DE-HNN: An effective neural model for Circuit Netlist representation

Zhishang Luo[1] **Truong Son Hy**[2] **Puoya Tabaghi**[1] **Donghyeon Koh**[4] **Michael Defferrard**[4]

Elahe Rezaei[3] **Ryan Carey**[3] **Rhett Davis**[5] **Rajeev Jain**[3] **Yusu Wang**[1]

[1] University of California San Diego [2] Indiana State University

[3] Qualcomm Technologies, Inc. [4] Qualcomm Wireless GmbH [5] North Carolina State University

Abstract

The run-time for optimization tools used in chip design has grown with the complexity of designs to the point where it can take several days to go through one design cycle which has become a bottleneck. Designers want fast tools that can quickly give feedback on a design. Using the input and output data of the tools from past designs, one can attempt to build a machine learning model that predicts the outcome of a design in significantly shorter time than running the tool. The accuracy of such models is affected by the representation of the design data, which is usually a netlist that describes the elements of the digital circuit and how they are connected. Graph representations for the netlist together with graph neural networks have been investigated for such models. However, the characteristics of netlists pose several challenges for existing graph learning frameworks, due to the large number of nodes and the importance of long-range interactions between nodes. To address these challenges, we represent the netlist as a directed hypergraph and propose a *Directional Equivariant Hypergraph Neural Network* (DE-HNN) for the effective learning of (directed) hypergraphs. Theoretically, we show that our DE-HNN can universally approximate any node or hyperedge based function that satisfies certain permutation equivariant and invariant properties natural for directed hypergraphs. We compare the proposed DE-HNN with several State-of-the-art (SOTA) machine learning models for (hyper)graphs and netlists, and show that the DE-HNN significantly outperforms them in predicting the outcome of optimized place-and-route tools directly from the input netlists. Our source code and the netlists data used are publicly available at `https://github.com/YusuLab/chips.git`.

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

1 Introduction

Chip design is a complicated process involving numerous steps, many of which involve solving hard optimization problems. Just consider the stage of the *place and route* of a synthesized netlist: Here the input is a netlist consisting of *cells* and *nets*, where cells refer to functional units such as logic gates, and nets refer to connections between cells. The goal is to produce a layout of this netlist in a specific 2D region, where gates are placed and connections among them are realized by wires laid out across multiple layers (called "routed"), all while aiming to optimize multiple key properties (e.g, minimizing total wirelength and reducing congested "hotspots"). This place-and-route stage is highly nontrivial to solve for large netlists, and requires a time-consuming process in practice with multiple stages and iterations.

There therefore arises the need for data-driven methods to predict properties of a design directly without the time-consuming place and routing process. To this end, graph neural networks become natural choices, given that the netlists are often represented as a graph or a hypergraph. In this paper, we aim to develop an efficient and effective graph learning architecture to predict post-routing properties (e.g., wirelength or congestion) for a synthesized netlist accurately.

The past decade has witnessed a tremendous growth

of graph learning models. Two most popular families are (i) message-passing neural networks (MPNNs) (Gilmer et al., 2017; Jegelka, 2022), and (ii) transformer based approaches (e.g, survey (Müller et al., 2023)). However, netlists present several challenges for existing graph learning architectures: (i) their size can be massive, from hundreds of thousands to millions of nets, (ii) long-range interactions are important (e.g., properties of interest might be caused by long paths and other long-range interactions), and (iii) properties of post-routing netlists seem to depend on complex information of graph topology beyond simple statistics such as in-/out-degrees (distributions).

Unfortunately, it is challenging for popular MPNNs to capture long-range interactions, due to issues such as over-smoothing of graph signals (Chen et al., 2020a) and oversquashing bottlenecks (Topping et al., 2022). MPNN's ability in capturing graph motifs (e.g cycles and trees) and higher-order structures is also limited (Xu et al., 2019; Jegelka, 2022; Hy et al., 2019). Transformer-based graph neural models appear to be more effective in capturing long-range interactions (Dwivedi et al., 2022; Ngo et al., 2023). However, each transformer layer typically takes time quadratic to the number of nodes. While there are sparse transformers (Katharopoulos et al., 2020; Tay et al., 2022), their representation power is also reduced. Furthermore, the primary ways for a graph transformer to capture input graph topology have been either via initial position/structure encoding of nodes, or via the use of certain pairwise graph distances to "reweight" the attention (Zhang et al., 2023). In general, it is not clear how sensitive a transformer based model is to features in input graph topology (e.g., specific paths which can be important to netlists properties).

Our work. In this paper, similar to (Xie et al., 2021), we model a netlist as a directed hypergraph and present a novel *Directional Equivariant Hypergraph Neural Networks* (DE-HNN) for the effective learning of (directed) hypergraphs. In particular, DE-HNN can be used to predict properties (e.g., congestion or netwirelength) of a post-routed netlist directly from an input netlist **before** performing the lengthy place-and-route process. Our DE-HNN incorporates several new ideas to address the aforementioned challenges posed by netlists. Our contributions are as follows:

- We advocate for the modeling of a netlist as a directed hypergraph. Indeed, a net usualy consists of a driver gate/cell c, togehter with a set S of "sinks"; see Section 2 and Figure 1. Recognizing the difference between the driver gate and sinks in the timing of a routed net, inspired by (Xie et al., 2021), we represent a net as a *directed-hyperedge* (c, S) to separate the roles of driver and sinks cells.
- We propose a learning model DE-HNN for directed hypergraphs. Theoretically, we show (Theorem 1) that our DE-HNN can universally approximate any node or hyperedge based function that satisfies certain permutation equivariant and invariant properties natural for directed hypergraphs.
- On the practical front, to mitigate the issue of large size of and long-range interactions in input netlists, we use a hierarchy of virtual nodes (VNs), which provides additional "bridges" to allow the integration of both local and global information while still maintaining original graph topology (unless in a graph pooling approach); see Figure 2 and Section 3.
- To make our initial node features more informative, in addition to using Laplacian eigenvectors to provide position encoding as in (Kim et al., 2022; Rampášek et al., 2022), we also use a topological summary called persistent homology (Edelsbrunner and Harer, 2010; Dey and Wang, 2022), which can be used to encode the "shape" of graph motif **around each node** in a multi-scale manner (e.g., (Zhao et al., 2020; Yan et al., 2021)).
- We compare our DE-HNN with several SOTA machine learning models for (hyper)graphs and netlists. Our model significantly outperforms them in predicting different properties of post-routed netlists directly from input netlists. We provide careful ablation studies to demosntrate the utilities of the use of directed hyperedge, (hierarchical) VNs and persistence-based topological summaries. Finally, we remark that ML research for chip design currently suffers from the scarcity of open-source benchmark datasets[1]. We hope our datasets (which will be made publicly available at `https://github.com/YusuLab/chips.git`) can help bridge this gap. These netlists (of sizes from 400K to 1.3M) can also serve as benchmark for long-range graph interactions for machine learning researchers.

While in this paper, we design DE-HNN with the goal of netlists representation and learning, our architecture as well as its theoretical results are general and applicable to any directed hypergraphs.

Related work on machine learning models for netlists. Earlier machine learning (ML) approaches for netlist property (e.g, congestion) prediction assume that the placement of cells (logic gates) are already

[1]Previous netlist property prediction work sometimes releases the input netlist designs, which our paper also uses. However, they do not release the resulting properties nor the post place-and-route netlists due to the use of commercial tools.

given, that is, the input are placed, but not yet routed. They then convert the input to a 2D image or other 2D grid-based representations to predict the final congestion map over this region using models such as convolutional neural networks (Tabrizi et al., 2018; Chen et al., 2020b; Xie et al., 2018; Al-Hyari et al., 2021; Liang et al., 2020; Chai et al., 2023; Alawieh et al., 2020; Chen et al., 2022; Wang et al., 2022). However, the placement information is itself very time-consuming to obtain. Furthermore, representing the input as a 2D image makes it hard to capture local and global connectivity information in the netlists.

Since a circuit is represented more accurately as a (hyper)graph, recent work deploys graph neural networks (GNNs) for congestion prediction. The work of (Kirby et al., 2019) constructs a homogeneous graph representation of netlist in which each node corresponds to a cell, and if two cells are connected by a net then there exists an edge between those nodes in the graph, and then applies Graph Attention Networks (GAT) (Veličković et al., 2018). Later follow-up work includes using node embedding computed from partitioned subgraph (to capture more global graph structure) (Ghose et al., 2021) and using dual graph with both cell and net features (Xie et al., 2021). Note that this convertion of a net to a clique can lead to very large sized cliques, and also loses net-specific topological information. The SOTA approach for netlists representation is (Yang et al., 2022), which introduces a heterogeneous (i.e., different edge types) graph construction, called *circuit graph*, in which both cells and nets are represented as nodes of a bipartite graph.

All these approaches still have the issues of capturing long-range interaction essentially for netlists. As we describe in "Our contributions", our DE-HNN deploys a suitable architecture, hierarchical virtual nodes, as well as informative persistent homology features, to build an effective graph learning model for netlists (and other directed hypergraphs). Note that similar to (Kirby et al., 2019; Yang et al., 2022), our DE-HNN performs learning on netlists **without** placement information. If placement informtion is available, they can easily be added to initial node position encoding.

2 Modeling netlists as directed hypergraphs

Circuit netlists. A circuit netlist is a textual representation of electronic components, such as logical boolean gates, and the connections between them. A (pre-placed, also called synthesized) netlist $\mathcal{H}$ consists of a collection of **cells** (logic gates) $\mathcal{C} = \{\mathsf{c}_1, \ldots, \mathsf{c}_n\}$, and a set of **nets** $\mathcal{N} = \{\sigma_1, \ldots, \sigma_m\}$. Each cell (gate) has a certain number of input pins and an output pin. The number of input pins is decided by the type of this gate (e.g., an AND gate takes two inputs). For every gate, its output will flow into the input pins of a collection of other gates. This information is captured by the concept of **net**, where a net $\sigma \in \mathcal{N}$ consists of the output pin of a certain cell, called its *driver cell* denoted by $\mathsf{v}_\sigma \in \mathcal{C}$, together with all those *sink cells*, denoted by $\mathsf{S}_\sigma \subseteq \mathcal{C}$, where the signal from this output pin will flow into. In other words, a net can be represented by a tuple $\sigma = (\mathsf{v}_\sigma, \mathsf{S}_\sigma)$. See Figure 1 (a).

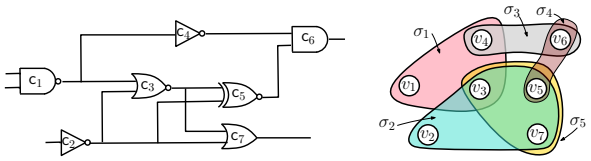
Given such a netlist, standard chip design pipelines will first lay it out in the physical space (*placement*). Then in the *routing stage*, the connection from output pin of one cell to the input of other cells are mapped to the routing channels within the chip's physical floorplan. Among other properties, one wishes to minimize the total wirelength of each net, and to reduce routing "congestions", which occurs when the number of edges to be routed in a specific region of the floorplan exceeds the available routing capacity. See Figure 4 in the Supplement for an illustration of a placed-and-routed netlist.

Directed hypergraphs. The standard *hypergraph* H is a tuple (V, Σ), where V is a set of nodes, Σ is a set of *hyperedge*, and each $e \in \Sigma$ is a subset of V; i.e., $e \subseteq V$. A *directed hypergraph* $\overrightarrow{H}$ is a tuple $(V, \overrightarrow{\Sigma})$, where each *directed hyperedge* $\sigma \in \overrightarrow{\Sigma}$ consists of an ordered pair $\sigma = (v_\sigma, S_\sigma)$ with $v_\sigma \in V$ and $S_\sigma \subseteq V$. It is easy to see that a netlist $(\mathcal{C}, \mathcal{N})$ can be naturally viewed as a directed hypergraph where we have: **cell** $\Leftrightarrow$ **node**, and **net** $\Leftrightarrow$ **directed hyperedge**. See Figure 1 (b).

In what follows, we often use cells / nodes, as well as nets / directed hyperedges, interchangeably. In fact, for simplicity, we often use the terms "**nodes**" and "**nets**" as they are more concise. We will also refer to v_σ and S_σ from a directed hyperedge $\sigma = (\mathsf{v}_\sigma, \mathsf{S}_\sigma)$ as its *driver* and its *sinks*, respectively, just like in a net σ.

Given a net $\sigma \in \overrightarrow{\Sigma}$, we say that it *contains* a node $v \in V$, denoted by $v \in \sigma$, if v is either the driver, or a sink of σ. Given a node $v \in V$, we say that a net σ is *incident on* v if σ contains v. The collection of nets incident to v is called the *incident-net-set* of v, denoted by $\mathcal{I}(v) = \{\sigma' \in \overrightarrow{\Sigma} \mid v \in \sigma'\}$. For example, in Figure 1 (b), the incident-net-set of v_3 is $\mathcal{I}(v_3) = \{\sigma_1, \sigma_2, \sigma_5\}$.

In the chip design literature, a netlist is oftentimes represented either (1) as a (directed) graph where two cells are connected if they belong to a common net; or (2) as a standard hypergraph where a net is a hyperedge consisting of the union of the driver cell and all sink cells. The former can lead to huge cliques (as some nets can consist of large number of cells) and also lose



(a) (b)

Figure 1: (a) A netlist with 7 cells $\mathcal{C} = \{\mathsf{c}_1, \ldots, \mathsf{c}_7\}$ and 5 nets. For example, the output of gate c_2 flows into cells c_3, c_5, and c_7, giving rise to the net $\sigma = (c_2, \{c_3, c_5, c_7\})$. That is, the driver cell of σ is $\mathsf{v}_\sigma = c_2$, while its sink-set being $\mathsf{S}_\sigma = \{c_3, c_5, c_7\}$. (b) The corresponding directed hypergraph $\overrightarrow{H} = (V, \overrightarrow{\Sigma})$ with 7 nodes and 5 hyperedges $\overrightarrow{\Sigma} = \{\sigma_1, \ldots, \sigma_5\}$. Each node v_i corresponds to cell c_i, and each hyperedge is marked as a shaded region.

sensitivity to the net topology. In the learning context, the work of Xie et al. (2021) first separated the role of the driver and sinks of a net in the representation of a netlist and provided justification for this choice. Their final representation is still a graph representation that is intuitively a directed version of the so-called *line graph* for a hypergraph. However, converting a hypergraph to a line graph is a lossy process. The directed hypergraph provides a more informative and cleaner representation: it both preserves the full net information and differentiates between drivers and sinks.

