRC-based GAT Prediction Scatterplot

4 Conclusion

In this project, we have tested both GCN and GAT models on the NCSU-DigIC-GraphData dataset. We discovered that models trained on one netlist are not robust and tend to vary a lot in terms of MSE and Correlation Coefficient. We compared GAT's performance on a single design and multiple designs and observed better results than GCN models. However, we noticed that GAT can take more time to train compared to GCN models if we increase the number of heads in the attention mechanisms. Node-based models are superior to GRC-based models based on their performance and robustness. Overall, GCN models perform worse than a naive mean predictor, while GAT models are slightly better than a naive mean predictor, indicting the poor scalability of GNN models on small netlists.

References

[1] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: vol. abs/1609.02907. 2016. arXiv: 1609.02907. URL: http://arxiv.org/abs/1609.02907.

[2] Robert Kirby et al. "CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks". In: *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE. Santa Clara, CA, USA, 2019, pp. –. DOI: 10.1109/VLSI-SoC.2019.8920342.

[3] Petar Veličković et al. "Graph Attention Networks". In: 2018. arXiv: 1710.10903 [stat.ML].

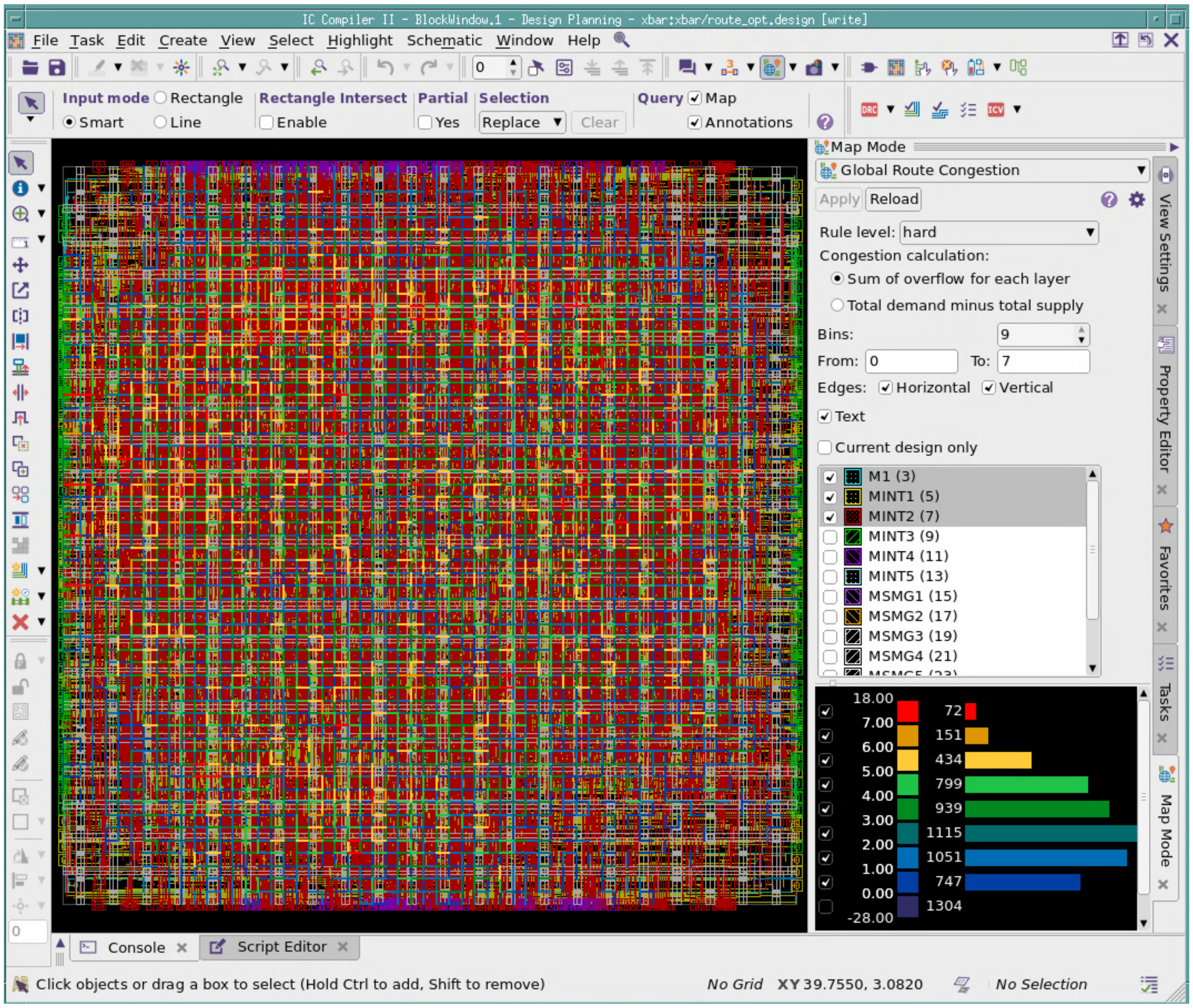
A NCSU-DigIC-GraphData

Design Data Files Each netlist's design data is encapsulated within three primary files:

