# Net$^2$: A Graph Attention Network Method Customized for Pre-Placement Net Length Estimation

Zhiyao Xie[†], Rongjian Liang[§], Xiaoqing Xu[‡], Jiang Hu[§], Yixiao Duan[†], Yiran Chen[†]

[†] Duke University, Durham, NC, USA
[‡] ARM Inc., Austin, TX, USA
[§] Texas A&M University, College Station, TX, USA
zhiyao.xie@duke.edu    yiran.chen@duke.edu

## ABSTRACT

Net length is a key proxy metric for optimizing timing and power across various stages of a standard digital design flow. However, the bulk of net length information is not available until cell placement, and hence it is a significant challenge to explicitly consider net length optimization in design stages prior to placement, such as logic synthesis. This work addresses this challenge by proposing a graph attention network method with customization, called Net$^2$, to estimate individual net length before cell placement. Its accuracy-oriented version Net$^{2a}$ achieves about 15% better accuracy than several previous works in identifying both long nets and long critical paths. Its fast version Net$^{2f}$ is more than 1000× faster than placement while still outperforms previous works and other neural network techniques in terms of various accuracy metrics.

## 1 INTRODUCTION

For state-of-the-art semiconductor manufacturing technology nodes, interconnect is a dominating factor for integrated circuit (IC) performance and power, e.g., it can contribute to over 1/3 of clock period [10] and about 1/2 of total chip dynamic power [18]. Interconnect characteristics are affected by almost every step in a design flow, but not explicitly quantified and optimized until the layout stage. Therefore, previous academic studies attempted to address the interconnect effect in design steps prior to layout, e.g., layout-aware synthesis [20, 21]. Moreover, a recent industrial trend among commercial implementation tools [2, 23] takes an ambitious goal to explicitly address the interaction between logic synthesis and [illegible]t. To achieve such a goal, an essential element is to enable fast [illegible]curate pre-layout net length prediction, which has received [illegible]icant research attention in the past[1, 5, 9–11, 15, 16]. Some [illegible] [1, 5] pre-define numerous features describing each net, then a polynomial model is built by fitting these features. The work of [15] estimates wirelength by artificial neural networks (ANN), but it is limited to the total wirelength on an FPGA only, which is easier to estimate than individual net length. The mutual contraction (MC) [9] estimates net length by checking the number of cells in every neighboring net. The intrinsic shortest path length (ISPL) [11] is an interesting heuristic, which finds the shortest path between cells in the net to be estimated, apart from the net itself. The idea in [16] is similar to [11] in measuring the graph distance between cells in the netlist. The recent work [10] on wirelength prediction can only estimate the wirelength of an entire path instead of individual nets, and it relies on the results from virtual placement and routing.

Although net length prediction has been extensively studied previously, we notice a major limitation in most works. That is, they only focus on the local topology around each individual net with an over-simplified model. In other words, when estimating each net, usually their features only include information from nets one or two-hop away. The big picture, which is the net's position in the whole netlist, is largely absent. However, a placer normally optimizes a cost function defined on the whole netlist. It is not likely to achieve high accuracy without accessing any global information. Some previous models indeed attempt to embrace global information like the number of 2-pin nets in an entire circuit [1, 5], or a few shortest paths [11], but such information is either too sketchy [1, 5] or still limited to a region of several hops [11]. Since the global or long-range impact on individual nets is much more complex than local circuit topologies, it can hardly be captured by simple models or models with only human-defined parameters that cannot learn from data. To solve this, we propose a new approach, called Net$^2$, based on graph attention network [24]. Its basic version, Net$^{2f}$, intends to be fast yet effective. The other version, which emphasizes more on accuracy and is denoted as Net$^{2a}$, captures rich global information with a highly flexible model through circuit partitioning.

Recently, deep learning has generated a huge impact on many applications where data is represented in Euclidean space. However, there is a wide range of applications where data is in the form of graphs. Machine learning on graphs is much more challenging as there is no fixed neighborhood structure like in images. All neural network-based methods on graphs are referred to as graph neural networks (GNN). The most widely-used GNN methods include graph convolution network (GCN) [13], graphSage (GSage) [7], and graph attention network (GAT) [24]. They all convolve each node's representation with its neighbors' representations, to derive an updated representation for the central node. Such operation essentially propagates node information along edges and thereby topology pattern is learned.

Similarly, in Electronic Design Automation (EDA), circuit designs are embedded in Euclidean space after placement, which inspired many convolutional neural network (CNN)-based methods [4, 25, 26]. But before placement, a circuit structure is described as a graph and spatial information is not yet available. Till recent years, GNN is explored for EDA applications [17, 27]. The work in [17] predicts

*ASPDAC '21, January 18–21, 2021, Tokyo, Japan*
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-7999-1/21/01…$15.00
https://doi.org/10.1145/3394885.3431562

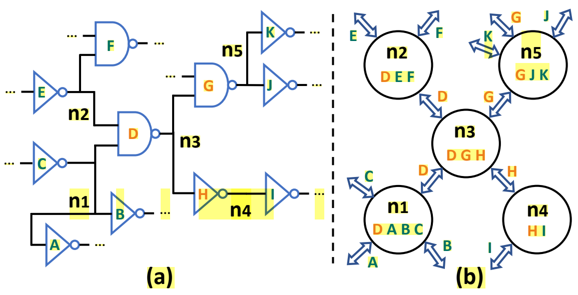

**Figure 1: (a) Part of a netlist. (b) The corresponding graph.**

observation point candidates with a model similar to GSage [7]. Graph-CNN [27] predicts the electromagnetic properties of post-placement circuits. This method is limited to very small-scale circuit graphs with less than ten nodes. Overall, GNN has great potential but is much less studied than CNN in EDA.

In this work, our contributions include:

- As far as we know, this is the first work making use of GNN for pre-placement net length estimation. GNNs (GCN, GSage, GAT) outperform both conventional heuristics and common machine learning methods in almost all measured metrics.
- We propose to extract global topology information through partitioning. Based on partition results, we define innovative directional edge features between nets, which substantially contribute to Net$^{\mathbf{2}}$'s superior accuracy.
- We propose a GAT-based model named Net$^{\mathbf{2}}$, which is customized for this net length problem. It includes a fast version Net$^{\mathbf{2f}}$, which is 1000× faster than placement, and an accuracy-centric version Net$^{\mathbf{2a}}$, which effectively extracts global topology information from unseen netlists and significantly outperforms plug-in use of existing GNN techniques.
- To focus on nets, we propose a graph construction method that treats nets as nodes. In designing Net$^{\mathbf{2}}$ architecture, we define different convolution layers for graph nodes and edges to incorporate both edge and node features.
- Besides net length, we further apply the proposed methods to estimate path length. Net$^{\mathbf{2}}$ also proves to be superior in identifying long paths.