3 DE-HNN: a neural network for directed hypergraphs

In this section, we first describe a basic neural network (NN) model in Section 3.1, which we refer to as base-DE-HNN, for the representation learning of directed hypergraphs. We provide theoretical justification of this model in Section 3.2. Then in Section 3.3, we describe how to augment this base model to make it more effective at capturing long-range interactions as well as the multi-scale graph topology.

3.1 Base-DE-HNN

Our base-DE-HNN uses message-passing mechanisms. However, it differs from standard MPNN (message-passing neural networks) (Gilmer et al., 2017) in how the messages are aggregated and updated, so as to process node and net based features, as well as to respect the direction of hyperedges.

More specifically, base-DE-HNN consists of L layers. Consider an input directed hypergraph $\overrightarrow{H} = (V, \overrightarrow{\Sigma})$. For the ℓ-th layer, each node $v \in V$ (resp. each net/directed hyperedge $\sigma \in \overrightarrow{\Sigma}$) will maintain a node feature (resp. net feature) denoted as $m^\ell(v)$ (resp. $M^\ell(\sigma)$). For simplicity, assume that $m^\ell(v), M^\ell(\sigma) \in \mathbb{R}^{d_\ell}$ are d_ℓ-dimensional vectors. Assume first that our final goal is to predict net properties. The base-DE-HNN will compute $m^\ell(v)$ and $M^\ell(\sigma)$ using feature representations from the $(\ell-1)$-th layers by the following two steps:

[Node Update]: First, the features of each **node** (cell) $v \in V$ is updated using features of the set of those **nets** containing it, that is, via the features of those nets in the incident-net-set $\mathcal{I}(v)$ of v as follows:

$$m^\ell(v) = \mathrm{Agg}^\ell_{\sigma \to v}(\{\{M^{\ell-1}(\sigma')\}\}_{\sigma' \in \mathcal{I}(v)}), \qquad (1)$$

where $\{\{\cdot\}\}$ denotes a *multiset* as some neighboring nets could have identical feature representations. The function $\mathrm{Agg}_{\sigma \to v}$ operates on a multiset and should be *invariant* to the order of neighboring nets of v in $\mathcal{I}(v)$. Similar to the Deep Set architecture (Zaheer et al., 2017) which can hancle such *permutation invariance* in multisets, we implement the function $\mathrm{Agg}_{\sigma \to v}$ by:

$$m^\ell(v) = \sum_{\sigma' \in \mathcal{I}(v)} \mathrm{MLP}^\ell_1\big(M^{\ell-1}(\sigma')\big), \qquad (2)$$

where MLP_1 stands for a multi-layer perceptron. For example, the update of node feature for v_4 in Figure 1 (b) is $m^\ell(v_4) = \mathrm{MLP}^\ell_1\big(M^{\ell-1}(\sigma_1)\big) + \mathrm{MLP}^\ell_1\big(M^{\ell-1}(\sigma_3)\big)$ as $\mathcal{I}(v_4) = \{\sigma_1, \sigma_3\}$. It is easy to see that the update in Eqn (2) satisfies the needed permutation invariance.

[Net Update]: Next, the features of each **net** $\sigma = (\mathsf{v}_\sigma, \mathsf{S}_\sigma)$ is updated using the new node features for those **nodes** contained in σ. Since the net (hyperedge) σ is directed, we wish to separate the roles of its driver node v_σ and the set of sinks S_σ, that is,

$$M^\ell(\sigma) = \mathrm{Agg}^\ell_{v \to \sigma}(m^\ell(\mathsf{v}_\sigma), \{\{m^\ell(v')\}\}_{v' \in \mathsf{S}_\sigma}) \qquad (3)$$

However, the update should **not** depend on the ordering of nodes in the sink set S_σ, i.e., $\mathrm{Agg}_{v \to \sigma}$ needs to be *permutation invariant* w.r.t. its second parameter, the multiset $\{\{m^\ell(v')\}\}_{v' \in \mathsf{S}(\sigma)}$. We use the following to implement Eqn (3) to guarantee the needed permutation invariance of $\mathrm{Agg}_{v \to \sigma}$ w.r.t. the ordering of nodes in S_σ:

$$M^\ell(\sigma) = \mathrm{MLP}^\ell_3\Big[m^\ell(\mathsf{v}_\sigma) \oplus \Big(\sum_{v' \in \mathsf{S}_\sigma} \mathrm{MLP}^\ell_2(m^\ell(v'))\Big)\Big] \qquad (4)$$

where MLP_2 and MLP_3 are multi-layer perceptrons, and $\oplus$ denotes vector concatenation. For example, in Figure 1, the update of $\sigma_1 = (v_1, \{v_3, v_4\})$ is $M^\ell(\sigma_1) = \mathrm{MLP}^\ell_3\big(m^\ell(v_1) \oplus (\mathrm{MLP}^\ell_2(m^\ell(v_3)) + \mathrm{MLP}^\ell_2(m^\ell(v_4)))\big)$.

We note that if the target task is to predict node-level features (instead of net-level features), then we will

apply dual update rules where the roles of nets and nodes will be swapped. Finally, in our implementation, we also add residuals (i.e, node / net features from the previous level) to each node / net features during the updates. There are L such layers, and in the end, if the given task is a regression task, then another linear layer ψ is applied to the values $m^L(\cdot)$ (resp. $M^L(\cdot)$) to generate the final node-based (resp. net-based) regression function. During the training process, the loss function in this case is the standard mean squared error; that is, if it is a node-based regression with ground-true function $\mathrm{y} : V \to \mathbb{R}$, then we have:

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{|V|} \sum_{v \in V} [\psi(m^L(v)) - \mathrm{y}(v)]^2, \tag{5}$$

where $\boldsymbol{\theta}$ denotes the set of all learnable parameters in the entire base-DE-HNN model. Node or net classification tasks are handled similarly but with the cross-entropy loss.

3.2 Theoretical analysis of DE-HNN

We now provide some theoretical guarantee of our base-DE-HNN model. For simplicity, in what follows we assume that our target (regression) functions are net-valued functions[2] (or simply *net-functions*) of the form $F : \overrightarrow{\Sigma} \to \mathbb{R}$. Our main result below holds for node-valued functions via a symmetric argument.

Let us consider one update stage at a fixed layer $\ell \in [1, L]$. From the net-function perspective, the goal of the update stage is the following: At the beginning of this stage, we start with input net features $\mu_\sigma := M^{\ell-1}(\sigma)$, for all $\sigma \in \overrightarrow{\Sigma}$. In the end, we obtain a set of new features $\mu^*_\sigma := M^\ell(\sigma)$ for each $\sigma \in \overrightarrow{\Sigma}$. If we view this feature update as a function on net features, then ideally, for any fixed $\sigma \in \overrightarrow{\Sigma}$, its new feature should depend on the features of all its *neighboring nets* which are those nets that share at least one node with σ. However, note that the neighbors of σ are naturally classified to two families:

- **(type-A)** those in the set $\mathcal{I}(\mathsf{v}_\sigma)$ (recall $\mathcal{I}(v)$ is the set of nets that contains node v), which are connected to σ via the driver node v_σ of σ; and
- **(type-B)** those in $\bigcup_{v' \in \mathsf{S}_\sigma} \mathcal{I}(v')$, where each set $\mathcal{I}(v')$ are those nets connected to σ via a sink node v' of σ (i.e., $v' \in \mathsf{S}_\sigma$). Note that $\Big\{\mathcal{I}(v')\Big\}_{v' \in \mathsf{S}_\sigma}$ is **a set of sets** of neighboring (type-B) nets of σ.

For example, in Figure 1, for $\sigma_5 = \{v_3, \{v_5, v_7\}\}$, its (type-A) neighbors are $\mathcal{I}(v_3) = \{\sigma_1, \sigma_2, \sigma_5\}$, while (type-B) is a set of sets $\{\{\sigma_2, \sigma_4, \sigma_5\}, \{\sigma_2, \sigma_5\}\}$. It is natural to model the desired update function for the directed hyper-edges as follows, which differentiate (type-A) and (type-B) neighbors of σ:

$$\mu^*_\sigma = \mathcal{F}\Big(\{\{\mu_{\sigma'}\}\}_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)}, \Big\{\Big\{\{\{\mu_{\sigma'}\}\}_{\sigma' \in \mathcal{I}(v')}\Big\}\Big\}_{v' \in \mathsf{S}_\sigma}\Big) \tag{6}$$

Note that the first variable, $\{\{\mu_{\sigma'}\}\}_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)}$, consists of the set of input feature representations of (type-A) neighbors of σ, while the second variable, $\Big\{\Big\{\{\{\mu_{\sigma'}\}\}_{\sigma' \in \mathcal{I}(v')}\Big\}\Big\}_{v' \in \mathsf{S}_\sigma}$, is **a set of sets**[3] **(of net features)**, constituting multisets of feature representations of those (type-B) neighbors of σ. The function $\mathcal{F}$ should be invariant not only to the order within each individual set $\mathcal{I}(u)$ for some cell u, but also invariant to the order of nodes in the sink-set S_σ, i.e., the order of sets $\{\mu_{\sigma'}\}_{\sigma' \in \mathcal{I}(v')}$'s within the outer-set in the second variable of $\mathcal{F}$. We refer to such a function $\mathcal{F}$ as **nested-permutation invariant**.

Now recall that in our base-DE-HNN, at each layer we perform update of node-feature by features of its incident nets as described in Eqn (2), followed by the update of net-features by the new features of those nodes contained in σ as specified in Eqn (4). In other words, one can think of the update of net features from $M^{\ell-1}(\cdot)$ to $M^\ell(\cdot)$ to be the composition of updates in Eqns (2) and (4). By construction, it is easy to see that the composition of these two udpate steps indeed gives rise to a nested-permutation invariant function. However, can any nested-permutation invariant net-function be represented (or approximated) by such a composition of two steps? A priori, the answer to this opposite direction is not clear at all as the iterative updates factored through the node features appears more restrictive. Our main theorem below shows that the answer is in fact positive.

Theorem 1 (Simplified)**.** *Let $\mathcal{F}$ be any continuous, nested-permutation invariant, net-value function as in Eqn (6). For simplicity, assume both input nets and output of M take values in a compact set $\mathcal{B} \subset \mathbb{R}^d$, a connected compact subset of $\mathbb{R}^d$. Then we have that $\mathcal{F}$ can be rewritten as the following sum-decomposition*

$$\begin{aligned} \forall \sigma: \ & \mathcal{F}\Big(\{\mu_{\sigma'}\}_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)}, \Big\{\{\mu_{\sigma'}\}_{\sigma' \in \mathcal{I}(v')}\Big\}_{v' \in \mathsf{S}_\sigma}\Big) \\ & = \rho\Big(\sum_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu_{\sigma'}), \sum_{v' \in \mathsf{S}_\sigma} \phi_2\Big(\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\mu_{\sigma'})\Big)\Big) \end{aligned} \tag{7}$$

where $\phi_1 : \mathbb{R}^d \to \mathbb{R}^{d'}$, $\phi_2 : \mathbb{R}^{d'} \to \mathbb{R}^{d''}$, and ρ are continuous functions.

[2] The net-valued function is more natural for properties such as net wirelength and congestion.

[3] For concision, we use "set" instead of "multiset" here.

Recall that for the ℓ-th layer, the input feature for any net σ' is $\mu_\sigma = M^{\ell-1}(\sigma')$. Now compare the right-hand side of Eqn (7) with Eqns (2) and (4): it is easy to see that by using MLP_1 from Eqn (2) to approximate the continuous function ϕ_1, and using MLP_2 and MLP_3 from Eqn (4) to approximate continuous functions ϕ_2 and ρ respectively, we then have that our iterative updates using Eqns (2) and (4) approximate any desired update of the net features via a nested-permutation invariant function as in Eqn (6). In other words, the iterative message-passing update steps in our base-DE-HNN provides an universal approximation of the desired nested-permutation invariant update functions over nets. The proof of this theorem and more discussions can be found in the Supplement.

3.3 Augmenting base-DE-HNN to DE-HNN

The base-DE-HNN provides an effective way to update features for both nodes and hyperedges in an iterative manner. We now describe further augmentation strategies for the resulting DE-HNN to capture long-range interactions as well as to be more senstive to the multi-scale graph topology.

Hierarchy of virtual nodes. The standard message-passing GNNs have difficulty to capture long-range interaction due to issues such as over-smoothing, over-squashing and under-reaching. Most GNNs used in practice have few layers. Transformer-type models for graphs may alleviate the issue (Dwivedi et al., 2022), but each self-attention layer requires quadratic computation, which does not scale to large graphs. Furthermore, it is not clear how to effectively encode graph structures for transformers, even with the use of initial position encoding (Müller et al., 2023) or reweighting based on graph distances (Zhang et al., 2023). Intuitively, we want to keep the simple and efficient message-passing framework over the input (very sparse) hypergraph, but also have a way to propagate long-range information among nodes that are far away from each other (in terms of graph distances). We will do so via the use of *virtual nodes*.

Specifically, a *virtual node (VN)* is an additional node we add that is connected to all input nodes. This effectively reduces the graph diameter to 2, and has been a popular strategy in graph learning literature; e.g. (Gilmer et al., 2017). However, note that messages cannot be directly passed between two nodes. Instead, information of all nodes have to be aggregated at the virtual node before being passed back to all nodes. Nevertheless, (Cai et al., 2023) shows that a message-passing GNN augmented by a single VN can already approximate Lineaer Transformer (Katharopoulos et al., 2020) and Performer (Choromanski et al., 2021) (as well as general transformers to some extent) and bring significant empirical gains.

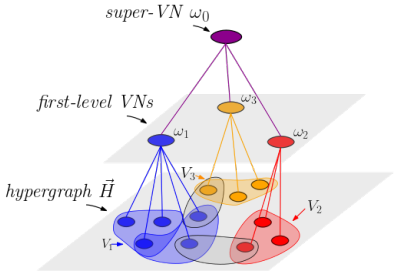


Figure 2: A two-level hierarchy of virtual nodes (VNs).