- **[design].json.gz**: This file stores the feature data, including both instance and net data. The key attributes are:
  - `name` (string): Name of the instance.
  - `id` (integer): Index of the instance in the array.
  - `cell` (integer): Master library cell ID (array index).
  - `xloc`, `yloc` (integer): Location of the instance in database units (DBU). To convert to user units (UU), i.e., microns, divide this by the `DBUtoUU` divisor (e.g., 1000 or 2000, as specified in the README).
  - `orient` (integer): Orientation of the instance.
- **[design]connectivity.npz**: Stores the incidence matrix data in the form of NumPy arrays.
- **cells.json.gz**: Contains the cell library data with the following key attributes:
  - `name` (string): Name of the cell/module.
  - `id` (integer): Index of the cell in the array.
  - `width`, `height` (integer): Dimensions of the cell in database units. Convert to microns by dividing by `DBUtoUU`.
  - `terms`: An array of terminal objects.
- **[design]congestion.npz**: encapsulates the congestion data in NumPy arrays, crucial for understanding the target values within our project. Rather than individual instance congestion values, the dataset provides congestion information for each Global Route Cell (GRC) as follows:
  - **Demand**: Number of routing tracks required in the GRC.
  - **Capacity**: Maximum possible routing tracks within the GRC.

  GRCs are organized into a rectangular grid, albeit with varying dimensions. The dataset employs `x|yBoundaryList` arrays to delineate GRC boundaries, aiding in the translation of instance locations to array indices. All dimensions are provided in database units. Notably, since IC layout designs incorporate multiple layers, congestion data is distinctly stored for each layer.

  Setting File The dataset includes a setting file documenting core utilization and the maximum routing layer for each netlist.



Global Route Congestion

Rule level: hard

Congestion calculation:
(● Sum of overflow for each layer) ○ Total demand minus total supply

Bins: 9

From: 0 To: 7

Edges: ✓ Horizontal ✓ Vertical

✓ Text

☐ Current design only

✓	M1 (3)
✓	MINT1 (5)
✓	MINT2 (7)
☐	MINT3 (9)
☐	MINT4 (11)
☐	MINT5 (13)
☐	MSMG1 (15)
☐	MSMG2 (17)
☐	MSMG3 (19)
☐	MSMG4 (21)

✓	18.00	72
✓	7.00	151
✓	6.00	434
✓	5.00	799
✓	4.00	939
✓	3.00	1115
✓	2.00	1051
✓	1.00	747
☐	0.00	1304
	-28.00	

Click objects or drag a box to select (Hold Ctrl to add, Shift to remove) No Grid XY 39.7550, 3.0820 No Selection

Figure 7: Xbar-Congestion with First 3 Layers Selected

B Model Configuration

To facilitate a comprehensive evaluation of our models, we meticulously configure the Graph Convolutional Network (GCN) and Graph Attention Network (GAT) architectures. This section delineates the specific configurations and hyperparameters employed for each model, ensuring reproducibility and clarity in our experimental setup.

B.1 Graph Convolutional Network (GCN) Configuration

The GCN model is structured to efficiently capture the spatial dependencies inherent in our graph-based dataset. The configuration details are as follows:

- **Input Layer:** The model ingests node features corresponding to the physical attributes of circuit components. The input dimension is tailored to match the size of these feature vectors.
- **Hidden Layers:** Two graph convolutional layers constitute the core of the model. The first layer expands the feature dimension to a higher-dimensional hidden space, facilitating intricate feature extraction. The subsequent layer further processes these features, preparing them for the final output stage. Each layer is followed by a ReLU activation function to introduce non-linearity.
- **Output Layer:** The final graph convolutional layer condenses the features into the target dimensionality, aligning with the congestion prediction requirements. This layer does not employ an activation function, providing raw predictions for post-processing or thresholding.
- **Dropout:** To mitigate overfitting, dropout is applied after each convolutional layer, excluding the output layer, with a dropout rate of 0.6.
- **Training Configuration:** The model is optimized using the Adam optimizer, with a learning rate of 0.005 and a weight decay factor of 5e-4. The training process spans 300 epochs, with early stopping criteria based on validation loss to prevent overfitting.

B.2 Graph Attention Network (GAT) Configuration

The GAT model leverages an attention mechanism to dynamically weigh the influence of neighboring nodes. This allows for a more nuanced aggregation of neighbor features. The model's configuration is as follows:

- **Input Layer:** Similar to the GCN, the input layer of the GAT model processes node features reflecting the characteristics of circuit elements.
- **Attention Layers:** The model comprises a sequence of GAT layers. The initial layer employs multiple attention heads to capture various aspects of the node relationships, enriching the feature space. The output of these heads is concatenated and fed into a subsequent GAT layer with a single attention head, focusing the model on critical features for prediction.
- **Output Processing:** The output from the final GAT layer directly corresponds to the predicted congestion metrics, adjusted through post-processing steps as necessary for evaluation against ground truth data.
- **Dropout and Activation:** Each attention layer is followed by a dropout mechanism with a rate of 0.6 to prevent overfitting, and an ELU activation function to introduce non-linearity.
- **Training Configuration:** The GAT model is trained using the Adam optimizer, with identical learning rate and weight decay settings as the GCN model. The training regimen also follows 300 epochs, incorporating early stopping based on validation performance to ensure model generalization.

Both models are rigorously evaluated on a hold-out test set, with performance metrics reported to facilitate comparison and assessment of their congestion prediction capabilities.