Generalizable Cross-Graph Embedding for GNN-based Congestion Prediction

Amur Ghose * †, Vincent Zhang * †, Yingxue Zhang * †, Dong Li †, Wulong Liu †, Mark Coates ‡

***Abstract*—Presently with technology node scaling, an accurate prediction model at early design stages can significantly reduce the design cycle. Especially during logic synthesis, predicting cell congestion due to improper logic combination can reduce the burden of subsequent physical implementations. There have been attempts using Graph Neural Network (GNN) techniques to tackle congestion prediction during the logic synthesis stage. However, they require informative cell features to achieve reasonable performance since the core idea of GNNs is built on the message passing framework, which would be impractical at the early logic synthesis stage. To address this limitation, we propose a framework that can directly learn embeddings for the given netlist to enhance the quality of our node features. Popular random-walk based embedding methods such as Node2vec, LINE, and DeepWalk suffer from the issue of cross-graph alignment and poor generalization to unseen netlist graphs, yielding inferior performance and costing significant runtime. In our framework, we introduce a superior alternative to obtain node embeddings that can generalize across netlist graphs using matrix factorization methods. We propose an efficient mini-batch training method at the sub-graph level that can guarantee parallel training and satisfy the memory restriction for large-scale netlists. We present results utilizing open-source EDA tools such as DREAMPLACE and OPENROAD frameworks on a variety of openly available circuits. By combining the learned embedding on top of the netlist with the GNNs, our method improves prediction performance, generalizes to new circuit lines, and is efficient in training, potentially saving over 90% of runtime.**

I. Introduction

Routability-aware or congestion-aware placement has been well-studied in the existing literature [1]–[3]. By leveraging the rough evaluation of congestion via RUDY [4] and NCTU-GR [5], [6], the candidate placement can be optimized iteratively. However, with the technology node scaling and the increase of chip size, each placement iteration could be very time-consuming in order to achieve the optimal placement objective with good routability or less congestion. In addition, it would remain difficult to alleviate the routing congestion even with unlimited iterations to implement the best placement results, which is actually induced by the improper logic implementation at the early design stage, e.g., using too many high fan-in and fan-out logic cells in the logic synthesis phase.

This naturally raises the problem of estimating logic-induced congestion from the netlist directly, prior to the placement iterations. This approach is explored in a recent work, CongestionNet [7]. Using a deep Graph Attention Network (GAT) [8], CongestionNet achieves relatively promising congestion prediction results in the logic synthesis phase. As the core idea of GNNs is built on top of the message passing framework, the work assumes that informative cell features can be used to achieve reasonable performance. However, this would be impractical at the early logic synthesis stage.

In addition, the aforementioned GAT is a subclass of the broader group of Graph Neural Networks (GNN) [9], [10]. Such GNN methods are effective for general prediction tasks on graph structured data. In the case of circuits, the key deviation from the usual use case of GNNs is that the train and test graphs can be completely disjoint. Given some congestion data for a known set of circuits, the desired prediction is often not on the unseen cells of the same circuits, but on completely new circuits. This more challenging case is called **inductive learning**, and it requires specific GNN design and training approaches.

Separate from learning-based methods, there exist methods like GTL [11] which estimate congestion directly from the netlist based on the graph structure. Analogously, in the key areas of GNN usage such as recommendation systems and node level regression, there exist methods for learning high quality **embedding vectors** to complement node attributes using structural features based on the graph (in this case, netlist). These embedding vectors are informative for predicting properties, i.e., the labels. Embedding learning has achieved great success in the field of Natural Language Processing (NLP), where methods such as Word2Vec [12] and GloVe [13], allow us to learn meaningful vector representations of words and use them for various downstream tasks. Crucially, the CongestionNet uses informative cell attributes (cell size and pin count) alone as the input to the GAT and does not use any embedding encoding the netlist structure.

In this work, we show that it is possible to create high quality structural embeddings, based on matrix factorization techniques, to enhance the node feature quality. We show that it significantly improves GNN-based congestion prediction performance. Our key findings are :

- The three most popular and mainstream embedding methods for node-level embedding learning – Node2vec [14], LINE [15], and DeepWalk [16] – require post-processing (alignment) to be usable for cross-graph prediction.
- Matrix-factorization based embedding learning [17], [18] (a less popular method) combined with subgraph level training is faster, more effective, and can generalize to unseen netlists.
- Concatenating cell structural embeddings with cell attributes directly improves performance. When informa-

† Noah's Ark Lab, Huawei ‡ McGill University
* {amur.ghose, vincent.zhang2, yingxue.zhang }@huawei.com (Corresponding authors)