While adding a single VN will only linearly increase the number of edges, the VN itself will have a high degree and potentially become a computatoinal bottleneck. Furthermore, since the features of all nodes have to be aggregated at the virtual node, the aggregated messages will lose sensitivity to individual node features. The benefits of adding a single VN thus diminishes as the graph becomes larger. We instead propose to use a **hierarchy of VNs** as follows (see Figure 2).

- Given an input hypergraph $\overrightarrow{H} = (V, \overrightarrow{\Sigma})$, we first use Metis (Karypis and Kumar, 1998) to partition its node set to k disjoint subsets $V = V_1 \cup V_2 \cup \cdots \cup V_k$. In our experiments, we keep the sizes of V_is roughly balanced, and therefore the value of k varies as input graph size changes.
- We introduce a VN ω_i for each subset V_i, for $i \in [1, k]$, and ω_i is connected to all nodes in V_i. These VNs are *first-level VNs*. Note that the total number of edges added this way is only $n = |V|$.
- We further create a *super-VN*, denoted by ω_0, which connects to all the first-level VNs. This introduce an additional k number of edges.

In what follows, we refer to nodes from input hypergraphs as *standard nodes*. During the updating of node/edge features, we will use *heterogeneous updates*, where the aggregation functions at the first-level VNs and super-VN are different from that at standard nodes. Different from graph pooling, the use of additional VNs keep the original hypergraph topology, while they act as additional "bridges" and allow information flow both at global (among far-apart nodes via VNs) and local scales (along original edges).

As the input hypergraph becomes even larger, we could use multiple layers in our hierarchy of VNs. Nevertheless, we observe that two layers already yield good performance in our current test cases. If nodes are placed, one could partition the node set by spatial locations (e.g., by decomposing the rectangular region) instead of using Metis.

Initial positional / structural encodings. Finally, we aim to encode meaningful multi-scale features for each node / hyperedge. To this end, for each node, we add the Laplacian positional encoding consisting of the function value of this node from the first s ($s = 10$ in our experiments) eigenvectors of the undirected version of input hypergraph.

Furthermore, to better capture the "shape" of the neighborhood of each node v in a more discriminative and multi-scale manner, similar to (Yan et al., 2021; Zhao et al., 2020), we use the so-called extended persistence diagram (PD) summary induced by the shortest path distance function within the r-hop neighborhood of each node ($r = 6$ in our experiments) as an initial *structural encoding* for each node. Note that it is known (Tian and Wang, 2019) that PDs constructed this way can encode rich graph information of the r-hop neighborhood of each node v (viewed as a local motif around v), such as clustering coefficients, number of nodes at distance $a \leq r$ to v, number of independent cycles, etc. See the Supplement for more details. Indeed, as our ablation study and results in Supplement show, adding PDs improve *both our and previous graph learning models.*

4 Experiments

4.1 Datasets

We used 12 of the Superblue circuits from (Viswanathan et al., 2011, 2012) to evaluate our proposed models and baselines. The Superblue circuits are some of the most complex yet publicly available circuits used in previous work about VLSI placement and routing. The size of these netlists range from 400K to 1.3M nodes, with similar number of nets. Note that these hypergraphs are very sparse, although there are a very small fraction of nets with large sizes. See the Supplement for more detailed statistics of these netlists, including the size of each design as well as the statistics of target properties.

We pulled our designs from the RosettaStone (Kahng et al., 2022) repository's Skywater 130 nm technology (Edwards, 2020) benchmark set. We then used a commercial physical design tool to perform placement optimization for each design with a utilization ratio of 0.7. We exported global-routing congestion information in the form of demand and capacity for each global-route cell (GRC) in each routing layer.

4.2 Setup

Baselines. We compare with a range of SOTA baselines: **(i)** Graph Convolutional Networks (GCN) (Kipf and Welling, 2017); **(ii)** a SOTA variant of Graph Attention Networks (GATv2) (Brody et al., 2022); **(iii)** Hypergraphs Message Passing Neural Network (HMPNN) (Heydari and Livi, 2022); **(iv)** Hypergraph (Neural) Networks with Hyperedge Neurons (HNHN) (Dong et al., 2020); **(v)** A multiset function framework for Hypergraph Neural Network (AllSet) (Chien et al., 2022); **(v)** the SOTA graph-based model specifically defined for netlist property predictions, NetlistGNN (Yang et al., 2022). See the Supplement for more details on their architecture, including model complexity. We also compare with the hypergraph convolutional operator (HyperConv) (Bai et al., 2021) and Linear Transformers (Katharopoulos et al., 2020). As both models' performances on average are not as good as other baseline models, we include their results only in the Supplement.

We compare these baselines with our DE-HNN model, which we refer to as full-DE-HNN in results to differentiate with base-DE-HNN model, whose performance we also include for reference. Note that base-DE-HNN is **without** persistence diagrams (PDs) as initial features **nor** virtual nodes (VNs). We implement our DE-HNN and the baselines with PyTorch (Paszke et al., 2019) and PyTorch-geometric (Fey and Lenssen, 2019).

Features. For each cell, its initial features include: type, width, height of gate, degree, degree distribution of a local neighborhood (summarized into a vector), top-10 Laplacian eigenvectors, and persistence diagram (PD) features (see Supplement). For all methods other than Linear Transformer, the initial net features are the nets' degrees. For Linear Transformer, net features are obtained by averaging the features of those nodes contained in it so as to provide more topology information. (see Supplement). For a fair comparison, the same initial cell/net features are used for all models (other than our base-DE-HNN, which do not have PDs as it is a base model without any augmentation). One might wonder whether adding persistence diagrams (PDs) features are beneficial for both our method and baselines. Indeed as we show later in Ablation study and in Supplement, using PDs improve model performance for **both our methods and baselines**. For example, it improves NetlistGNN by 3.6% on average in terms of demand regression in the single-design setting.

Prediction tasks. We test three different tasks in our experiments, including both net- and node-based tasks. These tasks cover the types of experiments performed by previous netlist prediction approaches.

- **Net-based wirelength regression**: We use half-perimeter wirelength (HPWL), a common estimator used for wirelength calculation, to calcu-

Table 1: Net-based wirelength regression. Last row "Improvement" refers to the improvement of our full DE-HNN model over the best baseline approach for each metric.

Model	Single-Design			Cross-Design		
	RMSE ↓	MAE ↓	Pearson ↑	RMSE ↓	MAE ↓	Pearson ↑
GCN	1.762	1.276	0.750	1.691	1.276	0.746
GATv2	1.812	1.330	0.687	1.717	1.281	0.737
AllSet	1.718	1.264	0.760	1.837	1.348	0.695
HMPNN	1.841	1.368	0.710	1.785	1.335	0.710
HNHN	1.852	1.368	0.717	1.754	1.333	0.701
NetlistGNN	1.773	1.320	0.740	1.762	1.324	0.718
base DE-HNN	1.751	1.269	0.748	1.731	1.291	0.730
full DE-HNN	**1.689**	**1.245**	**0.770**	**1.677**	**1.242**	**0.754**
Improvement	**1.7%**	**1.6%**	**1.3%**	**1.9%**	**2.6%**	**1.8%**

Table 2: Net-based demand regression for each design.

Model	Single-Design			Cross-Design		
	RMSE ↓	MAE ↓	Pearson ↑	RMSE ↓	MAE ↓	Pearson ↑
GCN	9.321	6.163	0.570	6.571	5.024	0.365
GATv2	9.342	6.118	0.561	6.623	5.137	0.363
AllSet	9.072	5.745	0.632	6.120	4.820	0.345
HMPNN	9.342	6.118	0.561	6.979	5.356	0.306
HNHN	9.342	6.118	0.561	6.390	4.870	0.358
NetlistGNN	9.063	5.839	0.623	8.328	6.839	0.367
base DE-HNN	8.997	5.764	0.630	6.778	5.085	0.337
full DE-HNN	**8.381**	**5.334**	**0.683**	**6.037**	**4.670**	**0.372**
Improvement	**7.5%**	**7.2%**	**8.1%**	**1.4%**	**4.1%**	**1.4%**

late the wirelength of each net (Mirhoseini et al., 2021). Similar to (Yang et al., 2022), we take the $\log_2$ of the wirelength to reduce the range.

- **Net-based demand regression**: We predict the net-based *demand*. Congestion happens when demands exceeds capacity. There is no concensus on how to define congestion (difference or ratio) and we thus directly predict demand.
- **Cell-based congestion classification**: Similar to (Yang et al., 2022) and (Wang et al., 2022), we classify the cell-based congestion values (computed as the ratio of demand/capacity) into (a) [0, 0.9], *not-congested*; and (b) [0.9, inf]; *congested*.

Our experiments have two settings:

- **Single-design:** We train and evaluate on each individual design separately. For each graph in a design, we apply 4-fold cross-validation to randomly split the nodes into 4 subsamples with same sizes (25%/25%/25%/25%). We report the average performance across all 12 designs with cross-validation applied to each design. The distribution of target values and the average performance from cross-validation for each design can be found in the Supplement.
- **Cross-design:** We aim to evaluate the ability of the models to generalize to unseen netlist topologies. Following previous work (Yang et al., 2022), we use 10 designs, Superblue1,2,3,5,6,7,9,11,14,16, for training, superblue18 for validation, and superblue19 for testing.

Table 3: Cell-based congestion classification.

Model	Single-Design			Cross-Design		
	Precision ↑	Recall ↑	F_score ↑	Precision ↑	Recall ↑	F_score ↑
GCN	0.761	0.857	0.802	0.633	0.997	0.773
GATv2	0.810	0.864	0.835	0.630	**0.999**	0.765
AllSet	0.782	0.837	0.804	0.645	0.964	0.773
HMPNN	0.774	0.826	0.792	0.633	**0.999**	0.772
HNHN	0.792	0.869	0.826	0.648	0.939	0.767
NetlistGNN	0.812	0.860	0.831	0.647	0.953	0.771
base DE-HNN	0.824	0.860	0.840	0.653	0.990	0.774
full DE-HNN	**0.833**	**0.876**	**0.853**	**0.660**	0.986	**0.780**
Improvement	**2.6%**	**0.8%**	**2.2%**	**1.7%**	—	**1.0%**

4.3 Results

The test performance of all baselines and our methods (base-DE-HNN and full-DE-HNN) are shown in Tables 1, 2 and 3. For the regression tasks, to be comprehensive, we use three metrics: the Mean Squared Error (MSE), the Mean Average Error (MAE), and the Pearson correlation (Pearson). For classification tasks, we report the Precision, Recall, and F-Score. Note that for MSE and MAE, the smaller the value is the better; while for Pearson, Precision, Recall, and F-Score, the larger the better. In all tables, we highlight the best performance results in **red**, while the best among all baseline models are colored **blue** (unless a baseline result is the best, in which case it will be red-colored). In the last row of these tables, we show the **Improvement** of our full-DE-HNN model over the **best of all baselines** for each metric; note that our improvement over any individual baseline can only be better than this improvement. Our full-DE-HNN model outperforms **all** baselines, sometimes significantly. For example, compared to NetlistGNN (Yang et al., 2022), the previous SOTA for netlist representatin learning, our improvement on average is around 5.3% for wirelength prediction, and 8.6% for demand prediction, both in terms of MAE. See the Supplement for the full results, including the test performances for each design in the single-design setting.

Ablation study. We carried out an ablation study to understand the effects of the different strategies employed in our DE-HNN. In particular, the factors we wish to test are: (a) the use of direction in modeling nets, (b) the use of persistence diagrams (PDs) as features, and (c) the use of single and hierarchical VNs (virtual nodes). To this end, we compare the performance of the following versions: (a) base-E-HNN stands for treating a net as a standard hyperedge (thus **no direction**), and using **neither** PD features **nor** VNs. (b) base-DE-HNN is the base model for directed hypergraph (described in Section 3.1) with **neither** PDs **nor** VNs. In other words, the difference between base-DE-HNN and base-E-HNN shows the effect of adding directions. (c) base-DE-HNN+PD is the base model with only PDs. Hence the difference between (c) and (b) is to show the effect of PDs. (d) base-DE-

HNN+PD+single VN is the base model with PD and a single global VN. (e) Finally, full-DE-HNN is our full model with PDs and a two-level hierarchy of VNs. The results for net-based demand regression and cell-based congestion classification are shown in Figure 3. Full results are found in the Supplements, Other metrics and tasks show a similar behavior. For example, for demand prediction, from base-E-HNN, to adding directions, PDs, single VN, and finally two-level VNs, performance improves over the previous version by 2.5%, 2.6%, 1.0% and 3.4%, respectively. Overall, full-DE-HNN improves over the base-DE-HNN by around 6.8%, while its improvement over base-E-HNN (base model with no direction) is 9.2%.

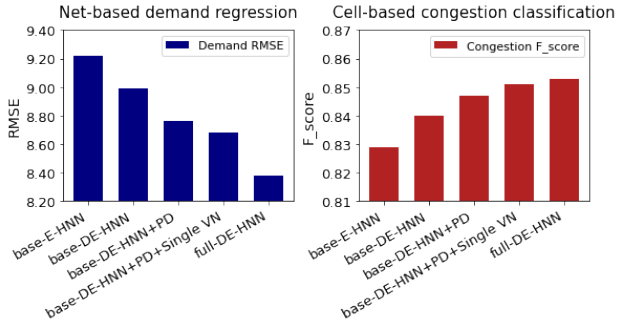


Figure 3: Ablation study for net-based demand regression (left, RMSE) and cell-based congestion classification (right, F-score).

5 Conclusion

In this paper, we presented an effective model for representation learning on directed hypergraphs. We considered learning on netlists, the hypergraph representation of circuits in chip design. This has great importance in practice but so far ML approaches for netlist representation learning has been limited, partly due to their huge sizes, long-range interactions, as well as the scarsity of benchmark datasets. We introduced several strategies to capture long-range interactions, graph motifs, and to consider direction in a hyperedge. Our model significantly outperforms a range of SOTA methods over a collection of chip designs. Our datasets will be publicly available, which we hope will facilitate further research on ML for chip design applications, pushing the ability of methods to capture long-range interactions. Finally, while we significantly outperformed other approaches, we note that in general, improvements in the cross-design setting are less prominent, potentially due to large variations in circuit designs. It will be interesting to further explore this direction.

Acknowledgements

This work is partially supported by NSF under grants CCF-2112665 and CCF-2310411, as well as by a gift fund from QualComm. The first, second and last authors would like to extend our sincere thanks to Prof. Andrew B. Kahng for the many insightful discussions and contributions in the early stage of this work, especially in exploring and experimenting the use of topological summaries for property prediction of netlists.

