2022 ACM/IEEE 4th Workshop on Machine Learning for CAD (MLCAD) | 978-1-6654-9729-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/MLCAD55463.2022.9900089

# Placement Optimization via PPA-Directed Graph Clustering

Yi-Chen Lu
yclu@gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Tian Yang
tiyang@nvidia.com
Nvidia
Santa Clara, CA, USA

Sung Kyu Lim
limsk@ece.gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Haoxing Ren
haoxingr@nvidia.com
Nvidia
Austin, TX, USA

## ABSTRACT

In this paper, we present the first Power, Performance, and Area (PPA)-directed, end-to-end placement optimization framework that provides cell clustering constraints as placement guidance to advance commercial placers. Specifically, we formulate PPA metrics as Machine Learning (ML) loss functions, and use graph clustering techniques to optimize them by improving clustering assignments. Experimental results on 5 GPU/CPU designs in a 5*nm* technology not only show that our framework immediately improves the PPA metrics at the placement stage, but also demonstrate that the improvements last firmly to the *post-route* stage, where we observe improvements of 89% in total negative slack (TNS), 26% in effective frequency, 2.4% in wirelength, and 1.4% in clock power.

## CCS CONCEPTS

• **Hardware** → **Placement**; **Physical design (EDA)**.

## KEYWORDS

placement optimization, unsupervised learning, graph clustering

**ACM Reference Format:**
Yi-Chen Lu, Tian Yang, Sung Kyu Lim, and Haoxing Ren. 2022. Placement Optimization via PPA-Directed Graph Clustering. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD '22), September 12–13, 2022, Snowbird, UT, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3551901.3556482

## 1 INTRODUCTION

Driven by the Moore's Law, modern VLSI designs easily consist of millions of instances that are required to be placed and routed. However, existing commercial placers leverage various heuristics or analytical methods that do not scale globally, which often leads to sub-optimal optimization results in advanced technologies. To improve placement quality, several Machine Learning (ML) techniques have been proposed to predict placement metrics using supervised learning [1–3]. However, these supervised models require a huge amount of data for training and are limited to the technologies that are trained upon, which often leads to undesirable generalizability.

In this work, to develop a robust ML framework that can generalize across different designs and technologies, we adopt unsupervised learning to improve commercial placers. Previous works [8, 9] have shown that unsupervised ML algorithms can provide cell clustering constraints as placement guidance to improve optimization quality, where popular approaches involve first building graph neural network (GNN) models to construct node embeddings, and then applying a spatial clustering algorithm (e.g., K-means) to determine cell clusters. However, this "two-step" approach often leads to sub-optimal Power, Performance, and Area (PPA) results because the GNN embedding process is not "goal-directed" (i.e., not guided by any design metric) as the representation learning and the clustering steps are not end-to-end differentiable.

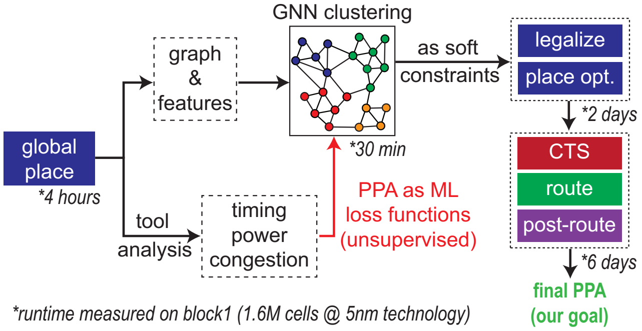


**Figure 1: Proposed PPA-directed unsupervised placement optimization framework in an industrial PD flow.**

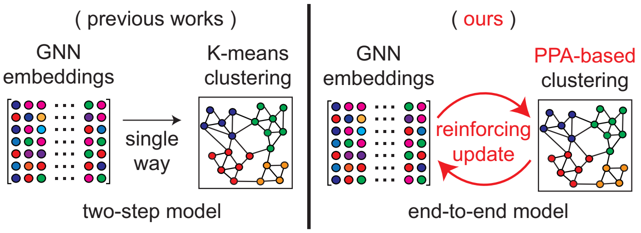


**Figure 2: Difference between prior works [8, 9] and ours.**

To overcome the above issues, in this paper, we develop an end-to-end, PPA-directed placement optimization framework as shown in Figure 1, where the key difference between our work and previous works [8, 9] is shown in Figure 2. Given an initial placement, our framework learns to discover the cell clusters that are critical for post-route PPA improvements by directly optimizing PPA metrics as ML loss functions, which are formulated unsupervisedly from the timing, power, and congestion analysis of the underlying placement. During placement optimization, the obtained clustering results are taken as soft constraints, where cells belonging to the same cluster will be grouped closer to each other with extra effort. Note that the entire learning process is unsupervised and end-to-end