## 2 PROBLEM FORMULATION

We define terminologies with Figure 1. Figure 1(a) shows part of a netlist, including five nets $\{n_1, n_2, n_3, n_4, n_5\}$ and 11 cells $\{c_A, c_B,$ ..., $c_K\}$. Now we focus on net $n_3$, which touches 3 cells $\{c_D, c_G, c_H\}$ and is referred to as a *3-pin net*. Its *driver* is cell $c_D$; its *sinks* are cells $\{c_G, c_H\}$. We denote the area of $n_3$'s driver cell as $a^3_{dri}$. Net $n_3$'s *fan-ins* $N^3_{in} = \{n_1, n_2\}$ ; its *fan-outs* $N^3_{out} = \{n_4, n_5\}$. Its *fan-in size* is 2, denoted as $|N^3_{in}| = 2$. Its *fan-out size* (number of sinks) is 2, denoted as $|N^3_{out}| = 2$. Every net can have only one driver but multiple sinks. Thus, the number of cells = $1 + |N^3_{out}| = 3$ for this net. Net $n_3$'s one-hop neighbors include both its fan-in and fan-out: $\mathcal{N}(n_3) = N^3_{in} \cup N^3_{out} = \{n_1, n_2, n_4, n_5\}$. The number of its neighbors is also known as the degree of $n_3$: $deg(n_3) = |\mathcal{N}(n_3)| = 4$.

The net length is the label for training and prediction. Each net's length is the half perimeter wirelength (HPWL) of the bounding box of the net after placement. The length of net $n_k$ is $L^k$.

To apply graph-based methods, we convert each netlist to one directed graph. Different from most GNN-based EDA tasks, net length

**Table 1: Commonly Used Notations**

| Notation | Description | Notaion | Description |
|---|---|---|---|
| $n_k$ | net or node | $O_k$ | node features |
| $\{c_G, c_H\}$ | cells | $E_{b\rightarrow k}$ | edge features |
| $a^k_{dri}$ | driver's area | $P[c_H]$ | cell partition IDs |
| $N^k_{in}, N^k_{out}$ | fan-in/fan-out nets | $M[n_k]$ | node partition IDs |
| $\mathcal{N}(n_k)$ | 1-hop neighbors | $h^{(t-1)}_k, h^{(t)}_k$ | GNN embeddings |
| $\|N^k_{in}\|, \|N^k_{out}\|$ | fan-in/fan-out size | $W^{(t)}, \theta^{(t)}$ | learnable weights |
| $deg(n_k)$ | degree of net | $g(\,), \sigma(\,)$ | activation functions |
| $L^k$ | net length label | $s(\,), \mu(\,)$ | std deviation, mean |
| $n_b \rightarrow n_k$ | directional edge | $[\ \|\|\ ]$ | concat to one list |
| $c_{kb}$ or $c_{bk}$ | cell on $n_b \rightarrow n_k$ | $\backslash$ | exclude from list |

prediction focuses on nets rather than cells. Thus we represent each net as a node, and use the terms *node* and *net* interchangeably. For each net $n_k$, it is connected with its fan-ins and fan-outs through their common cells by edges in both directions. The common cell shared by both nets on that edge is called its *edge cell*. For example, in Figure 1(b), net $n_3$ is connected with nets $n_4$ and $n_5$ through its sinks $c_G$ and $c_H$; it is connected with nets $n_1$ and $n_2$ through its driver $c_D$. The edges through edge cell $c_G$ is denoted as $n_3 \rightarrow n_5$ and $n_5 \rightarrow n_3$. The edge cell $c_G$ can also be referred to as $c_{35}$ or $c_{53}$. We differentiate edges in different directions because we will assign different edge features to $n_3 \rightarrow n_5$ and $n_5 \rightarrow n_3$.

An important concept throughout this paper is global and local topology information. We use the number of hops to denote the shortest graph distance between two nodes on a graph. The *information* of each net refers to its number of cells and driver's area. Local information includes the information about the estimated net itself, or from its one to two-hop neighboring nets. In contrast, global information means the pattern behind the topology of the whole netlist or the information from nets far away from the estimated net $n_k$. The range of neighbors that can be accessed by each model is referred to as the model's *receptive field*.

## 3 CHALLENGES

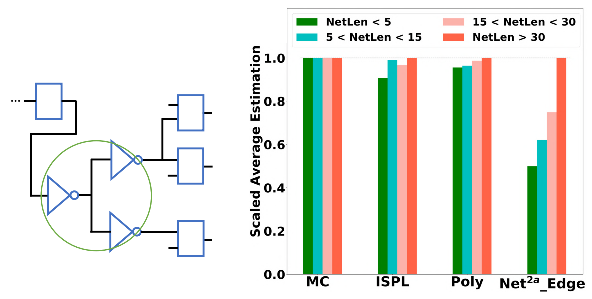

**Figure 2: (a) A typical local topology. (b) Predictions on nets with similar local topology information.**

We provide an example to show the challenge in net length prediction and the importance of global information. Figure 2(a) shows a net $n_6$ with a commonly seen local topology information: $|N^6_{in}| = 1$, $|N^6_{out}| = 2$, $deg(n_6) = 3$. For its neighbors, it has one 2-pin fan-in, one 2-pin fan-out and one 3-pin fan-out.

In a netlist of design B20 in ITC 99 [3], we find 725 nets with exactly the same driver cell's area, number of cells and one-hop neighbor information as $n_6$, but their net lengths after placement range from 1 µm to more than 100 µm. Distinguishing these similar nets is highly challenging without rich global information. To prove this, Figure 2(b) shows the prediction from different methods on these 725 similar nets. These nets are firstly divided into four different types according to their actual net length, ~~each type with 432, 190, 67, 36 nets,~~ respectively. We then plot the scaled averaged estimation by different methods for each type of net. MC [9], which only looks at one-hop neighbors, cannot distinguish these nets at all. ISPL [11], which captures some global information by searching shortest path, gives a slightly lower estimation on the shortest type (netLen < 5 µm). By looking at two-hop neighbors, a polynomial model with pre-defined features (Poly) [1] [5] captures the trend with a tiny difference between different types. For Net$^2$, we only train its edge convolution layer on other designs and present its output. Its estimations on different net types differ significantly. This example shows the importance of global information in distinguishing a large number of nets with similar local information.