References

Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., and Ziegelmeier, L. (2017). Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, 18(8):1–35.

Al-Hyari, A., Szentimrey, H., Shamli, A., Martin, T., Gréwal, G., and Areibi, S. (2021). A deep learning framework to predict routability for fpga circuit placement. *ACM Trans. Reconfigurable Technol. Syst.*, 14(3).

Alawieh, M. B., Li, W., Lin, Y., Singhal, L., Iyer, M. A., and Pan, D. Z. (2020). High-definition routing congestion prediction for large-scale fpgas. In *Asia and South Pacific Design Automation Conference*, pages 26–31.

Bai, S., Zhang, F., and Torr, P. H. (2021). Hypergraph convolution and hypergraph attention. *Pattern Recognition*, 110:107637.

Brody, S., Alon, U., and Yahav, E. (2022). How attentive are graph attention networks? In *Int. Conf. on Learning Representations*.

Cai, C., Hy, T. S., Yu, R., and Wang, Y. (2023). On the connection between MPNN and graph transformer. In *Int. Conf. on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 3408–3430. PMLR.

Chai, Z., Zhao, Y., Liu, W., Lin, Y., Wang, R., and Huang, R. (2023). Circuitnet: An open-source dataset for machine learning in vlsi cad applications with improved domain-specific evaluation metric and learning strategies. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–1.

Chen, D., Lin, Y., Li, W., Li, P., Zhou, J., and Sun, X. (2020a). Measuring and relieving the over-smoothing problem for graph neural networks from the topological view. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3438–3445.

Chen, J., Kuang, J., Zhao, G., Huang, D. J. H., and Young, E. F. Y. (2020b). Pros: A plug-in for

routability optimization applied in the state-of-the-art commercial eda tool using deep learning. In *Int. Conf on Computer-Aided Design*.

Chen, X., Di, Z., Wu, W., Wu, Q., Shi, J., and Feng, Q. (2022). Detailed routing short violation prediction using graph-based deep learning model. *IEEE Trans. Circuits and Systems II: Express Briefs*, 69(2):564–568.

Chien, E., Pan, C., Peng, J., and Milenkovic, O. (2022). You are allset: A multiset function framework for hypergraph neural networks. In *International Conference on Learning Representations*.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. (2021). Rethinking attention with performers. In *Int. Conf. on Learning Representations*.

Cohen-Steiner, D., Edelsbrunner, H., and Harer, J. (2009). Extending persistence using poincaré and lefschetz duality. *Foundations of Computational Mathematics*, 9(1):79–103.

Dey, T. K. and Wang, Y. (2022). *Computational Topology for Data Analysis*. Cambridge University Press. 452 pages.

Dong, Y., Sawin, W., and Bengio, Y. (2020). Hnhn: Hypergraph networks with hyperedge neurons.

Dwivedi, V. P., Rampášek, L., Galkin, M., Parviz, A., Wolf, G., Luu, A. T., and Beaini, D. (2022). Long range graph benchmark. In *Advances in Neural Information Processing Systems*, volume 35.

Edelsbrunner, H. and Harer, J. (2010). *Computational Topology : an Introduction*. American Mathematical Society.

Edwards, R. T. (2020). Google/skywater and the promise of the open pdk. In *Workshop on Open-Source EDA Technology*.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with pytorch geometric. *Int. Conf. on Learning Representations*.

Fitzpatrick, P. (1989). Klaus deimling, nonlinear functional analysis.

Ghose, A., Zhang, V., Zhang, Y., Li, D., Liu, W., and Coates, M. (2021). Generalizable cross-graph embedding for gnn-based congestion prediction. In *IEEE/ACM Int. Conf. Computer Aided Design, ICCAD*, page 1–9.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *Int. Conf. on Machine Learning*, ICML, page 1263–1272.

Heydari, S. and Livi, L. (2022). *Message Passing Neural Networks for Hypergraphs*, page 583–592. Springer Nature Switzerland.

Hy, T. S., Trivedi, S., Pan, H., Anderson, B. M., and Kondor, R. (2019). Covariant compositional networks for learning graphs. In *Proc. Int. Workshop on Mining and Learning with Graphs (MLG)*.

Jegelka, S. (2022). Theory of graph neural networks: Representation and learning. *Int. Congress of Mathematicians, ICM*.

Kahng, A. B., Kim, M., Kim, S., and Woo, M. (2022). Rosettastone: Connecting the past, present, and future of physical design research. *IEEE Design and Test*, 39(5):70–78.

Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. (1999). Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE Trans. Very Large Scale Integr. Syst.*, 7(1):69–79.

Karypis, G. and Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. (2020). Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Int. Conf. on Machine Learning*, ICML.

Kim, J., Nguyen, D., Min, S., Cho, S., Lee, M., Lee, H., and Hong, S. (2022). Pure transformers are powerful graph learners. In *Advances in Neural Information Processing Systems*, volume 35.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *Int. Conf. on Learning Representations*.

Kirby, R., Godil, S., Roy, R., and Catanzaro, B. (2019). Congestionnet: Routing congestion prediction using deep graph neural networks. In *IFIP/IEEE Int. Conf. Very Large Scale Integration, VLSI-SoC*, pages 217–222.

Liang, R., Xiang, H., Pandey, D., Reddy, L., Ramji, S., Nam, G.-J., and Hu, J. (2020). Drc hotspot prediction at sub-10nm process nodes using customized convolutional network. In *Int. Symposium on Physical Design*, ISPD, page 135–142.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E. M., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J. (2021). A graph placement methodology for fast chip design. *Nature*, 594:207 – 212.

Müller, L., Morris, C., Galkin, M., and Rampášek, L. (2023). Attending to Graph Transformers. *Arxiv preprint*.

Ngo, N. K., Hy, T. S., and Kondor, R. (2023). Multiresolution graph transformers and wavelet positional encoding for learning long-range and hierarchical structures. *The Journal of Chemical Physics*, 159(3):034109.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32.

Rampášek, L., Galkin, M., Dwivedi, V. P., Luu, A. T., Wolf, G., and Beaini, D. (2022). Recipe for a general, powerful, scalable graph transformer. In *Advances in Neural Information Processing Systems*, volume 35, pages 14501–14515.

Sutherland, W. A. (2009). *Introduction to metric and topological spaces*. Oxford University Press.

Tabaghi, P. and Wang, Y. (2023). Universal Representation of Permutation-Invariant Functions on Vectors and Tensors. *arXiv preprint arXiv:2310.13829*.

Tabrizi, A. F., Rakai, L., Darav, N. K., Bustany, I., Behjat, L., Xu, S., and Kennings, A. (2018). A machine learning framework to identify detailed routing short violations from a placed netlist. In *ACM/ESDA/IEEE Design Automation Conference, DAC)*, pages 1–6.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. (2022). Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6).

Tian, M. and Wang, Y. (2019). A limit theorem for the 1st Betti number of layer-1 subgraphs in random graphs. ArXiv preprint.

Topping, J., Giovanni, F. D., Chamberlain, B. P., Dong, X., and Bronstein, M. M. (2022). Understanding over-squashing and bottlenecks on graphs via curvature. In *Int. Conf. on Learning Representations*.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *Int. Conf. on Learning Representations*.

Viswanathan, N., Alpert, C., Sze, C., Li, Z., and Wei, Y. (2012). The DAC 2012 routability-driven placement contest and benchmark suite. In *Annual Design Automation Conference*, DAC, page 774–782.

Viswanathan, N., Alpert, C. J., Sze, C., Li, Z., Nam, G.-J., and Roy, J. A. (2011). The ISPD-2011 routability-driven placement contest and benchmark suite. In *Int. Symposium on Physical Design*, ISPD, page 141–146.

Wagstaff, E., Fuchs, F., Engelcke, M., Posner, I., and Osborne, M. A. (2019). On the limitations of representing functions on sets. In *Int. Conf. on Machine Learning*, pages 6487–6494. PMLR.

Wang, B., Shen, G., Li, D., Hao, J., Liu, W., Huang, Y., Wu, H., Lin, Y., Chen, G., and Heng, P. A. (2022). Lhnn: Lattice hypergraph neural network for vlsi congestion prediction. In *ACM/IEEE Design Automation Conference*, DAC, page 1297–1302.

Xie, Z., Huang, Y.-H., Fang, G.-Q., Ren, H., Fang, S.-Y., Chen, Y., and Hu, J. (2018). Routenet: Routability prediction for mixed-size designs using convolutional neural network. In *IEEE/ACM Int. Conf on Computer-Aided Design, ICCAD*, pages 1–8.

Xie, Z., Liang, R., Xu, X., Hu, J., Duan, Y., and Chen, Y. (2021). Net2: A graph attention network method customized for pre-placement net length estimation. In *Asia and South Pacific Design Automation Conference*, ASPDAC, page 671–677.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2019). How powerful are graph neural networks? In *Int. Conf. on Learning Representations*.

Yan, Z., Ma, T., Gao, L., Tang, Z., and Chen, C. (2021). Link prediction with persistent homology: An interactive view. In *Int. Conf. on Machine Learning*, pages 11659–11669. PMLR.

Yang, S., Yang, Z., Li, D., Zhang, Y., Zhang, Z., Song, G., and Hao, J. (2022). Versatile multi-stage graph neural network for circuit representation. In *Advances in Neural Information Processing Systems*, volume 35.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In *Advances in Neural Information Processing Systems*, volume 30.

Zhang, B., Luo, S., Wang, L., and He, D. (2023). Rethinking the expressive power of GNNs via graph biconnectivity. In *The Eleventh Int. Conf. on Learning Representations*.

Zhao, Q., Ye, Z., Chen, C., and Wang, Y. (2020). Persistence enhanced graph neural network. In *Int. Conf. on Artificial Intelligence and Statistics*, pages 2896–2906. PMLR.

Supplement of the AISTATS submission #1797, Title: "DE-HNN: An effective neural model for Circuit Netlist representation"

A More Background

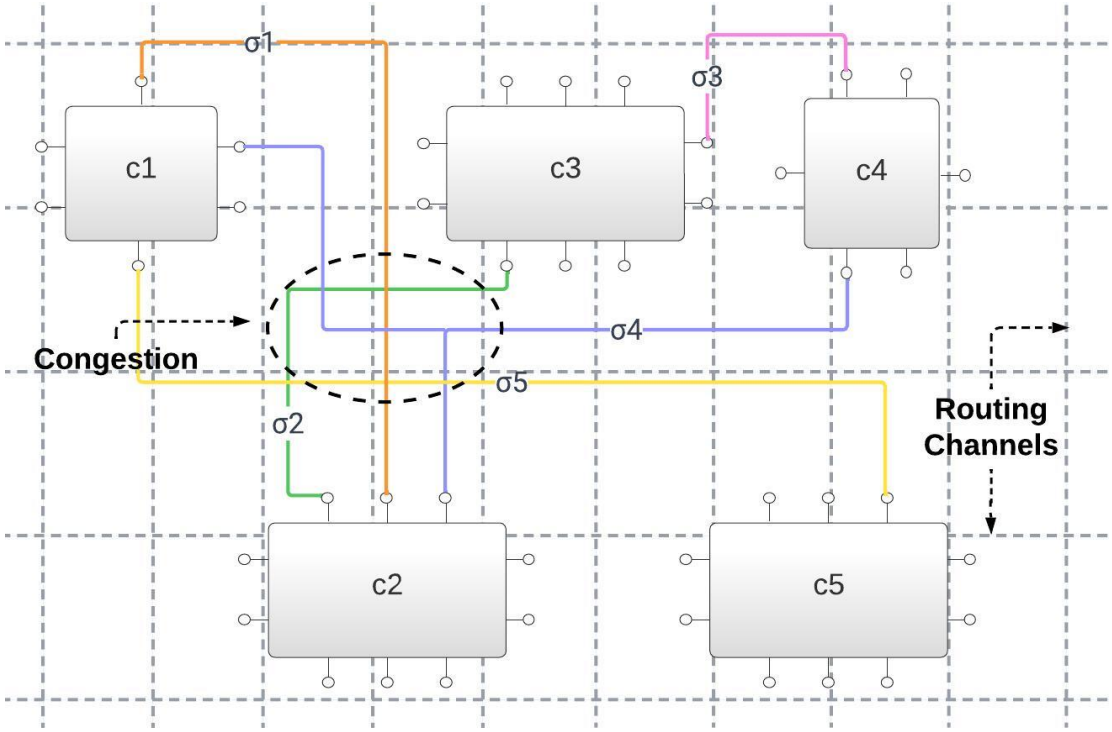


Figure 4: Visulization of a circuit netlits in the **post place-and-route** stage: Each cell (c_i) is positioned within the physical layout of the chip and interconnected with other components following the net maps (σ_i).

A.1 Circuit netlists

A circuit netlist is a textual representation of electronic components, such as logical boolean gates, and the connections between them. Figure 4 provides an example illustrating a circuit netlist consisting of five components interconnected by five nets. After laying out the netlist in the physical space (placement), during the routing stage, the edges of the netlist are mapped to the routing channels within the chip's physical floorplan (indicated by dashed lines in Figure 4). Routing congestion occurs when the number of edges to be routed in a specific region of the floorplan exceeds the available routing capacity.

A.2 Persistence homology based features

Persistent homology is one of the most important development in the field of topological data analysis in the past two decades. It can encode meaningful topological features in a multi-scale manner and has already been applied to numerous applications; see books (Edelsbrunner and Harer, 2010; Dey and Wang, 2022). Intuitively, given a domain X, a p-th homology class (or informally, a p-th homological feature) essentially captures a family of equivalent p-dimensional "holes" in X; for example, 0-, 1- and 2-dimensional homology captures connected components, equivalent classes of loops and of 2-dimensional voids, respectively. The p-th homology group $\mathrm{H}_p(X)$ (using $\mathbb{Z}_2$ coefficients) characterizes the space of p-th homological features of X, and its rank $rank(\mathrm{H}_p(X))$ corresponds to the number of independent p-th homological features. For example, $rank(\mathrm{H}_0(X))$ denotes the number of connected components of space X. If the domain X is a graph, then its 1-st homology group is homeomorphic the space of cycles (loops), and $rank(\mathrm{H}_1(X))$ is simply the number of independent cycles.

