Optimizing Training Cost for Scalable Graph Processing

Han Hoang
gihoang@ucsd.edu

Joshua Li
jol029@ucsd.edu

Lindsey Kostas
lkostas@qti.qualcomm.com

Dhiman Sengupta
dhimseng@qti.qualcomm.com

Abstract

The DE-HNN model is a state-of-the-art (SOTA) graph neural network model to predict congestion using circuit netlist representation, outperforming other hypergraph and netlist models in both node-based and net-based demand regression tasks. However, its high computational demands pose a significant barrier to practical deployment, limiting scalability and efficiency. To address this challenge, our research focuses on exploring strategies to optimize the training cost of DE-HNN, in terms of memory usage or training runtime while preserving prediction quality. We systematically tune hyperparameters and profile metrics to analyze the trade-offs between computational cost and accuracy, aiming to identify the optimal balance between efficiency and performance across various model configurations.

1 Introduction

The DE-HNN model (Luo et al. (2024)) is a type of Graph Neural Network (GNN) designed for congestion modeling in chip design by leveraging hypergraph structures and virtual nodes to capture long-range dependencies in dense circuits. Although it demonstrates strong predictive performance, DE-HNN is computationally expensive due to its dual update mechanism for nodes and nets, as well as the construction of virtual nodes that facilitate message passing. These factors lead to high memory consumption and extended training times, limiting the model's scalability to larger and more complex circuit netlists. Given the increasing complexity of modern chip designs, reducing the computational cost of DE-HNN without significantly degrading predictive performance is essential for its practical deployment.

DE-HNN addresses these challenges by introducing a directed hypergraph representation of netlists, which preserves the hierarchical structure of circuit elements and differentiates between driver and sink nodes. It enhances message-passing effectiveness by incorporating hierarchical virtual nodes (VNs) and persistence-based topological summaries, enabling better scalability for large-scale designs. The model has demonstrated SOTA performance in predicting wirelength and congestion from netlist inputs, outperforming other SOTA machine learning models for hypergraphs and netlists such as NetlistGNN (Yang et al. (2022))

Unlike efforts that emphasize accuracy improvements, our research focuses on the investigation of cost-optimizing strategies for DE-HNN through hyperparameter tuning and model architecture adjustments. Specifically, we employ metric profiling to identify key computational bottlenecks in terms of both runtime and memory usage for DE-HNN. Our hypothesis is that reducing the number of layers in the model, thereby shortening the propagation interval, will lead to significant reductions in both memory requirements and runtime. This is because we suspect that neural message-passing between nets and nodes drives the majority of the model's computational cost, and by targeting this bottleneck, we can explore the trade-off between DE-HNN training cost and its performance.

2 Methodology

2.1 Environment Setup

All experiments were conducted on a UCSD DSMLP cloud system with NVIDIA RTX A5000 GPU (24 GB VRAM), using PyTorch 2.2.2 and CUDA 12.2.

2.2 Dataset Description

The data used in this project consists of circuit netlists, which naturally form hypergraphs where nodes represent circuit components and hyperedges represent electrical connections spanning multiple components. Unlike standard graphs, hypergraphs allow for higher-

order interactions, making them well suited for modeling complex dependencies in large-scale systems. Beyond chip design, hypergraph-based learning extends to various real-world applications, including social networks, biological systems, and recommendation engines, where relationships often involve multiple entities rather than simple pairwise connections. By improving the efficiency of training large hypergraph neural networks, our work contributes to the broader goal of making hypergraph-based models more scalable and practical for real-world use.

Due to the resource constraints of our project (see 2.1), we were limited to using only 6 Superblue designs in our experiments, with 5 for training and 1 for testing. However, this limitation does not undermine the significance of our results, as our focus is on the relative improvements in training cost and changes in model performance across different model configurations.

2.3 Baseline Model

We used the unmodified full DE-HNN model with virtual nodes and persistence diagrams as our baseline. Due to the resource constraint of our project (see 2.1), the number of layers in our baseline model is 3 and the number of dimensions is 32 instead of 64 as stated in the original paper (Luo et al. (2024))

2.4 Training and Evaluation Strategy

Minimize mean squared error (MSE) in the regression training set. Performance is monitored on the validation set during training, and final evaluation is performed on the test set to assess generalization to unseen data.

2.5 Optimization Strategy

We applied **iterative optimization** by first using early stopping, followed by architecture adjustments based on Grid Search, and finally incorporating a dynamic learning rate, while using a fixed random seed throughout for weight initialization to ensure reproducibility by controlling the effects of each optimization component.