## 4 ALGORITHM

### 4.1 Node Features on Graph

Algorithm 1 shows how we build a directed graph and generate features for each node with a given netlist. On average, a net with more large cells tends to be longer. Thus, the most basic net features include the net's driver's area, fan-in and fan-out size $\{|N^k_{in}|, |N^k_{out}|, a^k_{dri}\}$. Feature $\sum a_{all}$ is the sum of areas over all cells in $n_k$. It is calculated by including the drivers of all $n_k$'s fan-outs in line 6, which are the sinks of $n_k$. Besides these basic features, we capture the more complex impact from neighbors. As shown in lines 3, we go through all neighbors of $n_k$ to collect their fan-in and fan-out sizes. The summation $\sum$ and standard deviation $s()$ of these neighboring information are added to node features $O_k$ in line 7.

### 4.2 Edge Features

In Algorithm 1, node features $O_k$ include up to two-hop neighboring information. The receptive field of the GNN method itself depends

**lgorithm 1** Graph Generation with Node Features

**put**: Basic features $\{|N^k_{in}|, |N^k_{out}|, a^k_{dri}\}$, net length label $L^k$, the n-in nets $N^k_{in}$ and fan-out nets $N^k_{out}$ of each net $n_k$.

**Generate Node Features**:

```
1: for each net $n_k$ do
2:     $a_{all} = [a^k_{dri}]$, $in_{in} = []$, $in_{out} = []$, $out_{in} = []$, $out_{out} = []$
3:     for each net $n_i \in N^k_{in}$, each net $n_o \in N^k_{out}$ do
4:         $in_{in}$.add $(|N^i_{in}|)$ ;   $in_{out}$.add $(|N^i_{out}|)$
5:         $out_{in}$.add $(|N^o_{in}|)$ ; $out_{out}$.add $(|N^o_{out}|)$
6:         $a_{all}$.add$(a^o_{dri})$
7:     $O_k = \{|N^k_{in}|, |N^k_{out}|, a^k_{dri}, \sum a_{all}, \sum out_{in}, \sum out_{out},$
           $\sum in_{in}, \sum in_{out}, s(out_{in}), s(out_{out}), s(in_{in}), s(in_{out})\}$
```

**ild Graph**:

```
    Initiate a graph $G$. Each net is a node.
    for each net $n_k$ do
        For node $n_k$ in $G$, set $O_k$ as node feature, $L^k$ as label.
4:      for each net $n_b \in N^k_{in} \cup N^k_{out}$ do
5:          Add directed edge $n_b \rightarrow n_k$.
```

**Output**: Graph $G$ with node features $O$ and label $L$.

**Algorithm 2** Define Edge Features on Graph

**Input**: Cell cluster ID $P[c_k]$ for each cell $c_k$, net cluster ID $M[n_k]$ and the neighbors $\mathcal{N}(n_k)$ of each net $n_k$. Directed graph $G$.

```
1: function MEASUREDIFF($c_{bk}, n_b, c_{ok}, n_o$)
2:     $f_0 = 1 - (P[c_{bk}] == P[c_{ok}])$
3:     $P_b = [P[c]$ for $c \in n_b]$ // cluster IDs for $n_b$'s cells
4:     $P_o = [P[c]$ for $c \in n_o]$ // cluster IDs for $n_o$'s cells
5:     $P_{b\_not\_o} = P_b \setminus P_o$        // IDs in $P_b$ but not in $P_o$
6:     $P_{o\_not\_b} = P_o \setminus P_b$        // IDs in $P_o$ but not in $P_b$
7:     $f_1 = \frac{|P_{b\_not\_o}|}{|P_b|} + \frac{|P_{o\_not\_b}|}{|P_o|}$ // percent of different IDs
8:     $f_2 = 1 - (M[n_b] == M[n_o])$
9:     return $[f_0, f_1, f_2]$
10: end function
11:
12: for each net $n_k$ do
13:     for each net $n_b \in \mathcal{N}(n_k)$ do
14:         $F_0 = [], F_1 = [], F_2 = []$
15:         Cell $c_{bk}$ is the edge cell on $n_b \rightarrow n_k$
16:         Other neighbors $N^k_{other} = \mathcal{N}(n_k) \setminus \{n_b\}$
17:         for each net $n_o \in N^k_{other}$ do
18:             Cell $c_{ok}$ is the edge cell on $n_o \rightarrow n_k$
19:             $f_0, f_1, f_2$ = MEASUREDIFF $(c_{bk}, n_b, c_{ok}, n_o)$
20:             $F_0$.add$(f_0)$ ;  $F_1$.add$(f_1)$ ;  $F_2$.add$(f_2)$
21:         $f_3 = 1 - (M[n_b] == M[n_k])$
22:         $E_{b\rightarrow k} = \{\sum F_0, \mu(F_0), \sum F_1, \mu(F_1), \sum F_2, \mu(F_2), f_3\}$
23:         Set $E_{b\rightarrow k}$ as the feature of edge $n_b \rightarrow n_k$ in $G$.
```

**Output**: Graph $G$ with edge features $E$.

on the model depth, which is usually two to three layers. Thus the model can reach as far as four to five-hop neighbors, which is already more than previous works. This work goes beyond that to capture more global information from the whole graph.

To capture global information, we use an efficient multi-level partitioning method hMETIS [12] to divide one netlist into multiple clusters / partitions. The partition method minimizes the overall cut between all clusters, which provides a global perspective. We denote the partition result as $M$. Each net $n_k$ is assigned a cluster ID $M[n_k]$, which denotes the cluster it belongs to. To generate more information, on the same netlist, we also build a hyper-graph $HG_c$ by using cells as nodes. In $HG_c$, each hyper-edge corresponds to a net. Similarly, the partition result on $HG_c$ is denoted as $P$. Each cell $c_k$ is assigned a cluster ID $P[c_k]$. Notice that $HG_c$ is only used to generate cluster ID for each cell.

Cluster IDs are not directly useful by themselves. What matters in this context is the difference in cluster IDs between cells and nets. Algorithm 2 shows how the cluster information is incorporated into GNN models through novel edge features $F_0, F_1, F_2, f_3$. The most important intuition behind this is: for a high-quality placement solution, on average, the cells assigned to different clusters tend to be placed far away from each other.