Persistent homology is a modern extension of homology, where instead of a single space X, we now inspect a sequence of growing spaces $X_1 \subseteq X_2 \subseteq \cdots X_m = X$, called a *filtration* [4]. Intuitively, we can view this filtration as an evolution of the space X. As the space grows, we track its corresponding topological features (as captured by the homology groups introduced above). Sometimes new topological features (e.g., a new component, or a hole) will appear, and sometimes they will disappear (e.g, a hole is filled and thus "disappear"). The *persistent homology (PH)* captures such creation and death of topological features, and outputs a feature summary called *persistent diagram (PD)*, consisting of the birth-time and death-time of classes of topological features during this evolution. In short, the PDs provide a multiscale summary for the topological features of X from the perspective of filtration $X_1 \subseteq X_2 \subseteq \cdots \subseteq X_m$. In the past few years, there have been a series of approaches to vectorize PDs or to kernelize them for ML applications; see Chapter 13 of (Dey and Wang, 2022). In our work, we use the so-called *persistence images* (Adams et al., 2017) to convert a PD to a finite dimensional vector to be used as part of the node feature.

Persistence diagram (PD) induced from Netlists. We compute the persistence diagram (PD) based on the following directed graph representation of the netlist (instead of using the directed hypergraph representation, for easier computation of persistence).

In particular, given a netlist, we create a directed graph $G = (V, E)$ where each node in V corresponds to a cell in the netlist, while a directed edge is formed between the driver cell of some net with each of the sink in that net; that is, each net $\sigma = (\mathsf{v}_\sigma, \mathsf{S}_\sigma)$ gives rise to a set of directed edges $\{(\mathsf{v}_\sigma, v)\}_{v \in \mathsf{S}_\sigma}$. We refer to this graph as the *star-graph* induced by the input netlist.

Now, for each node v in the star-graph G, we create its k-ring *out-flow neighborhood* G_v^{in} which is simply the subgraph in G spanned by all nodes reachable from v by a path of at most k hops. Symmetrically, the k-ring *in-flow neighborhood* G_v^{out} of v is the subgraph of G spanned by all nodes such that v is reachable within k hops. These two G_vs form the *local motifs* around v. We wish to obtain a feature vector summary for G_v^{in} and G_v^{out}. To this end, similar to (Yan et al., 2021; Zhao et al., 2020), we use the so-called 0*th extended persistence diagram* PD_v^* (Cohen-Steiner et al., 2009) of G_v^* (where $* \in \{in, out\}$) induced by the shortest path distance function to v as its summary. For our specific graph settig, using results of (Tian and Wang, 2019), one can show that PD_v^* computed this way, is a concise summary that encodes rich information around v, including the number of triangles incident to v, clustering coefficient of v, number of nodes at distance $r \leq k$ away from v, number of crossing edges from distance-r nodes to distance $(r+1)$ nodes, certain "shortest" system of cycle basis passing through v of G_v^*, and so on. Finally, each PD (for in-flow neighborhood and out-flow neighborhood of v) is vectorized to persistent images as in (Adams et al., 2017), and we further vectorize (flatten) each image matrix all together to generate a new vector as the persistent representation vector for each node.

A.3 Metis Partitioning

Metis (Karypis and Kumar, 1998) is currently State-of-the-art (SOTA) algorithm in chip design applications to provide balanced k-way partitions for large graphs efficiently. We choose Metis due to its practical performance, that it produces multiple clusters of balanced sizes, and the ease it is to control the size of the clusters. Besides Metis, hMetis (Karypis et al., 1999) developed on the base of Metis for hypergraph parititioning is an alternative

[4] Note that persistent homology has been extended for much broader families of filtrations then the growing sequence we describe here; see (Edelsbrunner and Harer, 2010; Dey and Wang, 2022).

choice. In our paper we used Metis in case we need partitioning for other graph based methods we compare our method with, and Metis will give us consistent partitioning. Nevertheless, we expect that hMetis can be used without much impact to the performance of our pipeline.

B Theoretical Analysis

B.1 Preliminaries

Let $\mathbb{D}$ be a domain, such as $\mathbb{Q}, \mathbb{R}$, and $\mathbb{R}^d$. Consider the set function $f : 2^{\mathbb{D}} \to \text{codom}(f)$ where $\mathbb{D}$ is a countable domain. Then, we have

$$\forall X \subseteq \mathbb{D} : f(X) = \rho \circ \Phi(X), \quad \Phi(X) = \sum_{x \in X} \phi(x), \tag{8}$$

where $\phi : \mathbb{D} \to \text{codom}(\phi) \subset \mathbb{R}$, and $\rho : \text{codom}(\phi) \to \text{codom}(f)$. This is the so-called sum-decomposable representation of f via $\mathbb{R}$ in terms of (ϕ, ρ) basis functions. We refer to ambient space of $\text{codom}(\phi)$ (in Eqn (8), it is $\mathbb{R}$) as the model's latent space (Zaheer et al., 2017). There is an extended sum-decomposable representation of f on multisets whose elements are drawn from an uncountable domain (e.g., $\mathbb{R}^d$) by restricting the size of input multisets. Recently, Tabaghi and Wang (2023) proposed an encoder $\phi : \mathbb{R}^d \to \text{codom}(\phi) \subset \mathbb{R}^{2dN}$ such that for any continuous multiset function $f : \mathbb{X}_{N,\mathcal{B}} \to \text{codom}(f)$ we have,

$$\forall X \in \mathbb{X}_{N,\mathcal{B}} : f(X) = \rho \circ \Phi(X),$$

where $\mathbb{X}_{N,\mathcal{B}}$ is the set of multisets of size N with elements drawn from $\mathcal{B}$ — a compact subset of $\mathbb{R}^d$ — and $\Phi(X) = \sum_{x \in X} \phi(x)$. The latent space dimension of this representation is $2dN$. As a result of this representation, $\Phi : \mathbb{X}_{N,\mathcal{B}} \to \text{codom}(\Phi)$ is an injective map where $\text{codom}(\Phi) \subset \mathbb{R}^{2dN}$. Furthermore, Tabaghi and Wang (2023) show that ρ is continuous over the latent space $\mathbb{R}^{2dN}$.

At their core, sum-decomposable models rely on a bijection between multisets and encoded features, that is, $X = \alpha \circ \Phi(X)$ for any multiset $X \in \mathbb{X}_{N,\mathcal{B}}$ and a bijective map α. In the following proposition, we first generalize this result to multisets with varied sizes.

Proposition 1. *Let $\mathcal{B}$ be a connected compact subset of $\mathbb{R}^d$. There exists a function $\phi : \mathbb{R}^d \to \mathbb{R}^{2dN}$ such that*

$$\forall X \in \mathbb{X}_{\leq N,\mathcal{B}} : X = \alpha\Big(\sum_{x \in X} \phi(x)\Big) = \alpha \circ \Phi(X)$$

where $\mathbb{X}_{\leq N,\mathcal{B}}$ is all multisets of size $\leq N$ with elements from $\mathcal{B}$, α is a continuous map over $\mathbb{R}^{2dN}$.

Proof. As mentioned earlier, Tabaghi and Wang (2023) proposed a continuous encoding function $\phi' : \mathbb{R}^d \to \mathbb{R}^{2dN}$ such that $\Phi'(x) = \sum_{x \in X} \phi'(x)$ is an injective map over multisets with exactly N elements, that is, $(\Phi')^{-1} \circ \Phi'(X) = X$ where elements of multiset X is drawn from $\mathbb{R}^d$ and $|X| = N$ [5].

Lemma 1. *The function Φ' is a homeomorphism.*

Proof. The function Φ' is continous and injective by contruction. We want to show that $(\Phi')^{-1}$ is continuous over $\text{codom}(\Phi') = \{\Phi'(X) : X \in \mathbb{X}_{N,\mathcal{B}}\}$ where $\mathcal{B}$ is a compact and connected subset of $\mathbb{R}^d$. The set $\mathbb{X}_{N,\mathcal{B}}$ is compact; refer to Lemma 5 in (Tabaghi and Wang, 2023). Also, $\text{codom}(\Phi') \subseteq \mathbb{R}^{2dN}$ forms a metric space with ℓ_2 metric; Hence, it is also a Hausdorff space. By the inverse function theorem, Φ' — a continuous bijection from a compact space to a Hausdorff space — has a continuous inverse; see Proposition 13.26 in (Sutherland, 2009). □

To extend the result of Lemma 1 to multisets of variable sizes, we follow the same proof sketch as the one used for the one-dimensional case (Wagstaff et al., 2019). In particular, let $x_\circ \in \mathbb{R}^d \setminus \mathcal{B}$ where $\inf_{x \in \mathcal{B}} \|x - x_\circ\|_2 > 0$.

[5] Note that we trivially changed the notation from ϕ to ϕ'.

Then, we define $\phi(x) = \phi'(x) - \phi'(x_\circ)$. For any multiset with $M \leq N$ elements from $\mathcal{B}$, say X, we have

$$\begin{aligned}\forall X \in \mathbb{X}_{\leq N,\mathcal{B}} : \Phi(X) &= \sum_{x\in X} \phi(x) = \sum_{x\in X} \phi'(x) - M\phi'(x_\circ) \\ &= \Phi'(X \cup \{\{\underbrace{x_\circ, \ldots, x_\circ}_{N-M}\}\}) + \text{const},\end{aligned}$$

where const $= -N\phi'(x_\circ)$. This is in the sum-decomposable form. Since Φ' is an injective map over $\mathbb{X}_{N,\mathcal{B}}$, Φ is also an injective map over $\mathbb{X}_{\leq N,\mathcal{B}}$. Specifically, we can derive a closed-form expression for its inverse as follows:

$$\begin{aligned}\Big({\Phi'}^{-1} \circ (\Phi(X) - \text{const})\Big) \cap \mathcal{B} &= \Big({\Phi'}^{-1} \circ \Phi'(X \cup \{\{\underbrace{x_\circ, \ldots, x_\circ}_{N-M}\}\})\Big) \cap \mathcal{B} \\ &= (X \cup \{\{\underbrace{x_\circ, \ldots, x_\circ}_{N-M}\}\}) \cap \mathcal{B} \\ &= X.\end{aligned}$$

In other words, we have $\Phi^{-1}(U) = {\Phi'}^{-1}\big(U - \text{const}\big) \cap \mathcal{B}$ for all $U \in \text{codom}(\Phi) = \{\Phi(X) : X \in \mathbb{X}_{\leq N,\mathcal{B}}\}$.

The function $\Phi : \mathbb{X}_{\leq N,\mathcal{B}} \to \text{codom}(\Phi)$ is a continuous bijection. Its domain is compact because it is a finite union of comapct spaces, that is, $\mathbb{X}_{\leq N,\mathcal{B}} = \cup_{n\in[N]}\mathbb{X}_{n,\mathcal{B}}$. Furthermore, its codomain forms a metric space with ℓ_2 distance; Hence, it is also a Hausdorff space. Therefore, by an inverse function theorem, Φ has a continuous inverse; (Sutherland, 2009). If we let $\alpha = \Phi^{-1}$, we arrive at the Proposition's statement. □

Remark 1. *Proposition 1 gurantees the continuity of α over a compact set* $\text{codom}(\Phi) \subset \mathbb{R}^{2dN}$. *This function can be continuously extended to the ambient space $\mathbb{R}^{2dN}$; refer to the continuous extension theorem (Fitzpatrick, 1989).*

B.2 Proof of Theorem 1

As discussed in the main text, the general net-value function is nested-permutation invariant and take the form of Eqn (6) in the main text. In what follows, we provide the detailed version of Theorem 1 and show that such nested-permutation invariant function adapts a nested sum-decomposition aligned with our DE-HNN's node and net update rules.

Theorem 1 (Detailed)**.** *Let $\mathcal{F}$ be any continuous, nested-permutation invariant, net-value function as in Eqn (6). For simplicity, assume both input nets and output of M take values in a compact set $\mathcal{B} \subset \mathbb{R}^d$, a connected compact subset of $\mathbb{R}^d$. We also let $N_v = \max\{|\mathcal{I}(v)| : \forall v\}$ and $N_\sigma = \max\{|\mathsf{S}(\sigma)| : \forall \sigma\}$. Then we have that $\mathcal{F}$ can be expressed as the following sum-decomposition:*

$$\forall \sigma : \mathcal{F}\Big(\{\mu_{\sigma'}\}_{\sigma'\in\mathcal{I}(\mathsf{v}_\sigma)}, \Big\{\{\mu_{\sigma'}\}_{\sigma'\in\mathcal{I}(v')}\Big\}_{v'\in\mathsf{S}_\sigma}\Big) = \rho\Big(\sum_{\sigma'\in\mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu_{\sigma'}), \sum_{v'\in\mathsf{S}_\sigma} \phi_2\big(\sum_{\sigma'\in\mathcal{I}(v')} \phi_1(\mu_{\sigma'})\big)\Big),$$

where $\phi_1 : \mathbb{R}^d \to \mathbb{R}^{d'(d,N_v)}$, $\phi_2 : \mathbb{R}^{d'(d,N_v)} \to \mathbb{R}^{d''(d,N_v,N_\sigma)}$, and $\rho : \mathbb{R}^{d'(d,N_v)} \times \mathbb{R}^{d''(d,N_v,N_\sigma)} \to \mathbb{R}^d$ are continuous functions.