*MLCAD '22, September 12–13, 2022, Snowbird, UT, USA*
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9486-4/22/09.
https://doi.org/10.1145/3551901.3556482

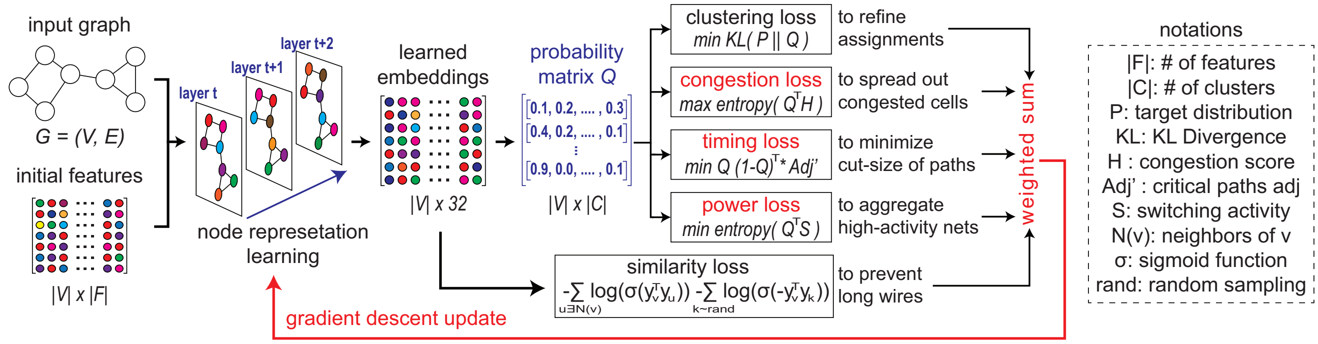


**Figure 3: Our PPA-directed unsupervised deep graph clustering framework. Given a netlist graph, initial node features, and PPA tool analysis (congestion, timing, power), our framework directly optimizes PPA metrics as ML loss functions by jointly improving the node embeddings and the clustering assignments in an end-to-end manner.**

differentiable, which means our framework can be applied to any design or technology as a standalone optimization algorithm.

The goal of this work is to present a generalizable placement optimization framework that can improve post-route PPA metrics with little runtime overhead. Undoubtedly, the ultimate goal of every PD implementation is to meet end-of-flow PPA target closures, and we believe the best way to achieve this is to start from a better placement. Our key contributions are summarized as follows:

- To the best of our knowledge, we are the ***first*** work that directly formulates PPA metrics as ML loss functions and optimizes them to improve the placement quality of commercial tools that are widely used across the entire semiconductor industry.
- To the best of our knowledge, we are the ***first*** work that develops an end-to-end unsupervised clustering framework in the realm of EDA where the representation learning and the clustering assignments are jointly updated in a goal-directed manner.
- We validate the proposed framework in an industrial PD flow using a commercial 5*nm* technology and benchmarks with *millions of cells*. We not only show that our framework immediately improves the PPA metrics at the placement stage, but also demonstrate that the improvements last firmly to the post-route stage.

## 2 OVERVIEW AND MOTIVATION

Figure 3 shows an high-level overview of the proposed placement optimization framework, which leverages unsupervised deep embedded clustering [15] equipped with GNN representation learning to identify the cell clustering assignments that can be leveraged to improve the underlying placement. The inputs to our framework are a netlist graph $G = (V, E)$, initial node features $Y^0 \in R^{|V| x |F|}$, and tool-based PPA analysis of the underlying placement, which includes congestion scores $H \in R^{|V|}$, maximum switching activities $S \in R^{|V|}$, and the adjacency matrix of critical timing paths $Adj'$. The key output of our framework is the probability matrix $Q \in R^{|V| x |C|}$ where each element $Q_{ij}$ represents the probability of a cell $i$ belonging to a cluster $j$. During the learning process, our framework will jointly refine the node embeddings and the clustering assignments by minimizing the proposed PPA-inspired ML loss functions using a gradient descent optimizer.

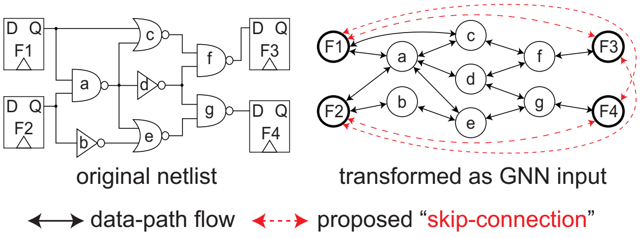


**Figure 4: Proposed netlist transformation.**

The main motivation of our framework is that if we know a path is timing critical, or if we know a net has a high switching activity, then we would like to shorten the path or the net by moving cells closer to each other in order to reduce the resistances and capacitances involved. In addition, if we know an area is highly congested, then we would want to spread out the cells within to reduce the congestion as it directly impacts the subsequent routing stage and hence the end-of-flow PPA metrics. To summarize, during placement optimization, we want to improve the cell locations based on the underlying PPA evaluations. In this work, we aim to unleash the power of ML to achieve this goal in a systematic and global manner by directly formulating PPA metrics as ML loss functions and optimizing them in consideration of every cell in the design.