In Algorithm 2, we design the edge features to quantify the source node's contribution to the target node's length. The contribution here means the source net is "pulling" the edge cell far away from other cells in the target net. The edge features measure such "pulling" strength. When the edge cell is "pulled" away, the target net results in a longer length. In Algorithm 2, for edge $n_b \rightarrow n_k$, function MEASUREDIFF measures the difference in assigned clusters

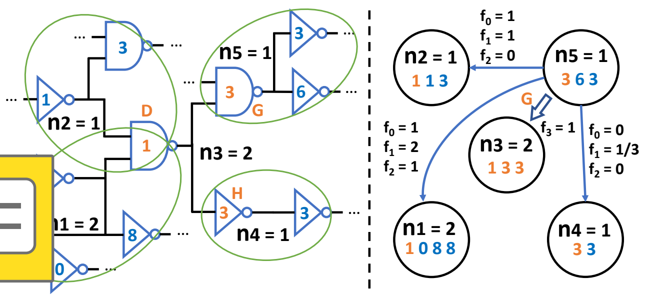

**Figure 3: Define edge features by partition results.**

between node $n_b$ and every other neighboring node $n_o$, which indicates the distance between $c_{bk}$ and $c_{ok}$. If the distance between edge cell $c_{bk}$ and every other cell $c_{ok}$ in $n_k$ is large, it means $c_{bk}$ is placed far away from other cells in net $n_k$. In this case, edges features $F_0, F_1, F_2, f_3$ are large. That is why edge features imply how strong the edge cell is "pulled" away from the target node.

Figure 3 shows an example of Algorithm 2 using the netlist same as Figure 1. The number on each cell or net is the cluster ID assigned to it after partition. Figure 3 measures the edge features of edge $n_5 \longrightarrow n_3$, representing how strongly edge cell $c_G$ is pulled by $n_5$ from both cells $\{c_D, c_H\}$ in $n_3$. To calculate this, we measure the distance between $c_G$ and $c_H$ by MEASUREDIFF($c_G$, $n_5$, $c_H$, $n_4$) in Algorithm 2; and the distance between $c_G$ and $c_D$ by MEASUREDIFF($c_G$, $n_5$, $c_D$, $n_1$) and MEASUREDIFF($c_G$, $n_5$, $c_D$, $n_2$).

Take MEASUREDIFF($c_G$, $n_5$, $c_H$, $n_4$) as an example to show how it measures distance between $c_G$ and $c_H$. As shown in line 2, feature $f_0$ measures the difference in $c_G$ and $c_H$' cluster IDs, $f_0 = 1 - (P[c_G] == P[c_H]) = 1 - (3 == 3) = 0$. Feature $f_1$ measures the difference in all cells between $n_5$ and $n_4$. As shown from line 3 to 7, $P_5 = [3, 6, 3]$ and $P_4 = [3, 3]$. Then $P_{5\_not\_4} = [6]$ and $P_{4\_not\_5} = []$. They are normalized by the number of cells $|P_5| = 3$ and $|P_4| = 2$, in order to avoid bias toward nets with many cells. Thus, $f_1 = \frac{1}{3} + \frac{0}{2} = \frac{1}{3}$. Feature $f_2$ measures the difference between $n_5$ and $n_4$, $f_2 = 1 - (M[n_5] == M[n_4]) = 1 - (1 == 1) = 0$. As this example shows, we only measure whether cells / nets have the same cluster IDs, and the order of IDs does not matter.

After measuring the difference in cluster ID between $c_G$ and all other cells in $n_3$, for the edge $n_5 \longrightarrow n_3$, $F_0 = [1, 1, 0]$; $F_1 = [2, 1, \frac{1}{3}]$; $F_2 = [1, 0, 0]$. $f_3$ measures the difference between $n_5$ and $n_3$, $f_3 = 1$. This example shows how we incorporate global information from partition into edge features. Actually, we generate multiple different partitioning results $M$, $P$ by requesting different number of clusters. That results in multiple different $\{F_0, F_1, F_2, f_3\}$. All these different edge features are processed in line 22 and concatenated together as the final edge features $E_{b \to k}$.

## 4.3 Common GNN Models

This section introduces how GNN models are applied on the graph G we build. GNN models are comprised of multiple sequential convolution layers. Each layer generates a new embedding for every node based on the previous embeddings. For node $n_k$ with node features $O_k$, denote its embedding at the $t^{\text{th}}$ layer as $h_k^{(t)}$. Its initial embedding is the node features $h_k^{(0)} = O_k$. Sometimes the operation includes both neighbours and the node itself, we use $n_\beta$ to denote it: $n_\beta \in \mathcal{N}(n_k) \cup \{n_k\}$. In each layer $t$, GNNs calculate the updated embedding $h_k^{(t)}$ based on the previous embedding of the node itself $h_k^{(t-1)}$ and its neighbors $h_b^{(t-1)} | n_b \in \mathcal{N}(n_k)$.

We show one layer of GCN, GSage, and GAT below. Notice that there exists other expressions. The two-dimensional learnable weight at layer $t$ is $W^{(t)}$. In GAT, there is an extra one-dimensional weight $\theta^{(t)}$. The operation [ || ] concatenates two vectors into one longer vector. Functions $\sigma$ and $g$ are sigmoid and Leaky ReLu activation function, respectively.

On GCN (with self-loops), $\mathcal{F}_{GCN}^{(t)}$ [13] is:

$$h_k^{(t)} = \sigma\Big(\sum_{n_\beta \in \mathcal{N}(n_k) \cup \{n_k\}} a_{k\beta} W^{(t)} h_\beta^{(t-1)}\Big)$$

$$\text{where } a_{k\beta} = \frac{1}{\sqrt{deg(k)+1}\sqrt{deg(\beta)+1}} \in \mathbb{R}$$

On GSage, $\mathcal{F}_{GSage}^{(t)}$ [7] is:

$$h_k^{(t)} = \sigma\Big(W^{(t)} \Big[h_k^{(t-1)} \Big|\Big| \frac{1}{deg(k)} \sum_{n_b \in \mathcal{N}(n_k)} h_b^{(t-1)}\Big]\Big)$$

On GAT, $\mathcal{F}_{GAT}^{(t)}$ [24] is:

$$h_k^{(t)} = \sigma\Big(\sum_{n_\beta \in \mathcal{N}(n_k) \cup \{n_k\}} a_{k\beta} W^{(t)} h_\beta^{(t-1)}\Big)$$

$$\text{where } a_{k\beta} = softmax_\beta(r_{k\beta}) \quad \text{over } n_k \text{ and its neighbors},$$

$$r_{k\beta} = g\big(\theta^{(t)\intercal} [W^{(t)} h_\beta^{(t-1)} || W^{(t)} h_k^{(t-1)}]\big) \in \mathbb{R}$$

Here we briefly discuss the difference between these methods. GCN scales the contribution of neighbors by a pre-determined coefficient $a_{k\beta}$, depending on the node degree. In contrast, GAT uses learnable weights $W$, $\theta$ to firstly decide node $n_\beta$'s contribution $r_{k\beta}$, then normalize the coefficient $r_{k\beta}$ across $n_k$ and its neighbors through a softmax operation. Such a learnable $a_{k\beta}$ leads to a more flexible model. GSage does not scale neighbors by any factor. For all these GNN methods, the last layer 's output embedding $h_k^{(t)}$ is connected to a multi-layer ANN.