Proof. The general net-value function takes the following form:

$$\forall \sigma : \ \mathcal{F}\Big(\{\mu_{\sigma'}\}_{\sigma'\in\mathcal{I}(\mathsf{v}_\sigma)}, \Big\{\{\mu_{\sigma'}\}_{\sigma'\in\mathcal{I}(v')}\Big\}_{v'\in\mathsf{S}_\sigma}\Big).$$

We assume $\mathcal{F}$ is a continuous function that takes values in $\mathcal{B}$, a connected compact subset of $\mathbb{R}^d$, and $N_v = \max\{|\mathcal{I}(v)| : \forall v\}$. From Proposition 1, there exist continuous functions ϕ_1 and α_1 such that

$$\forall X \in \mathbb{X}_{\leq N_v,\mathcal{B}} : X = \alpha_1(\sum_{x\in X} \phi_1(x)),$$

where $\text{codom}(\phi_1) \subseteq \mathbb{R}^{2dN_v}$, that is, the latent dimension depends on d and N_v which we denote as $d^{'}(d, N_v)$. Therefore, we apply the result in Proposition 1 to sets of net values (of size at most N_v) and express $\mathcal{F}$ follows:

$$\forall \sigma : \mathcal{F}\Big(\{\mu_{\sigma'}\}_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)}, \Big\{\{\mu_{\sigma'}\}_{\sigma' \in \mathcal{I}(v')}\Big\}_{v' \in \mathsf{S}_\sigma}\Big) = \mathcal{F}\Big(\alpha_1\big(\sum_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu'_\sigma)\big), \Big\{\alpha_1\big(\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\sigma')\big)\Big\}_{v' \in \mathsf{S}_\sigma}\Big)$$
$$= \mathcal{F}_1\Big(\sum_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu'_\sigma), \Big\{\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\sigma')\Big\}_{v' \in \mathsf{S}_\sigma}\Big)$$

where $\mathcal{F}_1(x, Y) \overset{\text{def.}}{=} \mathcal{F}(\alpha_1(x), \{\alpha_1(y)\}_{y \in Y})$ for all $x \in \text{codom}(\Phi_1) = \{\Phi_1(Y) = \sum_{y \in Y} \phi_1(y) : Y \in \mathbb{X}_{\leq N_v, \mathcal{B}}\}$ and any set Y with elements in $\text{codom}(\Phi_1)$. Since both α_1 and $\mathcal{F}$ are continuous functions, then $\mathcal{F}_1$ is also a continuous function. [6]

Next, we can apply the result of Proposition 1 to $\mathcal{F}_1$. The elements of the set $\Big\{\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\sigma')\Big\}_{v' \in \mathsf{S}_\sigma}$ take values in $\text{codom}(\Phi_1)$ — a connected compact subset of $\mathbb{R}^{d'(d, N_v)}$. If we assume $|\mathsf{S}_\sigma| \leq N_\sigma$ for all σ, Proposition 1 claims that there exist continuous functions α_2 and ϕ_2 such that the following holds true:

$$\forall \sigma : \mathcal{F}_1\Big(\sum_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu'_\sigma), \Big\{\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\sigma')\Big\}_{v' \in \mathsf{S}_\sigma}\Big) = \mathcal{F}_1\Bigg(\sum_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu'_\sigma), \alpha_2\Big(\sum_{v' \in \mathsf{S}_\sigma} \phi_2\big(\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\sigma')\big)\Big)\Bigg)$$
$$= \rho\Bigg(\sum_{\sigma' \in \mathcal{I}(\mathsf{v}_\sigma)} \phi_1(\mu'_\sigma), \sum_{v' \in \mathsf{S}_\sigma} \phi_2\big(\sum_{\sigma' \in \mathcal{I}(v')} \phi_1(\sigma')\big)\Bigg)$$

where $\rho(x, y) \overset{\text{def}}{=} \mathcal{F}_1(x, \alpha_2(y))$ for all $x \in \text{codom}(\Phi_1)$ and $y \in \text{codom}(\Phi_2) \overset{\text{def}}{=} \{\sum_{z \in Z} \phi_2(z) : Z \in \text{codom}(\Phi_1), |Z| \leq N_\sigma\}$. The function ρ is a composition of continuous functions $\mathcal{F}_1$ and α_2. Therefore, it is a continuous function. Also, we have $\text{codom}(\phi_2) \subseteq \mathbb{R}^{2d'(d, N_v)N_\sigma}$, that is, the latent dimension depends on d, N_v, and N_σ which we denote as $d''(d, N_v, N_\sigma)$. Finally, functions ϕ_2 and ρ are continuous over compact domains $\text{codom}(\Phi_1)$, and $\text{codom}(\Phi_1) \times \text{codom}(\Phi_2)$. Therefore, they can be continuously extended to $\mathbb{R}^{d'(d, N_v)}$ and $\mathbb{R}^{d'(d, N_v)} \times \mathbb{R}^{d''(d, N_v, N_\sigma)}$. □

C Experimental details

C.1 Dataset Statistics

We report the sizes of each design and their cell/net-degrees distribution in Table 5, and net-based demand/wirelength distributions in the Table 6. We can see that (hyper)graphs in our dataset are large and sparse, ranging from 400K to 1.3M nodes. with few outliers that have high degrees. We also summarize the more detailed distributions of net-based demand, net-based wirelength, and cell-based congestion in figure5.

C.2 Experiment Setup

We engineer the input features as Cell features and Net features. None of the following features contained placement information or are computed from placement information. We engineer cell features as follows:

- We analyze the statistics in a design including the minimum and maximum of all cells' width, height. Then, we normalize all these quantities to be in the range $[0, 1]$. We concatenate them with the cell's discrete orientation and cell's degree (number of nets each cell connecting to), that results into the cell's input feature vector.
- We calculate the top-10 eigenvectors of the graph Laplacian of the heterogenous graph as the positional encoding (denoted as LapPE) for every cell and net.
- We compute the persistence diagram (denoted as PD) for each cell-node v on a directed graph $G = (V, E)$, as we mentioned earlier in A.2, as a common cell-node input feature vector for all baselines, to capture the local topological information.

[6] We can use the matching distance to define a metric on multisets; refer to (Tabaghi and Wang, 2023) for details.

- We also compute the degree distribution for each cell-node v based on their k-ring ***undirected*** *neighborhood* G_v in the star-graph we introduced in A.2 to compute the persistence diagrams. However, we ignore the direction when we compute the degree distribution.

For the net features, for all models other than linear Transformer, we initialize the features of each **net** $\sigma = (\mathsf{v}_\sigma, \mathsf{S}_\sigma)$ as the net's degree (i.e. number of cells each net connecting to). For Linear Transformer, in order to provide better topology information, we instead initialize the features of each **net** $\sigma = (\mathsf{v}_\sigma, \mathsf{S}_\sigma)$ (denoted as $M(\sigma)$) as the average of all the features of the nodes this net contains, computed as,

$$M(\sigma) = \frac{1}{|\mathsf{S}_\sigma| + 1}(m(\mathsf{v}_\sigma) + \sum_{v' \in \mathsf{S}_\sigma} m(v')). \tag{9}$$

As also described in the main text, for GCN and GATv2, we use the bipartite graph representation of the input netlist, where there are two types of graph nodes: those corresponding to cells and those corresponding nets. If a cell is contained in a net, then there is an edge between their respective graph nodes. We use **heterogenous** message passing, where nodes corresponding to cells use a different message passing mechanism from nodes corresponding to nets.

For GCN, GATv2, HyperConv, and all other HNN based models, we used 4 layers with 64 dimension each layer as the setting. For Linear Transformer, we used 2 layers with 64 dimension each layer as the setting. For NetlistGNN we used 4 layers with node dimension 64 and net dimension 64 as the setting. For E-HNN/DE-HNN based models, we used 3 layers with node dimension 64.

We report the number of parameters, training time per epoch and total training epochs in the Table 4. We aim to use similar number of parameters, although currently full-DE-HNNdoes have the highest number of parameters, with NetlistGNN having the second largest. Interestingly, while they all take similar number of epochs to converge, our DE-HNN in fact is much faster per epoch, so overall takes less time to train. We used two NVIDIA A100-SXM4-80GB GPUs on Linux CentOS Stream system (Ver.8) for both single-design and cross-design experiments.

Model	GCN	GATv2	HyperConv	Lin. Transformer	NetlistGNN	AllSet	HMPNN	HNHN	base-DE-HNN	full-DE-HNN
Num. of parameters	218113	251905	218625	203201	364743	274433	276513	289565	272165	383105
Time per epoch (single design)	2.77	2.91	1.06	3.72	1.13	0.79	0.69	0.62	0.49	0.65
Total num. of epochs (single design)	878	775	670	636	716	689	691	699	673	810
Time per epoch (cross design)	33.20	36.28	12.28	55.37	16.51	11.35	4.14	5.37	5.26	5.50
Total num. of epochs (cross design)	478	482	473	471	455	460	472	468	479	473

Table 4: Comparison of Model complexity for all the models we used. We compare the number of parameters, training time per epoch for single design (on average), total training epoches to converge for single design (on average), trainig time per epoch for cross design, and total training epoches to converge for cross design.

C.3 More experimental results

In the main text, we have reported plots for ablation studies of different strategies used in our model. Here in Table 7 we report detailed numbers for all three tasks: net-based wirelength regression, net-based demand regression, and cell-based congestion classification across all the designs.

Besides ablation study over our models, as we mentioned in main paper, adding the persistence diagrams (PDs) features are beneficial for both our method and baselines. We have already shown the benefits for our model in the ablation studies in the main text and above. Here in Table 8, we show its benefits to the three best performing baseline models: NetlistGNN, GCN, GATv2. As we can see, using PDs improve all these models. Note that in the main text, when we report results of these baseline models, we are already reporting results with PDs.

We show more detailed experimental results of single-design experiments for net-based wirelength regression in Table 9, net-based regression in Table 10 and cell-based congestion classification in Table 11 for each design. We also show more results of cross-design experiments for net-based wirelength regression, net-based demand regression and cell-based congestion classification in Table 13. Similar to how we report the results in the main paper, we highlight the models' results with the best performance in red, and highlight the models (other than our models) that with second best performance as blue. Interestingly, in the single design case, it appears that

Design	Cells	Nets	Cell-degree				Net-degree			
			Min	Max	Mean	STD	Min	Max	Mean	STD
Superblue1	797,938	821,523	0.0	1243	3.70	5.66	0.0	140605	3.58	155.85
Superblue2	951,166	985,117	0.0	1317	3.51	5.66	0.0	190487	3.39	192.22
Superblue3	901,254	925,667	0.0	2245	3.65	5.23	0.0	168630	3.55	175.91
Superblue5	727,341	803,681	0.0	1381	3.46	5.82	0.0	114259	3.13	128.03
Superblue6	998,122	1049,225	0.0	1689	3.57	4.03	0.0	179410	3.39	175.69
Superblue7	1319,052	1339,522	0.0	849	3.91	3.03	0.0	265765	3.85	230.24
Superblue9	810,812	830.308	0.0	1265	3.83	4.78	0.0	129541	3.74	202.42
Superblue11	923,355	954,144	0.0	1983	3.74	6.97	0.0	203194	3.63	294.36
Superblue14	604,921	627,036	0.0	1023	3.90	4.66	0.0	167911	3.76	300.30
Superblue16	671,284	696,983	0.0	1016	3.77	6.12	0.0	140741	3.63	238.58
Superblue18	459,495	468,888	0.0	1192	4.22	4.30	0.0	102047	4.14	150.74
Superblue19	495,234	510,258	0.0	1507	3.58	6.02	0.0	94682	3.48	135.31

Table 5: Dataset details & statistics. 1st column in the table shows the name of the design, 2nd column and 3rd column show the number of cells and nets in each design, 4th-7th columns show the distribution of cell-degrees for each design and 8th-11th columns show the distribution of net-degrees for each design.

Design	Net-based demand				Net-based wirelength				Cell-based congestion			
	Min	Max	Mean	STD	Min	Max	Mean	STD	Min	Max	Mean	STD
Superblue1	0.0	103.50	26.24	8.01	8.91	23.92	14.80	2.45	0.0	12.00	1.21	0.55
Superblue2	0.0	139.37	26.61	8.52	8.91	24.61	15.21	2.69	0.0	5.00	0.70	0.45
Superblue3	0.0	103.50	24.07	7.31	8.91	23.83	14.91	2.45	0.0	5.19	0.90	0.47
Superblue5	0.0	1203.25	41.62	32.00	8.91	24.06	15.68	2.96	0.0	78.92	1.27	0.94
Superblue6	0.0	980.33	33.83	27.45	8.91	23.83	14.88	2.57	0.0	74.15	1.35	1.15
Superblue7	0.0	495.25	23.89	5.34	8.91	23.93	14.89	2.48	0.0	43.38	1.09	0.33
Superblue9	0.0	79.50	22.47	8.02	8.91	23.83	14.71	2.42	0.0	5.0	0.74	0.46
Superblue11	0.0	75.00	20.05	6.54	8.91	24.17	15.11	2.38	0.0	8.0	0.93	0.36
Superblue14	0.0	401.41	23.42	9.11	8.91	23.67	15.06	2.48	0.0	46.85	1.06	0.65
Superblue16	0.0	1091.0	28.96	14.09	8.91	23.51	15.01	2.54	0.0	65.53	1.29	0.91
Superblue18	0.0	50.0	20.39	4.13	8.91	23.26	14.98	2.28	0.0	4.0	0.91	0.27
Superblue19	0.0	87.83	23.05	6.40	8.91	23.53	14.86	2.36	0.0	6.0	0.96	0.50

Table 6: Dataset details & statistics. This table shows the distribution of net-based demands for each design, distribution of net-based logged wirelength for each design and the distribution of cell-based congestion values for each design. Remind that cell-based congestion, as we described in the paper, we classify the cell-based congestion values (computed as the ratio of demand/capacity) into (a) [0, 0.9], *not-congested*; and (b) [0.9, inf]; *congested*.