## 3 ALGORITHMS

### 3.1 Timing-Aware Netlist Transformation

Recently, [14] shows that encoding functionality accurately is critical to the success of GNN representation learning on netlists. Nonetheless, previous GNN-based placement optimization works [1, 8, 9] all adopted the clique-based model for netlist transformation. This approach does not consider netlist functionality and suffers from the fact that the number of edges in the transformed graph grows quadratically to the number of nodes in the original netlist, which causes memory issues on large designs and weakens the expressiveness of node representation learning as it GNNs will struggle to identify important connections with excessive edges.

To overcome these issues, we propose a new transformation method as shown in Figure 4. The proposed method brings two timing-related improvements upon the clique-based technique.

**Table 1: Our initial node features for deep graph clustering. $M$ denotes the number of memory macros.**

| type | # dim. | description |
|---|---|---|
| name embeddings | 16 | hierarchical name encoded by S-BERT [11] |
| memory affinity | $M$ | shortest logic distance to each memory |
| wst output slack | 1 | worst slack value at output pin |
| wst output slew | 1 | maximum transition at output pin |
| wst input slew | 1 | maximum transition among input pin(s) |
| largest activity | 1 | largest switching activity value among nets |
| locations | 2 | (x,y) location of initial placement |

First, for every net in the original netlist, we only introduce the driver-to-load connection(s) in the transformed graph instead of forcing every cell on the same net to share connections with each other. Second, given that the receptive field of a GNN model is limited by its number of layers, we introduce "skip-connections" (denoted in red) to link start points and end points of timing paths, which enables GNNs to more easily capture timing-related attributes. In our transformation, the number of edges in the transformed graph grows pseudo-linearly to the number of nodes in the original netlist, which is fully applicable to large-scale designs.

## 3.2 Node Representation Learning

Prior to the graph learning process, we collect initial node-specific features for each design instance (i.e., cell) as shown in Table 1, which include an instance's physical information and its timing and power related attributes. However, these initial node features are not sufficient to identify the essential cell clusters for placement optimization because they do not reflect connectivity among cells. To obtain better representations for each instance, we leverage GNNs to perform node representation learning on the transformed netlist graph as aforementioned. Considering the runtime and memory benefits of graph inductive learning, in this work, we leverage GraphSAGE [4] to perform the node representation learning as:

$$\begin{aligned} y^{k-1}_{N(v)} &= mean\_pool\left(\{\mathbf{W}^{agg}_k y^{k-1}_u, \forall u \in N(v)\}\right), \\ y^k_v &= sigmoid\left(\mathbf{W}^{proj}_k \cdot \text{concat}\left[y^{k-1}_v, y^{k-1}_{N(v)}\right]\right), \end{aligned} \tag{1}$$

where $N(v)$ denotes the neighbors of node $v$, $W^{agg}_k$ and $W^{proj}_k$ denote the aggregation and projection matrices at the $k$-th layer of the GNN module. After the transformation through Equation 1, the initial node features of each cell $y^0_v$ will be transformed into $y^K_v$, where $K$ denotes the total number of layers. The dimensions of $y^K_v$ is subject to the number of neurons at the last layer. In the implementation, we set $K = 6$ and $dim(y^K_v) = 32$.

### 3.2.1 *Similarity Loss* $L_{sim}$.

We now introduce the first objective of our framework which is termed as "similarity loss" and is directly calculated from the above GNN learned embeddings $\{y^K\}$. The key idea behind is to encourage cells on the same net to have higher probabilities of being assigned into the same cluster, while making nodes that are logically distant to have lower probabilities, which therefore minimizes the chances of creating long nets. The loss function $L_{sim}$ is designed as:

$$L_{sim} = \sum_v \left( - \sum_{u \in N(v)} \log\left(\sigma(y_v^\top y_u)\right) - \sum_{k \sim rand} \log\left(\sigma(-y_v^\top y_k)\right) \right), \tag{2}$$

where $y_v$ denotes the learned embeddings of node $v$, $\sigma$ denotes the sigmoid function, and $rand$ denotes the random sampling operation over the full netlist graph. By minimizing Equation 2, neighboring nodes will be encouraged to have similar embeddings $y$, which increases the probability of them being assigned to the same cluster and hence prevents creating long pin-to-pin connections.

## 3.3 PPA-Directed Deep Graph Clustering

Unlike previous works [8, 9] that rely on the weighted K-means clustering to heuristically determine cell clusters, in this work, we propose a PPA-directed clustering technique that identifies the essential cell clusters by optimizing PPA as ML loss functions.