2.5.1 Early Stopping (ES):

We created custom condition using 3 parameters: *patience*, *tolerance*, and *min epochs*. Training stops when validation loss exceeds a specified range (based on *tolerance*) around the average loss of prior epochs (determined by *patience*). This prevents overfitting while ensuring that training runs for at least *min epochs* number of epochs before the condition can trigger.

2.5.2 Architecture Adjustments (AA):

We used Grid search to explore combinations of two hyperparameters, the number of layers and dimensions, and identify cost-accuracy trade-offs. Early Stopping is applied to all Grid Search experiments.

2.5.3 Dynamic Learning Rate (DLR):

We used Cyclical Learning Rate (CLR) scheduler, which cyclically adjusts the learning rate between a minimum and a maximum value to accelerate convergence and avoid local minima.

3 Results

Table 1: Reduction in Training Cost and MSE from Training Adjustments

Adjustments	**Node Loss**	**Net Loss**	**Runtime**	**Memory**
ES[a]	4.9%	3.17%	84.37%	0%
ES[a]+ AA[b]	11.27%	3.65%	77.19%	38.83%
ES[a]+ AA[b]+ DLR[c]	6.4%	-17.6%	89.32%	38.83%

[*] Comparison is based against the baseline metrics
[a] Early Stopping
[b] Architecture Adjustment (Grid Search: 4 layers, 8 dimensions)
[c] Dynamic Learning Rate (Cyclical Learning Rate)

4 Conclusion

Our results demonstrate that DE-HNN is a highly expressive model with a fast convergence rate. We optimized DE-HNN to achieve maximum performance with minimal training using a simplified configuration **(4 layers, 8 dimensions)**, coupled with the scheduling of the cyclic learning rate and our custom Early Stopping condition.

It has less than a drop in average performance **6%** with as few as 11 training iterations, while achieving a significant improvement in both **runtime (89.32%)** and **memory (38.83%)**, compared to the baseline.

- DE-HNN performs better on the test set with fewer training iterations.
- Simpler configurations (e.g., 4 layers, 8 dimensions) outperform more complex ones (e.g., 4 layers, 32 dimensions).
- While simpler models require slightly more iterations to converge, they are more efficient in runtime and memory, since the additional number of training iterations is negligible.

- Cyclical Learning Rate accelerates convergence, allowing the Early Stopping condition to terminate training even earlier.

5 Contributions

- **Han Hoang**: Completed run-time profiling and analysis. Completed benchmarking for the baseline model. Completed test runs and environment setup for the experiments. Verified results of grid-search code run. Finalized report checkpoint and code checkpoint.
- **Joshua Li**: Ran smaller scale memory benchmark on local machine. Completed memory profiling. Completed and debugged code for grid-search. Completed draft for report checkpoint. Helped in environmental setup.

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." *arXiv preprint arXiv:2404.00477*

Yang, Shuwen, Zhihao Yang, Dong Li, Yingxue Zhang, Zhanguang Zhang, Guojie Song, and Jianye HAO. 2022. "Versatile Multi-stage Graph Neural Network for Circuit Representation." In *Advances in Neural Information Processing Systems*. [Link]

Appendices

A.1 Project Proposal

A.1.1 Problem Statement

Graph Neural Network (GNN) are computationally expensive, especially for large-scale graphs. In congestion prediction, GNNs are trained on circuit netlist representations containing billions of components, making training time a significant bottleneck and limits scalability and usefulness of the model in the chip design process. As chip designs become more complex, the need to optimize the runtime of GNN model training becomes increasingly important. Optimized model training runtime will result in a faster chip design cycle in which quicker adjustments can be made, ultimately accelerating the overall development and optimization of the chip.

A.1.2 Background Information

The DE-HNN model is a type of GNN that provides a good framework for optimizing chip design by capturing long-range interactions in dense graphs. However, De-HNN is computationally expensive due to the dual update of nodes and nets features and the addition of virtual nodes. The high number of message passing operations causes a significant bottleneck that limits the scalability of DE-HNN to larger and more complex graphs.

Previous work on the DE-HNN model has primarily emphasized enhancing predictive performance, often overlooking the critical issue of computational efficiency. However, in real-world applications, the value of a more accurate model must be weighed against the practical constraints of computational cost.

The goal is to improve runtime efficiency without compromising the predictive power of the model. Our quarter 2 project aims to achieve this goal by investigating specific modifications to the DE-HNN model. Key areas of exploration include:

- **Propagation Interval Optimization:** Change the frequency of propagation steps while maintaining the model's ability to capture long-range dependencies. Our intuition is that by decreasing the propagation interval, we lower our number of message passing operations and cut down computational training cost.