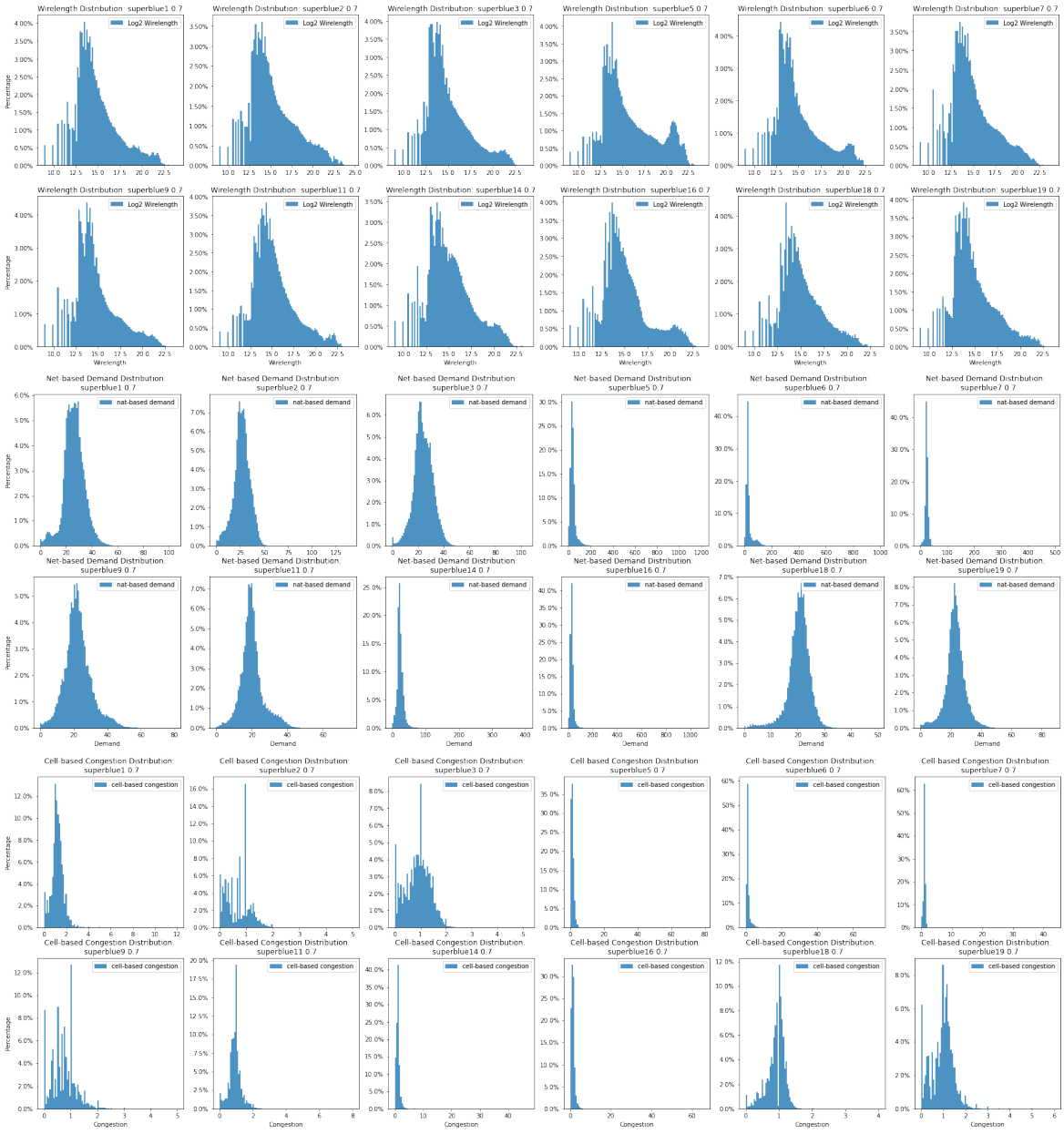


Figure 5: Net-based wirelength, net-based demand and cell-based congestion distributions of each design.

	net-based wirelength regression			net-based demand regression			cell-based congestion classification		
Model	**RMSE ↓**	**MAE ↓**	**Pearson ↑**	**RMSE ↓**	**MAE ↓**	**Pearson ↑**	**Precision ↑**	**Recall ↑**	**F_score ↑**
base E-HNN	1.818	1.344	0.731	9.228	5.959	0.591	0.816	0.864	0.829
base DE-HNN	1.751	1.269	0.748	8.997	5.764	0.630	0.824	0.860	0.840
base DE-HNN+PD	1.731	1.257	0.754	8.765	5.526	0.647	0.830	0.869	0.847
base DE-HNN+PD+S. VN	1.724	1.253	0.758	8.687	5.519	0.658	0.832	0.874	0.851
full DE-HNN	1.689	1.245	0.770	8.381	5.334	0.683	0.833	0.876	0.853

Table 7: Ablation Study: Full experimental results to show the improvements from base E-HNN to full DE-HNN.

Model	net-based wirelength regression			net-based demand regression			cell-based congestion classification		
	RMSE ↓	MAE ↓	Pearson ↑	RMSE ↓	MAE ↓	Pearson ↑	Precision ↑	Recall ↑	F_score ↑
NetlistGNN with no PD	1.818	1.344	0.731	9.237	6.060	0.592	0.806	0.855	0.818
NetlistGNN+PD	1.773	1.320	0.740	9.063	5.839	0.623	0.812	0.860	0.831
Improvement	2.5%	1.8%	1.2%	1.9%	3.6%	5.2%	0.7%	0.6%	1.6%
GCN with no PD	1.809	1.326	0.735	9.698	6.453	0.547	0.746	0.837	0.784
GCN+PD	1.762	1.276	0.750	9.321	6.163	0.570	0.761	0.857	0.802
Improvement	1.9%	3.6%	5.2%	3.9%	4.5%	4.2%	2.0%	2.4%	2.3%
GATv2 with no PD	1.920	1.401	0.659	9.710	6.392	0.539	0.802	0.856	0.811
GATv2+PD	1.812	1.330	0.687	9.342	6.118	0.561	0.810	0.864	0.835
Improvement	0.7%	0.6%	1.6%	3.8%	4.3%	4.1%	1.0%	1.0%	3.0%

Table 8: Ablation Study: the effect of using persistence diagrams (PDs) to three best-performing baselines. For each method, the 3rd row shows the percentage of improvement after using PD as part of the input features. Note that in the main text, the results we reported are those baselines+PD.

the methods AllSet and NetlistGNN often perform the best among the baselines for net-length and net-based demand regression tasks, while for the cell-based congestion classification, other models (e.g, GATv2) sometimes perform the best among baselines. In cross-design experiments, GCN and GATv2 sometimes emerge as winners. In the cases that a baseline model is better than our model, we highlight that baseline model's results in red instead.

Preliminary exploration with placement information. In this paper, we focus on the case when our input do not have placement information (i.e, coordinates of cells). We also conducted some preliminary experiments to examine the effect of adding cell placement information to initial node features, and use placement information based partitioning instead of Metis. We first partition the nodes in netlist circuit (after placement) into k bounding boxes V_i with fixed width and height as 0.8 millimeters. The number k is depending on the width and height of each netlist circuit. With coordinates of cells added and bounding boxes based partitioning (for full-DE-HNN), we rerun both single-design and cross-design net-based demand regression experiments on superblue19. See Table 12 below. Placement information helps to improve the performance on average around 10% for MAE and RMSE, and the improvement is more substantial in terms of Pearson Correlation. We aim to explore more about how to effectively leverage placement information in the future works.

Design	Model	RMSE ↓	MAE ↓	Pearson ↑	Design	Model	RMSE ↓	MAE ↓	Pearson ↑
Superblue1	GCN	1.731	1.243	0.751	Superblue9	GCN	1.819	1.338	0.697
	GATv2	1.742	1.253	0.742		GATv2	1.853	1.381	0.683
	HyperConv	1.988	1.413	0.645		HyperConv	1.966	1.435	0.639
	AllSet	1.700	1.235	0.757		AllSet	1.684	1.238	0.750
	HMPNN	1.779	1.291	0.727		HMPNN	1.827	1.355	0.695
	HNHN	1.813	1.319	0.720		HNHN	1.821	1.335	0.701
	NetlistGNN	1.761	1.276	0.735		NetlistGNN	1.805	1.325	0.701
	base DE-HNN	1.774	1.274	0.731		base DE-HNN	1.765	1.287	0.720
	full DE-HNN	**1.657**	**1.203**	**0.771**		**full DE-HNN**	**1.663**	**1.226**	**0.757**
	Improvement	2.5%	2.6%	1.8%		**Improvement**	1.2%	1.0%	0.9%
Superblue2	GCN	2.011	1.487	0.714	Superblue11	GCN	1.831	1.356	0.694
	GATv2	1.931	1.485	0.723		GATv2	1.814	1.349	0.702
	HyperConv	2.217	1.661	0.641		HyperConv	1.986	1.468	0.633
	AllSet	1.817	1.353	0.775		AllSet	1.717	1.274	0.740
	HMPNN	1.981	1.480	0.723		HMPNN	1.830	1.369	0.696
	HNHN	1.992	1.488	0.724		HNHN	1.834	1.353	0.697
	NetlistGNN	1.857	1.410	0.748		NetlistGNN	1.780	1.291	0.707
	base DE-HNN	1.933	1.438	0.741		base DE-HNN	1.741	1.290	0.731
	full DE-HNN	**1.810**	**1.344**	**0.778**		**full DE-HNN**	**1.690**	**1.257**	**0.751**
	Improvement	0.4%	0.7%	0.4%		**Improvement**	1.6%	1.3%	1.5%
Superblue3	GCN	1.741	1.273	0.738	Superblue14	GCN	1.792	1.331	0.739
	GATv2	1.716	1.270	0.748		GATv2	1.794	1.361	0.738
	HyperConv	1.941	1.438	0.661		HyperConv	2.116	1.547	0.605
	AllSet	1.685	1.239	0.759		AllSet	1.756	1.299	0.750
	HMPNN	1.752	1.300	0.734		HMPNN	1.873	1.415	0.707
	HNHN	1.769	1.303	0.729		HNHN	1.895	1.404	0.703
	NetlistGNN	1.689	1.239	0.754		NetlistGNN	1.812	1.363	0.731
	base DE-HNN	1.765	1.271	0.741		base DE-HNN	1.816	1.336	0.730
	full DE-HNN	**1.679**	**1.234**	**0.761**		**full DE-HNN**	**1.728**	**1.282**	**0.760**
	Improvement	0.4%	0.4%	0.3%		**Improvement**	1.6%	1.3%	1.3%
Superblue5	GCN	1.892	1.428	0.801	Superblue16	GCN	1.763	1.265	0.741
	GATv2	1.909	1.438	0.793		GATv2	1.741	1.267	0.751
	HyperConv	2.114	1.606	0.742		HyperConv	2.047	1.446	0.639
	AllSet	1.842	1.361	0.810		AllSet	1.688	1.207	0.772
	HMPNN	2.224	1.754	0.706		HMPNN	1.896	1.362	0.701
	HNHN	2.069	1.562	0.756		HNHN	1.816	1.312	0.731
	NetlistGNN	1.915	1.406	0.790		NetlistGNN	1.773	1.274	0.736
	base DE-HNN	1.881	1.393	0.799		base DE-HNN	1.705	1.218	0.767
	full DE-HNN	**1.795**	**1.330**	**0.822**		**full DE-HNN**	**1.656**	**1.194**	**0.782**
	Improvement	2.6%	2.3%	1.5%		**Improvement**	1.9%	1.1%	1.3%
Superblue6	GCN	1.811	1.341	0.740	Superblue18	GCN	1.635	1.249	0.739
	GATv2	1.782	1.322	0.751		GATv2	1.701	1.246	0.714
	HyperConv	1.928	1.435	0.702		HyperConv	1.937	1.345	0.681
	AllSet	1.733	1.260	0.766		AllSet	1.664	1.246	0.730
	HMPNN	1.851	1.360	0.730		HMPNN	1.769	1.335	0.686
	HNHN	1.859	1.365	0.727		HNHN	1.752	1.314	0.696
	NetlistGNN	1.940	1.452	0.725		NetlistGNN	**1.625**	**1.213**	**0.752**
	base DE-HNN	1.786	1.296	0.749		base DE-HNN	1.653	1.263	0.735
	full DE-HNN	**1.689**	**1.233**	**0.780**		**full DE-HNN**	1.632	1.219	0.743
	Improvement	2.5%	2.1%	1.8%		**Improvement**	0.4%	0.5%	1.2%
Superblue7	GCN	1.842	1.315	0.710	Superblue19	GCN	1.637	1.214	0.762
	GATv2	1.847	1.371	0.699		GATv2	1.634	1.198	0.765
	HyperConv	2.010	1.479	0.628		HyperConv	1.910	1.405	0.666
	AllSet	1.753	1.295	0.733		AllSet	1.582	1.171	0.783
	HMPNN	1.889	1.409	0.681		HMPNN	1.705	1.255	0.739
	HNHN	1.858	1.370	0.694		HNHN	1.752	1.288	0.727
	NetlistGNN	1.801	1.316	0.721		NetlistGNN	1.635	1.205	0.765
	base DE-HNN	1.760	1.291	0.701		base DE-HNN	1.641	1.208	0.763
	full DE-HNN	**1.719**	**1.267**	**0.747**		**full DE-HNN**	**1.557**	**1.153**	**0.792**
	Improvement	1.9%	2.2%	1.9%		**Improvement**	1.6%	1.5%	1.1%

Table 9: Average results of single-design net-based hpwl(wirelength) regression for each design, based on 4-fold cross validations. Last row "Improvement" refers to the improvement of our full DE-HNN model over the best baseline approach for each metric.