### 3.3.1 *Unsupervised Clustering Loss* $L_{cl}$.

One of the main challenges of the clustering task is the non-existence of label guidance. To overcome this challenge, we devise a self-reinforcing method that iteratively converts "distances" of trained GNN node embeddings $\{y\}$ into "probabilities" of clustering assignments. Particularly, we leverage the Student's t-distribution [13] as a kernel to perform the distance-to-probability conversion as:

$$Q_{ic} = \frac{\left(1 + \|y_i - \mu_c\|^2\right)^{-1}}{\sum_k \left(1 + \|y_i - \mu_k\|^2\right)^{-1}}, \tag{3}$$

where $Q_{ic}$ denotes the probability of node i belonging to cluster $c$, $y_i$ denotes the learned GNN node embeddings of node $i$, and $\mu_c$ denotes the embeddings of centroid $c$ which is a trainable vector that is improved in every iteration, and $\|\cdot\|^2$ denotes the Euclidean distance. To optimize the clustering assignments (i.e., matrix $Q$) in a self-reinforcing manner, we further construct a target matrix $P$ by strengthening the assignments of $Q$ as:

$$P_{ic} = \frac{Q^2_{ic} / \sum_i Q_{ic}}{\sum_k Q^2_{ik} / \sum_i Q_{ik}}. \tag{4}$$

The rationale behind Equation 4 is that since $Q$ is a stochastic matrix which means $0 \leq Q_{ic} \leq 1$, raising and then normalizing by the second power will make the probability distribution of each row (i.e., assignment distribution of a cell) skew towards to the largest value. Hence, the assignments are strengthened. Now, with the target matrix $P$ and the approximate matrix $Q$, we can define the clustering loss $L_{cl}$ as:

$$L_{cl} = KL(P||Q), \tag{5}$$

where $KL$ denotes the Kullback-Leibler divergence [6]. Minimizing Equation 5 will encourage the matrix $Q$ to approximate the matrix $P$. To this end, we have bridged the gap between node representation learning and cell clustering by converting the GNN node embeddings into cell clustering probabilities. With the probability matrix $Q$ that represents clustering assignments, we can further quantify its "expected" impact on important PPA metrics. In this work, we specifically focus on formulating the congestion, power, and timing objectives as ML loss functions to improve placement.

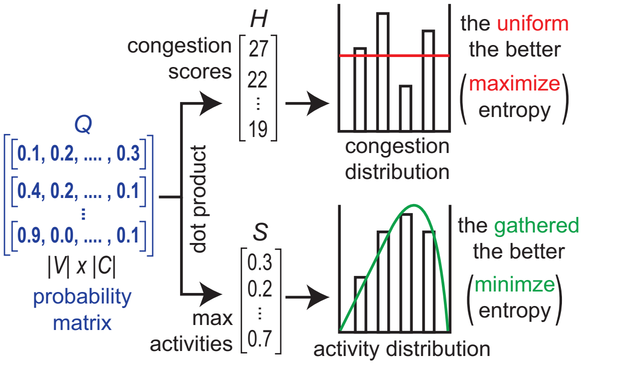


**Figure 5: Illustration of congestion loss and power loss formulations that rely on entropy maximization and minimization. Note that both distributions are normalized as probabilities.**

*3.3.2* ***Congestion Loss*** $L_{cong}$*.* To reduce congestion, we adopt the concept of cell spreading, where the key idea is to spread out cells in congested regions by adjusting their clustering assignments. To formulate this objective into an ML loss function, we leverage Shannon entropy [12] to quantify the "randomness" of probability distributions and define the congestion loss $L_{cong}$ as:

$$\max\ entropy\left(Q^{\top}H\right) \rightarrow L_{cong} = -entropy\left(Q^{\top}H\right), \qquad (6)$$

where $H \in R^{|V|}$ is a vector that denotes the congestion score of each cell, $Q^{\top}H \in R^{|C|}$ thus represents the expected congestion score of each cluster, and finally $entropy(\cdot)$ denotes the function mapping that first normalizes each element by the sum of all elements, and then calculates the Shannon entropy of the normalized probability vector. With Equation 6, the probability matrix $Q$ will be encouraged to spread out cells in the congested regions as the maximum entropy is achieved by having an equal amount of congestion (i.e., $\frac{\sum_v H_v}{|C|}$) in each cluster. Figure 5 shows the illustration of congestion loss.