## 4.4 Net$^2$ Model

The node convolution layer of the Net$^2$ is based on GAT, considering its higher flexibility in deciding neighbors' contribution $a_{k\beta}$. Thus node convolution layer is $\mathcal{F}_{GAT}^{(t)}$. In the final embedding, we concatenate the outputs from all layers, instead of only using the output of the final layer like most GNN works. This is a customization, by which the embedding includes contents from different depths. The shallower one includes more local information, while the deeper one contains more global information. Such an embedding provides more information for the ANN model at the end and may lead to better convergence. The idea of combining shallow and deep layers has inspired many classical deep learning methods in Euclidian space [8] [22], but it is not widely applied in GNNs for node embeddings. After three layers of node convolution, the final embedding for each node is $[h_k^{(1)} || h_k^{(2)} || h_k^{(3)}]$. Without partitioning, this is the embedding for our fast solution Net$^{\mathbf{2f}}$.

In order to utilize edge features, here we define our own edge convolution layers $\mathcal{E}$ as customization. For each directed edge $n_b \longrightarrow n_k$, we concatenate both target and source nodes' features $[O_k || O_b]$ together with its edge features $E_{b \to k}$ as the input of edge convolution. Combining node features when processing edge features

**Table 2: Number of Cells in Netlists for Each Design**

| | B14 | B15 | B17 | B18 | B20 | B21 | B22 |
|---|---|---|---|---|---|---|---|
| Smallest | 13 K | 5.3 K | 18 K | 54 K | 26 K | 26 K | 39 K |
| Largest | 34 K | 15 K | 49 K | 138 K | 67 K | 66 K | 99 K |

enables $\mathcal{E}$ to distinguish different edges with similar edge features. The output embedding is:

$$e_{k\_sum} = \sum_{n_b \in \mathcal{N}(n_k)} W_2 W_1 [O_k || E_{b \to k} || O_b]$$

$$e_{k\_mean} = \frac{1}{deg(k)} e_{k\_sum}$$

The two two-dimensional learnable weights $W_1$ and $W_2$ can be viewed as applying a two-layer ANN to the concatenated input. We choose two-layer ANN rather than one-layer here because the input vector $[O_k || E_{b \to k} || O_b]$ is long and contains heterogeneous information from both edge and node. We prefer to learn from them with a slightly more complex function. After the operation, both $e_{k\_sum}$ and $e_{k\_mean}$ are on nodes. Then, we add an extra node convolution using the output from edge convolution as input. This structure learns from neighbors' edge embeddings $e_{b\_sum}, e_{b\_mean}$.

$$h_k^{(e)} = \mathcal{F}_{GAT}^{(e)}([e_{k\_sum} || e_{k\_mean}], [e_{b\_sum} || e_{b\_mean}])$$

pired by the same idea in Net$^{2f}$, we combine the contents from ers for our accurate solution Net$^{2a}$. Its final embedding is $_k ||h_k^{(2)} || h_k^{(3)} || e_{k\_sum} || e_{k\_mean} || h_k^{(e)}]$. For both Net$^{2f}$ and Net$^{2a}$, their final embeddings are then connected to an ANN.

## 5 EXPERIMENTAL RESULTS

### 5.1 Experimental Setup

Seven different designs from ITC99 [3] are synthesized with Synopsys Design Compilier® in 45nm NanGate Library, and then placed by Cadence Innovus™ v17.0. For each design, we synthesize 10 different netlists with different synthesis parameters. Altogether there are 70 different netlists and corresponding placement solutions. Table 2 shows the size of these netlists. When testing ML models on each design, the model is only trained on the 60 netlists from the *other* six designs to prevent information leakage. Then the result is averaged over the 10 netlists of the tested design.

All GNNs are built with Pytorch 1.5 [19] and Pytorch-geometric [6]. The partition on graphs is performed by hMETIS [12] executable files. The experiment is performed on a machine with a Xeon E5 processor and an Nvidia GTX 1080 graphics card.

Here we introduce the best hyper-parameters after parameter tuning. For all GNN methods, we use three layers of GNN with two layers ANN. The attention head number of GAT is two. The size of each node convolution output is 64. The size of edge convolution output is twice of the input size $[O_k || E_{b \to k} || O_b]$. The size of the first-layer ANN is the same as its input embedding, and the size of the second-layer ANN is 64. A batch normalization layer is applied after each GNN layer for better convergence. Because of the difference in graph size, each batch includes only one graph, and the training data is shuffled during training. We use stochastic gradient descent (SGD) with learning rate 0.002 and momentum factor 0.9 for optimization. GNN models converge in 250 epoches.

When partitioning each netlist, we generate seven different cell-based partitions $P$ by requesting the number of output clusters to be the number of cells divided by 100, 200, 300, 500, 1000, 2000, and 3000. Because different partitions are generated in parallel, the overall runtime depends on the slowest one. Similarly, we generate three net-based partitions $M$ by requesting the cluster number to be the number of nets divided by 500, 1000, and 2000. These cluster numbers are achieved by tuning during experiments, which provides good enough coverage over different cluster sizes.

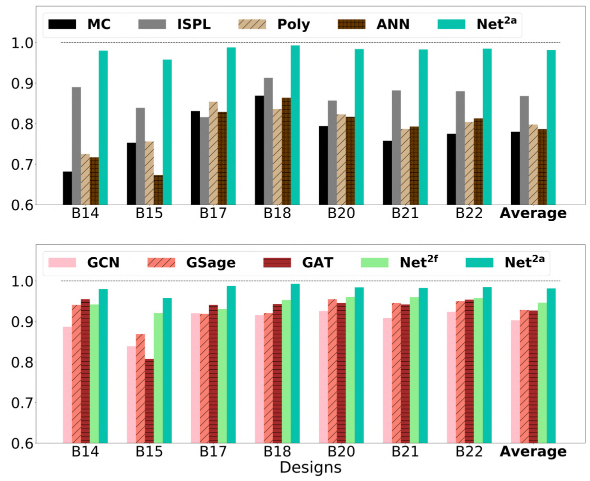

**Figure 4: Correlation coefficient R between prediction and label measured in 20 bins. Net$^{2a}$ is reported in both subplots.**

Representative previous methods MC [9], ISPL [11], and Poly [5] are implemented for comparisons. As for traditional ML models, we implement a three-layer ANN model using node features $O$.