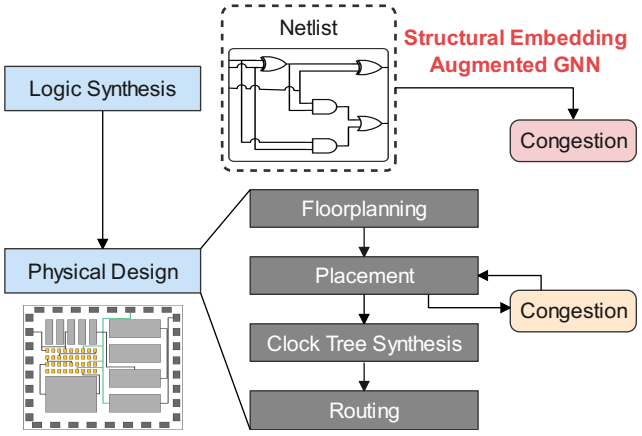

Fig. 1. Congestion prediction at different EDA stages. Routing congestion is commonly estimated in placement stage and used as a feedback to improve routability of placement solution. The GNN-based method proposed in this paper predicts congestion using netlist in logic synthesis phase.

tive cell attributes are not present, a meaningful prediction can be made from netlist embeddings alone.

- Instead of deep, wide GATs, wide and shallow SAGE [10] GNNs achieve superior performance.

We present our results on the publicly available DAC 2012[1] benchmarks using the superblue circuit line, with additional results on the OPENROAD framework.

II. Background - EDA and similarity on graphs

A. Congestion prediction in EDA

In the EDA workflow, Register Transfer Level (RTL) design in VDHL or Verilog is converted to a physical layout for manufacturing through logic synthesis and physical design (Figure 1). In the physical design stage, the circuit elements are placed on the circuit board and this is followed by the routing step. Although the global routing result provides a good estimation of routing congestion [6], [19], an awareness of high congestion areas at an early design stage is of great importance to provide fast feedback and shorten design cycles.

Multiple works have attempted to predict detailed routing congestion in the placement step in an effort to optimize routability of the placement solution. In [20], a probabilistic routing model is used to estimate routing demand during placement. In [4], the authors proposed a fast congestion prediction tool named RUDY, which estimates the wire density within the enclosing rectangle of a net using HPWL. A variety of techniques can be used to improve routability when knowledge of the congestion is available. In POLAR 2.0 [1], cells that are estimated to have high congestion are spread out and inflated to distribute routing demand more evenly. Alternatively, as proposed in [2], an appropriate amount of white space can be allocated to different areas according to the congestion estimation. All these techniques are implemented in the placement step and need the position information of cells.

To avoid the high computation cost of placement, it is more useful to be able to predict congestion in the logic synthesis phase. A few works have tried to identify network structures in a synthesized netlist that can indicate high routing congestion in local areas. As shown in [21], the size of the local neighborhood is a simple but effective metric that can be used to approximate congestion. Adhesion of a logic network, defined as the sum of min-cuts for all node pairs in a graph, is a good proxy of peak congestion as shown in [21]. In [11], the authors introduced the Groups of Tangled Logic (GTL) metric based on Rent's rule. This metric is able to detect clusters of cells that have the potential for high congestion.

A Graph Neural Network (GNN)-based method, named CongestionNet, has been proposed recently in [7] to predict congestion based on a synthesized netlist. As mentioned previously, Kirby et al. trained a GAT-based model using cell features as input to predict the final routing congestion, without any embedding to encode the structural properties of the netlist. The key difference between our approach and their model lies in our construction of an embedding pipeline for EDA netlists. We begin by discussing similarity notions in the context of embedding learning on graphs.

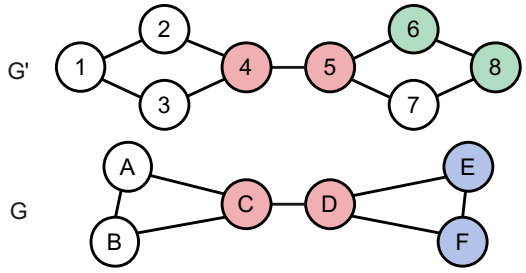

Fig. 2. Structural (red) vs proximity-based (blue and green) similarity across two disjoint graphs. In particular, $4, 5$ and C,D are similar even though they belong to two different graphs.

B. Proximity vs structural node similarity

The most straightforward notion of similarity between two vertices i, j in a graph is their **proximity** in the graph. Neighboring vertices are the most similar, and the similarity decreases as more edges must be traversed to travel from one vertex to the other. Separately from this, we may consider the notion of **structural** similarity, which relates to properties of a node such as its degree, spectral properties, etc. Two nodes can be structurally similar even if they belong to two different graphs. The contrast between these two ideas of similarity is depicted in Figure 2, where we outline structural similarity. We now move to a detailed discussion of these similarity metrics and their suitability for EDA.