*3.3.3* ***Power Loss*** $L_{power}$*.* Our key idea to improve power is to shorten the nets with high-switching activities by aggregating related cells closer to each other, which effectively reduces the switching capacitance involved. To achieve this, we again leverage the concept of entropy and formulate the power objective $L_{power}$ as:

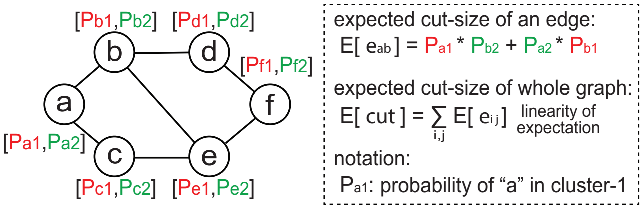
$$\min\ entropy\left(Q^{\top}S\right) \rightarrow L_{power} = entropy\left(Q^{\top}S\right), \qquad (7)$$

where $S \in R^{|V|}$ denotes the largest switching activity of the net that a cell is connecting to. The idea behind power loss is similar to that of the congestion loss as shown in the bottom-part of Figure 5. The difference is that we are now minimizing rather than maximizing the entropy to aggregate the cells instead of spreading them.

*3.3.4* ***Timing Loss*** $L_{timing}$*.* Our key idea to improve timing is to let cells on timing critical paths have higher chances of being clustered into the same group. To achieve this, as in [10], we formulate the "cut-size" of timing critical paths as an ML loss function, which is resulted from the current clustering assignments $Q$. Figure 6 shows an illustration of our cut-size loss formulation. Note that although a two-way partitioning example is shown in the figure, this formulation can be easily extended to handle multi-way partitioning through matrix factorization. For $|C|$-way partitioning, we formulate the cut-size as timing loss $L_{timing}$ based on the probability matrix $Q \in R^{|V|\text{x}|C|}$ as:

$$L_{timing} = reduce\_sum\left(Q\,(1-Q)^{\top} \odot A_{critic}\right), \qquad (8)$$

where $A_{critic}$ denotes the adjacency matrix of timing critical paths, $reduce\_sum(\cdot)$ denotes the operation that adds up all the input elements, and $\odot$ denotes the element-wise multiplication.


[Pb1,Pb2]
[Pd1,Pd2]
[Pf1,Pf2]
[Pa1,Pa2]
[Pc1,Pc2]
[Pe1,Pe2]
a
b
c
d
e
f
expected cut-size of an edge:
E[ eab ] = Pa1 * Pb2 + Pa2 * Pb1
expected cut-size of whole graph:
E[ cut ] = ∑ i,j E[ eij ] linearity of expectation
notation:
Pa1: probability of "a" in cluster-1

*example is a two-way partition, but can easily be extended to multi-way

**Figure 6: Illustration of "cut-size" loss formulation.**

**Algorithm 1** End-to-End Unsupervised Training Methodology. We use default values of $sim\_epoch = 10, full\_epoch = 50, \alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \lambda_1 = 1, \lambda_2 = 10, \lambda_3 = 1, \lambda_4 = 0.5$.

```
Input: $G = (V, E)$: transformed graph. $\{y^0\}$: initial node features. $H \in R^{|V|}$: congestion scores. $A_{critic}$: critical path adjacency matrix. $S \in R^{|V|}$: maximum switching activities. $\{W\}$: weights of GNN. $sim\_epoch$: number of epochs for similarity-only learning. $full\_epoch$: number of epochs for full-objective learning. $\{\beta_1, \beta_2\}$: Adam params. $\alpha$: learning rate. $\{\lambda\}$: objective weights.
Output: $Z \in R^{|V|}$: final clustering assignment of each cell
for $i = 0$; $i < sim\_epoch$; $++i$ do                 ▷ Pre-train GNN weights
    $y \leftarrow \text{GNN}(G, y^0; W)$                 ▷ GNN embeddings by Equation 1
    $L_{sim} \leftarrow sim\_loss\,(y)$                  ▷ similarity loss by Equation 2
    $W \leftarrow Adam\,(L_{sim}, \beta_1, \beta_2, \alpha; W)$
$\{\mu\} \leftarrow$ obtain initial centroids from $y$ using K-means
add $\{\mu\}$ to ML computational graph                  ▷ make $\{\mu\}$ trainable
for $i = 0$; $i < full\_epoch$; $++i$ do
    $y \leftarrow \text{GNN}(G, y^0; W)$
    $L_{sim} \leftarrow sim\_loss\,(y)$
    $Q \leftarrow$ probability matrix from $\{y, \mu\}$  ▷ by Equation 3
    if $i \,\%\, 3 == 0$ then
        $P \leftarrow$ target matrix from $Q$            ▷ by Equation 4
    $L_{cl} \leftarrow$ clustering loss from $\{P, Q\}$  ▷ by Equation 5
    $L_{cong} \leftarrow$ congestion loss from $\{Q, H\}$  ▷ by Equation 6
    $L_{power} \leftarrow$ power loss from $\{Q, S\}$    ▷ by Equation 7
    $L_{timing} \leftarrow$ timing loss from $\{Q, A_{critic}\}$  ▷ by Equation 8
    $L = L_{sim} + \lambda_1 L_{cl} + \lambda_2 L_{cong} + \lambda_3 L_{timing} + \lambda_4 L_{power}$
    $W, \mu \leftarrow Adam\,(L, \beta_1, \beta_2, \alpha; W, \mu)$ ▷ update GNN, centroids
$Z \leftarrow$ get argmax of $Q$ by row                  ▷ final clustering assignments
```