- **Partitioning Optimization:** Change the number of partitioned neighborhoods while preserving the expressiveness of the model. Our intuition is that by decreasing the number of graph partitions, we decrease the number of constructed virtual nodes and lower our number of message passing operations, resulting in lower computational training cost.

This investigation directly addresses a deficiency identified during the Q1 Project by focusing on computational efficiency rather than predictive performance. By building on the insights gained from the DE-HNN re-implementation and leveraging existing methodologies from GNN optimization, the objectives of our quarter 2 projects are:

- Enhance the scalability of DE-HNN for large, dense hypergraphs.
- Provide a framework for balancing accuracy and efficiency in hypergraph-based models.
- Establish a roadmap for adapting DE-HNN to real-world chip design applications, where computational constraints are a critical factor.

The proposed work not only fills a gap in prior research but also extends the applicability of DE-HNN, making it a more practical tool for congestion modeling in chip design. These advancements have the potential to impact other domains requiring scalable hypergraph neural networks.

A.1.3 Deliverables

The primary output will be a comprehensive report detailing the methods, experiments, and results of modifying the DE-HNN model. The report will include the following.

- **Problem Context:** An explanation of computational bottlenecks in the model.
- **Methodology:** Descriptions of the modifications and tests we performed on the DE-HNN model, including rationale and implementation specifics for each optimization we tried.
- **Benchmarking and Analysis:** An evaluation of the impact of these modifications on the model runtime, using average runtime as the primary performance metric. This will include comparison with the baseline DE-HNN implementation.

The report will also feature visualizations including graphs comparing the runtime performance of the baseline DE-HNN and the modified versions we will be testing, accuracy metrics to ensure that runtime optimizations do not significantly degrade the baseline model's performance, and graph statistics to analyze how structural properties may influence the effectiveness of the modifications.

A.2 De-HNN Model Architecture

A.2.1 Update Functions

One key difference between the De-HNN model architecture and other types of message-passing GNNs lie in the way it aggregates and updates cells and nets features.

Let a directed hypergraph $H = (V, \sum)$ where V is a set of nodes and $\sum$ is a set of hyperedges, the update function for a cell is given by:

$$m^{\ell}(v) = \sum_{\sigma' \in \mathscr{L}(v)} \mathrm{MLP}_1^{\ell}(M^{\ell-1}(\sigma'))$$

Where:

- $m^{\ell}(v)$ is the node feature of some ℓ GNN layer.

- MLP is a Multilayer perceptron.

- σ is a net in the set of nets $\sum$.

- $M^{\ell-1}$ is the net feature of the previous GNN layer.

- $\mathscr{L}(v)$ is the set of nets that node v is in.

This function denotes that a node feature gets updated by the aggregated sum of the MLP-transformed net features of the nets the node belongs to.

And the update function for a net is given by:

$$M^{\ell}(\sigma) = \mathrm{MLP}_3^{\ell}[m^{\ell}(v_{\sigma}) \bigoplus (\sum_{v' \in S_{\sigma}} \mathrm{MLP}_2^{\ell}(m^{\ell}(v')))]$$

Where:

- $M^{\ell}(\sigma)$ is the net feature of some ℓ GNN layer.

- MLP is a Multilayer perceptron.

- $m^{\ell}(v_{\sigma})$ is the driver cell of the net—cell that establishes incoming connections from other nets.

- $m^{\ell}(v_{\sigma})$ is the sink cell of the net—cell(s) other than the driver node from the same net.

- $\bigoplus$ is the vector concatenate operation.

This function denotes that a net feature gets updated by the MLP-transformed concatenate vector representation of the driver node features and the aggregated sum of all MLP-transformed features of the sink nodes.

Explanation. The cell features are first transformed by an MLP in a GNN layer, and then used to update the net features in the graph. De-HNN performs a dual update for both cells

and nets, with information passing through the GNN layers sequentially from cells to nets. This dual update mechanism is key to De-HNN's ability to capture long-range interactions effectively, as it dynamically adjusts the directed hypergraph properties for contextual relevance, enabling the model to learn distant node features with minimal information dilution.

A.2.2 Aggregated Nodes (Virtual Nodes)

A key difference between the De-HNN model and other message-passing GNNs is its use of aggregated or "virtual nodes". In De-HNN, the graph is first partitioned into k number of neighborhoods using the Metis partitioning method. For each neighborhood, a virtual node is created by aggregating the features of all the nodes within that neighborhood. Information is then propagated through these virtual nodes, rather than directly between all individual nodes in the graph.