Here we summarize the receptive field of all methods. MC is limited to one-hop neighbors, while Poly and ANN can reach two-hop neighbors. The receptive field of ISPL varies among different nodes. According to [11], ISPL for most nets is within several hops. In comparison, all three-layer GNNs and Net$^{2f}$ can access five-hop neighbors. Net$^{2a}$ measures the impact from the whole netlist.

### 5.2 Correlation on Net Length

To measure the correlation between prediction and ground truth, we apply a classical criterion used in many net length estimation works [5, 11, 16]. Firstly, we calculate a range of net length labels $[L_{0\%min}, L_{95\%max}]$. It means from the shortest net length (close to zero) to the 95 percentile largest net length. The top 5% longest nets are excluded to prevent an extraordinarily large range. Then, the calculated range is partitioned into 20 equal bins and the average of both predictions and labels in each bin are calculated. Figure 4 shows the correlation coefficient between these 20 averaged predictions and labels. Unlike some previous works, we define such 20 bins using labels instead of predictions for a fair comparison. In this way, the nets assigned to each bin are kept the same for different methods. On average, the correlation coefficient for GraphSage and GAT approaches 0.93. Net$^{2f}$ further improves it to around 0.95. In comparison, the best traditional method is ISPL, with correlation equals 0.87. The correlation for Net$^{2a}$ is higher than 0.98.

### 5.3 Identifying Long Nets

In practice, we are also interested in how different methods can identify those longer nets. Table 3 shows the accuracy in identifying the top 10% longest nets. For each netlist, the 10% longest nets

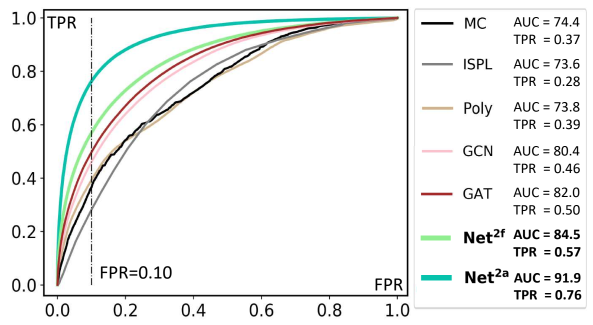

**Figure 5: ROC curves in identifying 10% longest nets.**

are labeled as true, and the accuracy is measured in ROC curve's area under curve (AUC). Then, the accuracy for each design is averaged over its 10 netlists. Since the net with many cells tends to be long, we add a baseline which directly estimates net length with the number of cells. On average, ANN, Poly and ISPL outperform MC and the number of cells with AUC~=0.76. Models capturing two or more-hop neighbors outperform models viewing one-hop neighbors only. Then graph methods (AUC~=0.82) are better than ANN by learning with a larger receptive field reaching five-hop neighbors. By combining shallow and deep embeddings, Net$^{\mathbf{2f}}$ outperforms normal GNN with AUC~=0.85. Net$^{\mathbf{2a}}$ significantly outperforms all graph methods by learning the global information on edge features with its edge convolution layer.

Figure 5 shows a visualization of the ROC curves for identifying the top 10% longest nets. A few less important methods are omitted for clarity. To visualize results from 70 netlists on one ROC curve, after prediction finishes, we put all nets from all 70 netlists together. Then the 10% longest nets are selected and the ROC curve is measured on all these nets. The vertical dashed line means when the false positive rate (FPR) equals 0.1. The AUC measured on all nets for 10% longest nets is fairly consistent with the average value shown in Table 3. On the right of Figure 5 also shows the true positive rate (TPR) for each method when FPR = 0.1. Both Net$^{\mathbf{2f}}$ and Net$^{\mathbf{2a}}$ significantly outperform previous works under all FPR.

## 5.4 Results on Path Length Estimation

In many EDA tools, the pre-placement timing report does not include wire delay, which hinders identifying critical paths accurately at an early stage. An application of net length estimation is to predict the length of any given path, which correlates with post-placement wire delay on the path. The path length is defined as the summation of the net lengths over all nets on this path.

**Table 3: Identify 10% Longest Nets in ROC AUC (%)**

| Methods | B14 | B15 | B17 | B18 | B20 | B21 | B22 | Ave |
|---|---|---|---|---|---|---|---|---|
| NumCell | 71.5 | 67.5 | 65.7 | 68.8 | 71.5 | 71.7 | 71.8 | 69.8 |
| MC | 73.1 | 70.3 | 71.3 | 72.8 | 75.0 | 74.3 | 74.8 | 73.1 |
| ISPL | 79.9 | 71.9 | 69.2 | 71.1 | 78.6 | 78.9 | 79.3 | 75.6 |
| Poly | 75.4 | 73.0 | 73.5 | 72.8 | 75.9 | 75.2 | 76.4 | 74.6 |
| ANN | 75.9 | 72.7 | 73.6 | 74.6 | 77.1 | 76.6 | 77.8 | 75.5 |
| GCN | 85.1 | 75.5 | 76.1 | 76.1 | 85.0 | 84.6 | 85.5 | 81.1 |
| GSage | 84.9 | 75.9 | 75.3 | 74.6 | 85.6 | 85.2 | 84.8 | 80.9 |
| GAT | 85.5 | 76.5 | 75.4 | 78.0 | 87.0 | 85.7 | 86.0 | 82.0 |
| Net$^{\mathbf{2f}}$ | 86.9 | 80.1 | 80.9 | 79.3 | 88.7 | 88.2 | 87.3 | 84.5 |
| Net$^{\mathbf{2a}}$ | 93.5 | 91.7 | 90.7 | 90.3 | 93.0 | 93.0 | 92.9 | 92.2 |