## 3.4 End-to-End Unsupervised Training

Now, after describing all the objectives of our framework, we jointly optimize them using a gradient descent optimizer Adam [5] to minimize the weighted sum of each objective as:

$$L = L_{sim} + \lambda_1 L_{cl} + \lambda_2 L_{cong} + \lambda_3 L_{timing} + \lambda_4 L_{power}, \qquad (9)$$

where $\lambda_i \geq 0$ controls the contribution of each objective to the clustering assignment. After the training is complete, we obtain the

**Table 2: Detailed PPA comparison between the default tool flow and the enhanced flow. We normalize wirelength and power values due to proprietary. All designs have ultra-high frequency targets with total number of cells ranging from $1.3M - 1.6M$. The effective frequencies across all benchmarks are improved by up to 26%. The place-to-route runtime takes around 10 days.**

| design<br>#clusters | PD<br>stage | default industrial PD flow (no clustering) | | | | | | | default + unsupervised clustering (ours) | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | wns<br>(ps) | TNS<br>(ns) | #<br>vios | total<br>WL | clock<br>WL | total<br>power | clock<br>power | wns<br>(ps) | TNS<br>(ns) | #<br>vios | total<br>WL | clock<br>WL | total<br>power | clock<br>power |
| design1 | post-place | -48 | -41.45 | 3306 | 1 | - | 1 | - | -39 | -14.18 (-66%) | 1622 | 0.999 | - | 1.000 | - |
| $\|C\| = 10$ | post-route | -88 | -2.89 | 574 | 1 | 1 | 1 | 1 | -9 | -0.32 (-88%) | 149 | 1.000 | 0.994 | 1.001 | 0.994 |
| design2 | post-place | -13 | -0.084 | 23 | 1 | - | 1 | - | -4 | -0.007(-92%) | 2 | 1.000 | - | 1.001 | - |
| $\|C\| = 11$ | post-route | -29 | -3.82 | 1440 | 1 | 1 | 1 | 1 | -22 | -3.07 (-20%) | 1288 | 0.998 | 0.999 | 0.999 | 0.989 |
| design3 | post-place | -4 | -0.019 | 10 | 1 | - | 1 | - | -1 | -0.001(-95%) | 1 | 0.998 | - | 0.997 | - |
| $\|C\| = 11$ | post-route | -20 | -2.24 | 996 | 1 | 1 | 1 | 1 | -15 | -1.62 (-28%) | 823 | 0.996 | 0.978 | 0.996 | 0.998 |
| design4 | post-place | -5 | -0.042 | 16 | 1 | - | 1 | - | -2 | -0.007(-83%) | 4 | 0.977 | - | 0.992 | - |
| $\|C\| = 10$ | post-route | -22 | -2.65 | 1221 | 1 | 1 | 1 | 1 | -12 | -2.16 (-19%) | 1103 | 0.976 | 0.985 | 0.991 | 0.986 |

**Table 3: Optimization results analysis using proposed PPA losses (Eq 6,7,8). Colored entries denote better evaluations.**

| design | congestion entropy | | path cut-size | | power entropy | |
|---|---|---|---|---|---|---|
| | default | ours | default | ours | default | ours |
| design1 | 1.901 | 1.902 | 908.6 | 905.5 | 1.940 | 1.937 |
| design2 | 2.356 | 2.360 | 629.4 | 603.7 | 2.142 | 2.139 |
| design3 | 2.289 | 2.291 | 864.1 | 835.3 | 2.106 | 2.104 |
| design4 | 2.014 | 2.015 | 789.4 | 784.5 | 1.995 | 1.988 |

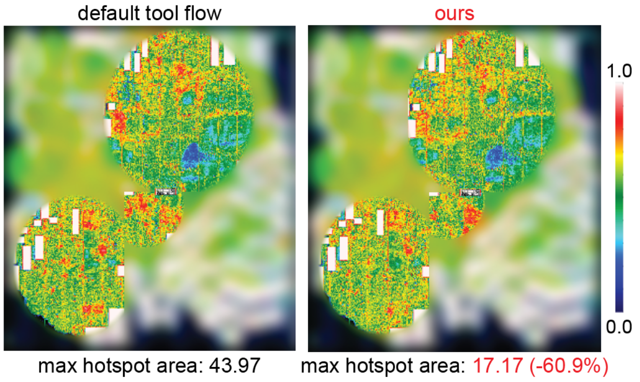
final clustering assignment of each cell $v$ as:

$$\text{assignment of node } v = \operatorname*{argmax}_{c} Q_{vc}. \tag{10}$$

Algorithm 1 summarizes the training process. In Lines 1–4, we first pre-train the GNN module using the similarity loss (Equation 2). Then, in Lines 5–6, based on the pre-trained embeddings, we obtain the initial clustering centers $\{\mu\}$ (i.e., centroids) in high dimensions using the K-means algorithm [7] and make these centroids trainable by adding them to the ML computational graph. Note that the K-means algorithm is only conducted once and for all to obtain the initial clusters. In Lines 7–19, we compute each objective function as described in the above equations and jointly optimize them using gradient descent. It is worth to mention that in Lines 11–12, we update the target matrix $P$ once in every three iterations to stabilize the convergence. Finally, the computed gradients are taken to update the parameters in the ML computational graph including the GNN weight matrices $\{W\}$ and the center locations $\{\mu\}$.

## 4 EXPERIMENTAL RESULTS

We validate the proposed framework on 5 industrial GPU/CPU designs in a commercial $5nm$ technology node with the total number of cells ranging from $1.3M$ to $1.6M$, number of flops ranging from $95k$ to $150k$, and number of macros ranging from 20 to 150. Due to proprietary we cannot disclose the exact attributes of each design. The proposed deep graph clustering framework is implemented with the *PyTorch* library. The entire training process is conducted on a single *NVIDIA TESLA V*100 GPU with $32GB$ memory. For each benchmark, *the entire training only takes less than 30 minutes.*



*max hotspot area is the largest contiguous area of gcells with overflow

**Figure 7: Impact on congestion after placement. Our optimization technique reduces the worst congestion by 60.9%.**

### 4.1 Optimization Results on Industrial Flows

Table 2 shows the detailed PPA impact of our framework being integrated with an industrial design flow. We observe that across all benchmarks, our framework demonstrates immediate PPA improvements at the placement stage, and these improvements last thoroughly to the post-route stage. The timing improvements are especially significant. With all the benchmarks operating in ultra-high frequency ($> 3GHz$), the effective frequency can be improved by 26%. Another trend worth to mention is that the proposed framework consistently improves the clock wirelength and the clock power across all designs by up to 2.2% and 1.4%, respectively. We believe these clock-related improvements are resulted from the proposed "skip-connection" technique that introduces GNN message passing edges between launch flops and capture flops. Finally, to determine the total number of clusters $|C|$ that each benchmark is clustered into, we conduct 8 parallel model training with the target number of clusters varying between 7 and 14 (inclusive) to find the one that achieves the minimum loss (Equation 9). Note that even on designs with millions of cells, *the entire parallel training process takes less than* 30 *minutes*, which introduces almost no additional runtime overhead compared with the full-flow implementation runtime that takes around 10 days. Figure 7 visualizes the congestion impact. It

**Table 4: Detailed comparison with the previous work [8].**

| PD | post-place | | post-cts | | post-route | |
|---|---|---|---|---|---|---|
| stage | [8] | ours | [8] | ours | [8] | ours |
| design: design1 ( $> 1.3M$ cells ), $\lvert C\rvert_{[23]} = 13$, $\lvert C\rvert_{ours} = 10$ | | | | | | |
| WNS (ns) | -0.044 | -0.039 | -0.048 | -0.039 | -0.056 | -0.009 (-83.9%) |
| TNS (ns) | -35.35 | -14.18 | -67.71 | -61.45 | -2.965 | -0.032 (-98.9%) |
| # vios | 3119 | 1622 | 6411 | 6049 | 867 | 149 (-82.8%) |
| WL | 1 | 0.999 | 1 | 0.998 | 1 | 0.999 |
| design: design5 ( $> 1.3M$ cells ), $\lvert C\rvert_{[23]} = 10$, $\lvert C\rvert_{ours} = 8$ | | | | | | |
| WNS (ns) | -0.058 | -0.073 | -0.193 | -0.121 | -0.115 | -0.107 (-6.9%) |
| TNS (ns) | -6.42 | -6.06 | -23.35 | -19.29 | -27.37 | -15.85 (-42.1%) |
| # vios | 479 | 374 | 1608 | 1333 | 1118 | 413 (-63.1%) |
| WL | 1 | 0.993 | 1 | 0.981 | 1 | 0.992 |

is shown that the proposed framework immediately reduces the worst hotspot area by 60.9%, which justifies the effectiveness of the proposed congestion loss function (Equation 6).

#### 4.1.1 *Analysis on Optimization Results using PPA Losses.*

Throughout the past several decades, half-perimeter wirelength (HPWL) and overflow have been the two dominant metrics for placement evaluation. However, it is widely acknowledged that these two popular metrics no longer show good correlation with post-route PPA particularly in advanced technology nodes. In this experiment, instead of using the two conventional metrics, we demonstrate the effectiveness of using our PPA loss functions to evaluate placement quality as shown in Table 3. This implies that we can improve placement *in the same direction* as improving post-route QoR metrics by optimizing the proposed PPA loss functions.