Design	Model	RMSE ↓	MAE ↓	Pearson ↑	Design	Model	RMSE ↓	MAE ↓	Pearson ↑
Superblue1	GCN	6.469	4.979	0.595	Superblue9	GCN	6.871	5.156	0.520
	GATv2	6.409	4.964	0.612		GATv2	6.893	5.139	0.512
	HyperConv	6.662	4.951	0.224		HyperConv	7.014	5.241	0.289
	AllSet	6.100	4.587	0.650		AllSet	6.184	4.576	0.640
	HMPNN	6.770	5.206	0.541		HMPNN	7.152	5.367	0.467
	HNHN	6.394	4.825	0.610		HNHN	6.544	4.884	0.582
	Lin. Transformers	7.991	6.046	0.089		Lin. Transformers	8.007	5.934	0.092
	NetlistGNN	6.039	4.623	0.660		NetlistGNN	6.511	4.796	0.589
	base DE-HNN	6.093	4.670	0.653		base DE-HNN	2.990	2.228	0.696
	full DE-HNN	**5.674**	**4.263**	**0.709**		**full DE-HNN**	**5.685**	**4.237**	**0.709**
	Improvement	6.0%	7.1%	7.4%		**Improvement**	8.1%	7.4%	10.8%
Superblue2	GCN	6.556	5.245	0.641	Superblue11	GCN	5.693	4.224	0.502
	GATv2	6.736	5.288	0.616		GATv2	5.684	4.203	0.504
	HyperConv	7.654	6.151	0.374		HyperConv	6.243	4.523	0.105
	AllSet	6.430	4.981	0.659		AllSet	5.115	3.792	0.625
	HMPNN	6.982	5.501	0.579		HMPNN	5.974	4.402	0.418
	HNHN	6.699	5.217	0.625		HNHN	5.277	3.882	0.592
	Lin. Transformers	8.251	6.356	0.313		Lin. Transformers	6.576	4.678	0.034
	NetlistGNN	6.259	4.932	0.682		NetlistGNN	5.176	3.830	0.617
	base DE-HNN	6.399	5.035	0.663		base DE-HNN	5.214	3.855	0.608
	full DE-HNN	**5.966**	**4.637**	**0.718**		**full DE-HNN**	**4.918**	**3.677**	**0.666**
	Improvement	4.7%	6.0%	5.3%		**Improvement**	3.9%	3.0%	6.6%
Superblue3	GCN	5.789	4.512	0.612	Superblue14	GCN	7.261	4.827	0.584
	GATv2	5.837	4.558	0.565		GATv2	7.370	4.825	0.545
	HyperConv	6.468	5.149	0.265		HyperConv	8.210	5.241	0.210
	AllSet	5.265	4.014	0.695		AllSet	7.162	4.565	0.619
	HMPNN	6.022	4.713	0.572		HMPNN	7.687	4.992	0.540
	HNHN	5.686	4.338	0.631		HNHN	7.327	4.693	0.597
	Lin. Transformers	7.046	5.493	0.264		Lin. Transformers	8.853	6.168	0.176
	NetlistGNN	5.414	4.214	0.673		NetlistGNN	6.872	4.444	0.642
	base DE-HNN	5.423	4.188	0.670		base DE-HNN	6.874	4.453	0.639
	full DE-HNN	**5.041**	**3.837**	**0.725**		**full DE-HNN**	**6.533**	**4.211**	**0.684**
	Improvement	4.3%	4.4%	4.3%		**Improvement**	4.9%	5.2%	6.5%
Superblue5	GCN	27.169	14.867	0.504	Superblue16	GCN	**11.774**	8.242	0.391
	GATv2	27.343	14.754	0.508		GATv2	12.853	8.283	0.377
	HyperConv	29.563	15.817	0.107		HyperConv	16.501	9.486	0.175
	AllSet	27.881	14.632	0.490		AllSet	12.558	7.837	0.469
	HMPNN	28.753	15.485	0.439		HMPNN	13.539	8.582	0.312
	HNHN	28.314	15.309	0.473		HNHN	12.720	8.082	0.446
	Lin. Transformers	32.614	16.889	0.076		Lin. Transformers	14.020	8.827	0.003
	NetlistGNN	27.586	14.470	0.515		NetlistGNN	12.982	8.385	0.353
	base DE-HNN	27.205	14.129	0.536		base DE-HNN	12.282	7.946	0.465
	full DE-HNN	**26.684**	**13.512**	**0.565**		**full DE-HNN**	11.867	**7.644**	**0.520**
	Improvement	1.8%	6.6%	9.7%		**Improvement**	-	2.5%	10.9%
Superblue6	GCN	21.963	12.692	0.607	Superblue18	GCN	3.061	2.262	0.681
	GATv2	21.492	12.119	0.629		GATv2	3.102	2.285	0.672
	HyperConv	25.615	13.356	0.128		HyperConv	4.013	2.915	0.255
	AllSet	17.945	10.156	0.759		AllSet	3.057	2.294	0.674
	HMPNN	21.868	12.633	0.611		HMPNN	3.246	2.446	0.624
	HNHN	17.735	10.094	0.767		HNHN	3.208	2.377	0.637
	Lin. Transformers	28.807	14.583	0.119		Lin. Transformers	4.090	2.913	0.154
	NetlistGNN	20.238	11.696	0.697		NetlistGNN	2.882	2.173	0.726
	base DE-HNN	19.935	11.227	0.694		base DE-HNN	2.990	2.228	0.696
	full DE-HNN	**16.946**	**9.680**	**0.790**		**full DE-HNN**	**2.855**	**2.136**	**0.730**
	Improvement	4.4%	4.1%	3.0%		**Improvement**	0.9%	1.7%	0.6%
Superblue7	GCN	4.243	3.064	0.600	Superblue19	GCN	5.034	3.734	0.616
	GATv2	4.403	3.247	0.561		GATv2	4.949	3.691	0.636
	HyperConv	4.689	3.327	0.225		HyperConv	5.746	3.974	0.312
	AllSet	4.201	2.991	0.621		AllSet	4.682	3.474	0.685
	HMPNN	4.527	3.246	0.541		HMPNN	5.294	3.980	0.571
	HNHN	4.458	3.165	0.557		HNHN	5.063	3.750	0.620
	Lin. Transformers	5.245	3.752	0.139		Lin. Transformers	6.315	4.565	0.127
	NetlistGNN	4.115	2.986	0.634		NetlistGNN	4.683	3.520	0.681
	base DE-HNN	4.110	2.957	0.631		base DE-HNN	4.946	3.720	0.632
	full DE-HNN	**3.971**	**2.860**	**0.662**		**full DE-HNN**	**4.429**	**3.317**	**0.723**
	Improvement	3.5%	4.2%	4.4%		**Improvement**	5.4%	4.5%	5.5%

Table 10: Results of single-design net-based demand regression for each design.

Design	Model	Precision ↑	Recall ↑	F_score ↑
Superblue1	GCN	0.839	0.944	0.888
	GATv2	0.867	0.944	0.904
	HyperConv	0.873	0.966	0.917
	AllSet	0.880	0.955	0.916
	HMPNN	0.866	0.968	0.916
	HNHN	0.868	0.969	0.916
	Lin. Transformers	0.853	0.941	0.895
	NetlistGNN	0.862	0.936	0.920
	base DE-HNN	0.876	0.967	0.920
	full DE-HNN	**0.885**	**0.969**	**0.925**
	Improvement	1.6%	0.5%	0.5%
Superblue2	GCN	0.741	0.657	0.697
	GATv2	0.782	0.739	0.760
	HyperConv	0.779	0.706	0.741
	AllSet	0.727	0.664	0.694
	HMPNN	0.730	0.587	0.649
	HNHN	0.718	0.633	0.670
	Lin. Transformers	0.752	0.530	0.621
	NetlistGNN	0.765	0.614	0.682
	base DE-HNN	0.796	0.717	0.755
	full DE-HNN	**0.797**	**0.767**	**0.782**
	Improvement	1.2%	9.7%	2.9%
Superblue3	GCN	0.731	0.837	0.780
	GATv2	0.768	**0.840**	0.798
	HyperConv	0.770	0.815	0.792
	AllSet	0.728	0.773	0.747
	HMPNN	0.711	0.777	0.739
	HNHN	0.706	0.777	0.737
	Lin. Transformers	0.749	0.757	0.753
	NetlistGNN	0.786	0.814	0.799
	base DE-HNN	0.791	0.819	0.805
	full DE-HNN	**0.817**	0.816	**0.816**
	Improvement	0.7%	-	2.1%
Superblue5	GCN	0.745	0.932	0.834
	GATv2	0.783	0.923	0.848
	HyperConv	0.827	0.935	0.878
	AllSet	0.795	0.939	0.861
	HMPNN	0.788	0.932	0.853
	HNHN	0.786	0.932	0.852
	Lin. Transformers	0.798	0.911	0.851
	NetlistGNN	0.844	0.933	0.885
	base DE-HNN	0.842	0.938	0.887
	full DE-HNN	**0.852**	**0.940**	**0.894**
	Improvement	4.9%	1.3%	1.0%
Superblue6	GCN	0.837	0.921	0.877
	GATv2	0.876	0.920	0.897
	HyperConv	0.851	0.916	0.891
	AllSet	0.817	0.940	0.874
	HMPNN	0.809	**0.965**	0.879
	HNHN	0.815	0.947	0.875
	Lin. Transformers	0.833	0.906	0.868
	NetlistGNN	0.819	0.928	0.889
	base DE-HNN	0.859	0.928	0.892
	full DE-HNN	**0.885**	0.930	**0.906**
	Improvement	0.7%	-	1.0%
Superblue7	GCN	0.839	0.980	0.904
	GATv2	0.863	**0.981**	0.920
	HyperConv	0.899	0.967	0.932
	AllSet	0.888	0.956	0.921
	HMPNN	0.874	0.962	0.916
	HNHN	0.875	0.955	0.913
	Lin. Transformers	0.792	0.870	0.891
	NetlistGNN	0.868	0.918	0.923
	base DE-HNN	0.900	0.938	0.887
	full DE-HNN	**0.908**	0.969	**0.937**
	Improvement	5.6%	-	0.5%

Design	Model	Precision ↑	Recall ↑	F_score ↑
Superblue9	GCN	0.684	0.556	0.613
	GATv2	0.719	0.613	0.666
	HyperConv	0.716	0.605	0.656
	AllSet	0.675	0.505	0.577
	HMPNN	0.612	0.418	0.495
	HNHN	0.664	0.447	0.529
	Lin. Transformers	0.649	0.551	0.596
	NetlistGNN	**0.778**	0.568	0.656
	base DE-HNN	0.740	0.647	0.690
	full DE-HNN	0.695	**0.653**	**0.673**
	Improvement	-	6.5%	1.1%
Superblue11	GCN	0.634	0.896	0.743
	GATv2	0.706	0.844	0.769
	HyperConv	0.644	0.910	0.755
	AllSet	0.620	**0.946**	0.749
	HMPNN	0.627	0.927	0.748
	HNHN	0.628	0.897	0.738
	Lin. Transformers	0.671	0.691	0.680
	NetlistGNN	0.691	0.914	0.787
	base DE-HNN	0.677	0.863	0.759
	full DE-HNN	**0.719**	0.850	**0.789**
	Improvement	1.1%	-	0.3%
Superblue14	GCN	0.763	0.891	0.822
	GATv2	0.834	0.889	0.860
	HyperConv	0.830	0.886	0.857
	AllSet	0.809	0.863	0.835
	HMPNN	0.819	0.858	0.838
	HNHN	0.800	0.855	0.826
	Lin. Transformers	0.735	0.764	0.749
	NetlistGNN	0.827	0.870	0.787
	base DE-HNN	**0.856**	0.876	0.866
	full DE-HNN	**0.856**	**0.902**	**0.878**
	Improvement	3.7%	10.6%	2.1%
Superblue16	GCN	0.713	0.926	0.807
	GATv2	0.864	**0.928**	0.894
	HyperConv	0.855	0.833	0.844
	AllSet	0.844	0.827	0.833
	HMPNN	0.838	0.821	0.829
	HNHN	0.831	0.819	0.822
	Lin. Transformers	0.810	0.831	0.815
	NetlistGNN	0.779	0.912	0.865
	base DE-HNN	0.874	0.823	0.847
	full DE-HNN	**0.895**	0.910	**0.903**
	Improvement	8.5%	-	1.0%
Superblue18	GCN	0.779	0.845	0.811
	GATv2	0.798	0.848	0.822
	HyperConv	0.790	0.888	0.836
	AllSet	0.753	0.892	0.816
	HMPNN	0.763	0.888	0.821
	HNHN	0.749	0.881	0.810
	Lin. Transformers	0.775	0.852	0.812
	NetlistGNN	**0.868**	**0.939**	**0.902**
	base DE-HNN	0.798	0.890	0.842
	full DE-HNN	0.807	0.886	0.845
	Improvement	-	-	-
Superblue19	GCN	0.812	0.894	0.851
	GATv2	0.857	0.899	0.878
	HyperConv	0.879	0.877	0.878
	AllSet	0.870	0.866	0.868
	HMPNN	0.861	0.862	0.862
	HNHN	0.865	0.835	0.848
	Lin. Transformers	0.809	0.880	0.843
	NetlistGNN	0.869	**0.946**	0.856
	base DE-HNN	0.883	0.885	0.884
	full DE-HNN	**0.895**	0.910	**0.903**
	Improvement	0.4%	-	2.8%

Table 11: Results of single-design cell-based congestion classification for each design.

	Single-Design without placement			Single-Design with placement			Cross-Design without placement			Cross-Design with placement		
Model	RMSE	MAE	Pearson	RMSE (imp.)	MAE (imp.)	Pearson (imp.)	RMSE	MAE	Pearson	RMSE (imp.)	MAE (imp.)	Pearson (imp.)
GCN	5.034	3.734	0.616	4.495 (10.7%)	3.334 (10.7%)	0.717 (16.4%)	6.571	5.024	0.365	6.126 (6.8%)	4.709 (6.3%)	0.440 (20.5%)
GATv2	4.949	3.691	0.636	4.382 (11.4%)	3.112 (15.7%)	0.758 (19.2%)	6.623	5.137	0.363	5.812 (10.8%)	4.695 (8.6%)	0.442 (21.8%)
full DE-HNN	4.429	3.317	0.723	4.005 (9.6%)	2.987 (10.0%)	0.785 (8.6%)	6.037	4.670	0.372	5.795 (4.0%)	4.337 (7.1%)	0.452 (21.5%)

Table 12: Results of net-based demand regression for **Superblue19**. For each metric, the **(imp.)** refers to the improvements when placement information added.

		Wirelength Regression			**Demand Regression**			**Congestion Classification**		
Design	**Model**	**RMSE** ↓	**MAE** ↓	**Pearson** ↑	**RMSE** ↓	**MAE** ↓	**Pearson** ↑	**Precision** ↑	**Recall** ↑	**F_score** ↑
Superblue19	GCN	1.691	1.276	0.746	6.571	5.024	0.365	0.633	0.997	0.773
	GATv2	1.717	1.281	0.737	6.623	5.137	0.363	0.630	0.999	0.765
	Lin. Transformer	2.159	1.588	0.521	6.564	4.819	0.086	0.618	0.859	0.772
	NetlistGNN	1.762	1.324	0.718	8.328	6.839	0.367	0.647	0.953	0.771
	HyperConv	2.390	1.788	0.558	8.569	5.294	0.241	0.655	0.923	0.778
	Allset	1.837	1.348	0.695	6.120	4.820	0.345	0.645	0.964	0.773
	HMPNN	1.785	1.335	0.710	6.979	5.356	0.306	0.633	0.999	0.773
	HNHN	1.754	1.333	0.701	6.390	4.870	0.358	0.648	0.939	0.767
	base DE-HNN	1.731	1.291	0.730	6.778	5.085	0.337	0.653	0.990	0.774
	full DE-HNN	1.677	1.242	0.754	6.037	4.670	0.372	0.660	0.986	0.780
	Improvement	1.9%	2.6%	1.8%	1.4%	4.1%	1.4%	0.7%	-	0.3%

Table 13: Results of cross-design net-based hpwl(wirelength) regression, net-based demand regression and cell-based congestion classification for different netlist design, including comparisons with other HNN models.