**Table 4: Identify 10% Longest Paths in ROC AUC (%)**

| Methods | B14 | B15 | B17 | B18 | B20 | B21 | B22 | Ave |
|---|---|---|---|---|---|---|---|---|
| ISPL | 58.9 | 57.5 | 56.5 | 74.0 | 72.5 | 63.0 | 75.5 | 65.4 |
| Poly | 65.5 | 80.0 | 78.0 | 68.0 | 82.0 | 85.0 | 84.0 | 77.5 |
| ANN | 68.0 | 76.0 | 80.0 | 69.0 | 78.5 | 82.0 | 75.5 | 75.6 |
| GCN | 63.5 | 75.0 | 86.5 | 56.0 | 82.0 | 81.5 | 85.5 | 75.7 |
| GSage | 65.0 | 88.0 | 93.0 | 77.0 | 81.5 | 67.0 | 80.0 | 78.8 |
| GAT | 63.0 | 92.0 | 95.0 | 83.5 | 83.5 | 76.0 | 89.5 | 83.2 |
| Net$^{\mathbf{2f}}$ | 79.0 | 88.5 | 97.5 | 84.0 | 75.5 | 83.0 | 92.0 | 85.6 |
| Net$^{\mathbf{2a}}$ | 86.5 | 95.0 | 96.0 | 90.5 | 90.5 | 93.5 | 95.5 | 92.5 |

**Table 5: Comparing Pair of Paths by Lengths (%)**

| Methods | B14 | B15 | B17 | B18 | B20 | B21 | B22 | Ave |
|---|---|---|---|---|---|---|---|---|
| ISPL | 67.1 | 55.0 | 58.2 | 77.4 | 68.9 | 59.7 | 69.5 | 65.1 |
| Poly | 83.9 | 86.6 | 83.3 | 70.4 | 83.4 | 80.4 | 86.3 | 82.0 |
| ANN | 82.0 | 74.8 | 75.3 | 68.1 | 81.9 | 65.4 | 80.5 | 75.4 |
| GCN | 74.5 | 85.9 | 83.0 | 62.4 | 83.4 | 81.0 | 86.2 | 79.5 |
| GSage | 84.2 | 92.5 | 83.9 | 75.3 | 89.1 | 62.8 | 88.1 | 82.3 |
| GAT | 82.4 | 93.5 | 85.1 | 80.6 | 89.7 | 87.5 | 88.2 | 86.7 |
| Net$^{\mathbf{2f}}$ | 87.3 | 92.7 | 87.6 | 93.1 | 91.1 | 91.2 | 86.9 | 90.0 |
| Net$^{\mathbf{2a}}$ | 96.8 | 97.0 | 91.4 | 95.9 | 92.2 | 94.2 | 94.4 | 94.6 |

To verify models' performance on path length, for each netlist, we collect the timing-critical paths according to the pre-placement timing report. The report at this stage only includes cell delay. The path's slack has to be negative. Among these paths, the longer ones would be even more timing-critical considering wire delay after placement. Thus we apply models to identify those longest critical paths. The estimated path length is a summation over the predicted net lengths among all nets on the path.

Table 4 shows the ROC curve accuracy in identifying 10% longest path. MC is not supposed to be *linearly* proportional to actual wire length; thus the summation over MC is not included as path length estimation. Because the number of critical paths is much less than the number of nets, the trend on accuracy is not very consistent on different designs. On average, we can still observe an obvious trend that Net$^{\mathbf{2a}}$ > Net$^{\mathbf{2f}}$ > GAT > other methods.

In addition to locating the longest paths, we are interested in how models recognize the longer path when comparing any pair of two paths. To avoid meaningless comparisons between paths with very similar lengths, for each path, we compare it with all paths 30% longer or shorter than it in its netlist. Table 5 shows the percentage of correct comparisons. Poly is the best method among previous works. The trend in accuracy is similar, Net$^{\mathbf{2a}}$ > Net$^{\mathbf{2f}}$ > GAT > other methods.

## 5.5 Runtime Comparison

Table 6 shows the runtime of placement and Net$^{\mathbf{2}}$. The runtime is averaged over all netlists. For a fair comparison, the runtime of placement includes the placement algorithm only, without any extra time for file I/O, floorplanning, or placement optimization. Net$^{\mathbf{2a}}$ takes slightly longer inference time than Net$^{\mathbf{2f}}$ for its extra edge convolution layer. The overall runtime of Net$^{\mathbf{2a}}$ includes both

**Table 6: Runtime Comparison (In seconds)**

| Design | Place | Partition | Net$^{\mathbf{2f}}$ Infer | Net$^{\mathbf{2a}}$ Infer | Net$^{\mathbf{2f}}$ Speedup | Net$^{\mathbf{2a}}$ Speedup |
|---|---|---|---|---|---|---|
| Ave | 97.8 | 7.0 | 0.05 | 0.07 | 1.7K × | 14.3× |

Table 7: Identify 10% Longest Nets in ROC AUC (%)

| Methods | B14 | B15 | B17 | B18 | B20 | B21 | B22 | Ave |
|---|---|---|---|---|---|---|---|---|
| Edge ANN | 90.6 | 83.9 | 85.5 | 87.7 | 89.9 | 89.6 | 90.5 | 88.2 |
| Simple Net | 92.3 | 89.6 | 88.9 | 89.7 | 91.6 | 91.7 | 91.7 | 90.8 |
| F0 Net | 92.4 | 90.8 | 89.7 | 89.0 | 92.0 | 91.7 | 91.6 | 91.0 |
| F1 Net | 93.0 | 91.4 | 90.3 | 89.8 | 92.6 | 92.3 | 92.5 | 91.7 |
| F2+f3 Net | 91.2 | 88.3 | 85.9 | 85.9 | 91.0 | 90.5 | 90.4 | 89.0 |
| Less $P$ Net | 92.9 | 91.2 | 89.7 | 89.6 | 92.7 | 92.5 | 92.3 | 91.6 |
| Net$^{2a}$ | 93.5 | 91.7 | 90.7 | 90.3 | 93.0 | 93.0 | 92.9 | 92.2 |

Table 8: Identify 10% Longest Paths in ROC AUC (%)

| Methods | B14 | B15 | B17 | B18 | B20 | B21 | B22 | Ave |
|---|---|---|---|---|---|---|---|---|
| Edge ANN | 77.0 | 91.0 | 94.5 | 88.0 | 85.0 | 89.5 | 94.0 | 88.4 |
| Simple Net | 83.5 | 93.0 | 94.5 | 89.5 | 87.5 | 92.0 | 94.5 | 90.6 |
| F0 Net | 77.0 | 89.5 | 95.0 | 90.0 | 90.0 | 87.5 | 93.5 | 88.9 |
| F1 Net | 81.5 | 93.0 | 95.5 | 90.5 | 89.5 | 87.5 | 95.0 | 90.4 |
| F2+f3 Net | 82.0 | 92.5 | 97.0 | 85.5 | 87.5 | 93.5 | 93.5 | 90.2 |
| Less $P$ Net | 84.5 | 94.0 | 96.5 | 88.0 | 91.0 | 94.0 | 95.5 | 91.9 |
| Net$^{2a}$ | 86.5 | 95.0 | 96.0 | 90.5 | 90.5 | 93.5 | 95.5 | 92.5 |

partition and inference. Partitioning takes the majority of the runtime. Net$^{2a}$ is more than 10× faster than placement. The runtime of Net$^{2a}$ can be potentially improved by using coarser but faster partition $P$ and $M$, especially on larger designs. Without partition, Net$^{2f}$ is more than 1000× faster than placement. In addition, it takes around 30 minutes to train the Net$^{2a}$ model.