### 4.2 Comparing with Single-Way Clustering [8]

To demonstrate the effectiveness of the proposed PPA-directed, end-to-end clustering approach, in this experiment, we perform head-to-head comparisons with the previous work [8] that requires a two-step process to determine the clustering assignments. Table 4 demonstrates the comparison results. We observe that our framework achieves significantly better optimization results in every major PPA metric across all PD stages. We believe the improvements come from the fact that compared with [8], our framework not only can perform better representation learning with the PPA feedback from the clustering assignments, but also can improve the clustering results with improved node representations.

### 4.3 Why Does our Framework Work?

In the above experiments, we have demonstrate the superior results achieved by our framework. It significantly improves an industrial design flow and the previous work [8]. We believe the achievements of our framework can be accounted by the following reasons:

#### 4.3.1 *Global and Systematic Optimization.*

We think the first reason comes from the fact that the proposed ML-based placement optimization technique improves design PPA metrics more globally and systematically compared with the existing heuristic algorithms in commercial tools that often perform the optimization in a local and ad-hoc manner (i.e., path-by-path, cone-by-cone, or subgraph-by-subgraph) because of their inabilities to deal with large design complexity. Unlike these heuristic algorithms, our framework optimizes the entire netlist graph as a complete entity, where the PPA-related objectives are calculated across every cell in the design. Therefore, it has the ability to capture the complicated interactions among instances that are distant to each other.

#### 4.3.2 *PPA-Directed End-to-End Optimization.*

Unlike previous graph unsupervised learning works [8, 9] that leverages GNNs to obtain node embeddings without any guidance from the subsequent clustering task, in this work, we design PPA-inspired ML loss functions related to congestion, timing, and power to optimize both node representation learning and clustering assignments in an end-to-end manner. In contrary to previous works that improve PPA metrics indirectly, our framework works as a stand-alone optimizer that directly improves the PPA metrics using ML algorithms. The direct benefits of our approach is demonstrated in Table 4.

## 5 CONCLUSION

In this paper, we have presented the first placement optimization framework that directly formulates PPA metrics as ML loss functions and optimizes them to discover the cell clusters that can improve the underlying placement and end-of-flow QoR metrics. We validate the proposed framework with industry-leading commercial tools in an industrial design flow and demonstrate that the improvements are significant and consistent across different commercial GPU/CPU designs. We believe this work shall demonstrate that ML algorithms can not only be utilized to solve the prediction or simulation tasks in EDA, but can also be leveraged as standalone algorithms that directly optimize design-related metrics.

## REFERENCES

[1] A. Agnesina, K. Chang, and S. K. Lim. Vlsi placement parameter optimization using deep reinforcement learning. In Proceedings of the 39th International Conference on Computer-Aided Design, pages 1–9, 2020.

[2] X. Gao, Y.-M. Jiang, L. Shao, P. Raspopovic, M. E. Verbeek, M. Sharma, V. Rashingkar, and A. Jalota. Congestion and timing aware macro placement using machine learning predictions from different data sources: Cross-design model applicability and the discerning ensemble. In Proceedings of the 2022 International Symposium on Physical Design, pages 195–202, 2022.

[3] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In Proceedings of the 59th Annual Design Automation Conference 2022. ACM, 2022.

[4] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. Advances in neural information processing systems, 30, 2017.

[5] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[6] S. Kullback and R. A. Leibler. On information and sufficiency. The annals of mathematical statistics, 22(1):79–86, 1951.

[7] S. Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2):129–137, 1982.

[8] Y.-C. Lu, S. Pentapati, and S. K. Lim. The law of attraction: Affinity-aware placement optimization using graph neural networks. In Proceedings of the 2021 International Symposium on Physical Design, pages 7–14, 2021.

[9] Y.-C. Lu, S. S. K. Pentapati, L. Zhu, K. Samadi, and S. K. Lim. Tp-gnn: A graph neural network framework for tier partitioning in monolithic 3d ics. In 2020 57th ACM/IEEE Design Automation Conference (DAC), pages 1–6. IEEE, 2020.

[10] A. Nazi, W. Hang, A. Goldie, S. Ravi, and A. Mirhoseini. Gap: Generalizable approximate graph partitioning framework. arXiv:1903.00614, 2019.

[11] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084, 2019.

[12] C. E. Shannon. A mathematical theory of communication. The Bell system technical journal, 27(3):379–423, 1948.

[13] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. Journal of machine learning research, 9(11), 2008.

[14] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang. Functionality matters in netlist representation learning. In Proceedings of the 59th Annual Design Automation Conference 2022. ACM, 2022.

[15] J. Xie, R. Girshick, and A. Farhadi. Unsupervised deep embedding for clustering analysis. In International conference on machine learning. PMLR, 2016.