## 6 DISCUSSION

In previous sections, we demonstrate the superior accuracy of Net$^2$ over other methods. Now we decompose Net$^{2a}$ to figure out which strategy contributes the most to its high accuracy. Table 7 and 8 measure the accuracy in identifying 10% longest nets and 10% longest paths. The 'Edge ANN' means using edge features without GNN. Edge features are aggregated on their target node and processed with ANN together with node features. The 'Simple Net' means using a simplified GNN structure. There is only one GAT layer for node convolution. The edge convolution only includes edge features and one layer of weights: $e_{k\_simple} = \sum_{b \in \mathcal{N}(k)} W_1 E_{b \to k}$. When not considering edge features, the receptive fields of 'Edge ANN' and 'Simple Net' are reduced to two hops and three hops, respectively. To analyze the contribution of different edge features, models 'F0 Net', 'F1 Net' and 'F2+f3 Net' use only $F0$, $F1$, $F2\,\&\,f3$ in edge features, respectively. The 'Less $P$ Net' uses only the 4 coarser-granularity but faster cell partitions over the 7 partitions $P$ used in Net$^{2a}$. Thus its number of clusters equals the number of cells divided by 500, 1000, 2000, and 3000. Its smallest granularity is 5× coarser than $P$. It is ~1.5× faster than Net$^{2a}$ and more than 20× faster than placement.

All these variations from Net$^{2a}$ outperform GAT. The 'Edge ANN' without GNN architecture is the worst variation, with around 4% performance degradation. ANN's receptive filed and flexibility are not as good as Net$^{2a}$. But it still demonstrates the effectiveness of edge features. The 'Simple Net' is 1.5-2% worse compared with Net$^{2a}$. Among the edge features, $F1$ contributes the most to accuracy. The model using less cell partitions $P$ is only around 0.6% less accurate than Net$^{2a}$. It means using coarser-granularity partitions will not largely affect the performance of Net$^{2a}$.

## 7 CONCLUSION

In this paper, we propose Net$^2$, a graph attention network method customized for individual net length estimation. It includes a fast version Net$^{2f}$ which is 1000 × faster than placement and an accuracy-centric version Net$^{2a}$ which efficiently extracts global information and outperform all previous methods. In future works, we will extend it to pre-placement timing analysis.

## ACKNOWLEDGMENTS

This work is partially supported by Semiconductor Research Corporation Tasks 2810.021 and 2810.022 through UT Dallas' Texas Analog Center of Excellence (TxACE).

## REFERENCES

[1] Srinivas Bodapati and Farid N Najm. 2001. Prelayout estimation of individual wire lengths. *TVLSI* (2001).
[2] Cadence. 2020. Cadence digital full flow optimized to deliver improved quality of results with up to 3X faster throughput. https://www.cadence.com/en_US/home/company/newsroom/press-releases/pr/2020/cadence-digital-full-flow-optimized-to-deliver-improved-quality-.html
[3] Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. 2000. RT-level ITC'99 benchmarks and first ATPG results. *Design & Test of computers* (2000).
[4] Yen-Chun Fang et al. 2018. Machine-learning-based dynamic IR drop prediction for ECO. In *ICCAD*.
[5] Bahareh Fathi, Laleh Behjat, and Logan M Rakai. 2009. A pre-placement net length estimation technique for mixed-size circuits. In *SLIP*.
[6] Matthias Fey and Jan E. Lenssen. 2019. Fast graph representation learning with PyTorch Geometric. In *ICLR-W*.
[7] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NeurIPS*.
[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*.
[9] Bo Hu and Malgorzata Marek-Sadowska. 2003. Wire length prediction based clustering and its application in placement. In *DAC*.
[10] Daijoon Hyun, Yuepeng Fan, and Youngsoo Shin. 2019. Accurate wirelength prediction for placement-aware synthesis through machine learning. In *DATE*.
[11] Andrew B Kahng and Sherief Reda. 2005. Intrinsic shortest path length: a new, accurate a priori wirelength estimator. In *ICCAD*.
[12] George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. 1999. Multilevel hypergraph partitioning: applications in VLSI domain. *TVLSI* (1999).
[13] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. In *ICLR*.
[14] Robert Kirby et al. 2019. CongestionNet: Routing Congestion Prediction Using Deep Graph Neural Networks. In *VLSI-SoC*.
[15] Qiang Liu, Jianguo Ma, and Qijun Zhang. 2012. Neural network based pre-placement wirelength estimation. In *FPT*.
[16] Qinghua Liu and Malgorzata Marek-Sadowska. 2004. Pre-layout wire length and congestion estimation. In *DAC*.
[17] Yuzhe Ma et al. 2019. High performance graph convolutional networks with applications in testability analysis. In *DAC*.
[18] Nir Magen, Avinoam Kolodny, Uri Weiser, and Nachum Shamir. 2004. Interconnect-power dissipation in a microprocessor. In *SLIP*.
[19] Adam Paszke et al. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*.
[20] Massoud Pedram and Narasimha Bhat. 1991. Layout driven technology mapping. In *DAC*.
[21] Massoud Pedram and Narasimha B Bhat. 1991. Layout driven logic restructuring/decomposition. In *ICCAD*.
[22] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.
[23] Synopsys. 2020. Fusion Compiler: the singular RTL-to-GDSII digital implementation solution. https://www.synopsys.com/implementation-and-signoff/physical-implementation/fusion-compiler.html
[24] Petar Veličković et al. 2017. Graph attention networks. In *ICLR*.
[25] Zhiyao Xie et al. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *ICCAD*.
[26] Zhiyao Xie, Haoxing Ren, Brucek Khailany, Ye Sheng, Santosh Santosh, Jiang Hu, and Yiran Chen. 2020. PowerNet: Transferable Dynamic IR Drop Estimation via Maximum Convolutional Neural Network. In *ASPDAC*.
[27] Guo Zhang, Hao He, and Dina Katabi. 2019. Circuit-GNN: Graph neural networks for distributed circuit design. In *ICML*.