[1]http://archive.sigda.org/dac2012/contest/dac2012_contest.html

III. Background - EDA, embedding and GNNs

A. From proximity measures to embeddings

The process of node embedding involves learning a free vector $\boldsymbol{e}_v$ for each node. If i, j are nodes sharing some chosen idea of similarity – either proximity or structure-based – $\boldsymbol{e}_i, \boldsymbol{e}_j$ should be encoded similarly in the latent space, with the similarity usually measured by the dot product $\boldsymbol{e}_i \cdot \boldsymbol{e}_j$. Methods that encode proximity similarity include random-walk based embedding methods like Node2vec [14], LINE [15], and DeepWalk [16]. Methods that instead utilize structural similarity to learn embeddings include GraphWAVE [22], Role2vec [23], and struct2vec [24].

B. Random-walk based embedding method

Random-walk based embedding methods like Node2vec, LINE and DeepWalk are widely used in network embedding. These methods are derived from the skip-gram encoding method Word2vec [25] in Natural Language Processing (NLP). The methods encode a proximity relationship, and neighbouring nodes have similar embeddings using these methods. Despite their effectiveness in NLP embedding and network embedding tasks, there are two aspects of EDA that pose difficulties for standard random-walk based methods. First, the typical circuit is extremely large compared to standard graphs in the machine learning literature and comparable to the largest social network graphs. The netlists in industrial level design have hundreds of millions of nodes. Second, in the congestion prediction context, the desired prediction is often on the unseen cells in a new circuit. Thus, the train and test graphs are distinct. As random-walk based embedding only captures the proximity similarities of nodes within the same graph, training and testing on distinct graphs requires extra alignment post-processing [26], [27], which is both challenging and extremely time consuming.

C. Embedding alignment problem

Given a graph $\mathcal{G}$, the d-dimensional output $\boldsymbol{X}$ of a node embedding algorithm optimizes some function of $\langle \boldsymbol{X_i}, \boldsymbol{X_j} \rangle$. The proximities between nodes are preserved if we apply an orthogonal transformation matrix $\boldsymbol{Q}$ ($\tilde{\boldsymbol{X}} = \boldsymbol{XQ}$.) Consider two embeddings $\boldsymbol{X}, \boldsymbol{X'}$ obtained from applying any proximity-based embedding method on two graphs $\mathcal{G}, \mathcal{G}'$. For the sake of simplicity, we consider both to have the same number of nodes. $\boldsymbol{X'}, \boldsymbol{X}$ can differ via the proximity-preserving $\boldsymbol{Q}$.

One way to formulate the graph alignment task is to identify an orthogonal $d \times d$ matrix $\boldsymbol{Q}$ so that we minimize the distance between $\tilde{\boldsymbol{X}} = \boldsymbol{XQ}$ and $\boldsymbol{X}'$. A caveat is that even if $\mathcal{G}$ and $\mathcal{G}'$ are identical, the ordering of the vertices $\mathcal{V}$ and $\mathcal{V}'$ may not be the same. We should therefore consider all possible permutations $\boldsymbol{PX'}$, where $\boldsymbol{P}$ is a permutation matrix of shape $|\mathcal{V}| \times |\mathcal{V}|$.

This formulation is **Wasserstein-Procrustes** alignment, used in network embedding alignment methods such as

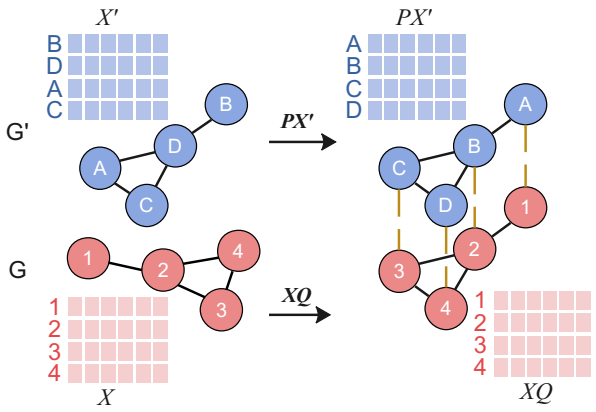

Fig. 3. Wasserstein-procrustes alignment method, such as CONE-ALIGN, uses a permutation matrix and a rotation jointly to align two graphs $\mathcal{G}, \mathcal{G}'$ and embeddings $\boldsymbol{X}, \boldsymbol{X'}$ through $\boldsymbol{PX'}$ and $\boldsymbol{XQ} = \tilde{\boldsymbol{X}}$

CONE-ALIGN [27], and depicted in Figure 3. The problem becomes to find appropriate $\boldsymbol{P}, \boldsymbol{Q}$:

$$\underset{\boldsymbol{P} \in P_n, \boldsymbol{Q} \in O_n}{\arg\min} \boldsymbol{P}, \boldsymbol{Q} \; ||\boldsymbol{XQ} - \boldsymbol{PX'}||^2 \tag{1}$$

Alternate alignment methods include joint factorization (REGAL [26]) and anchor methods (IsoRank [28]).

D. Pointwise Mutual Information (PMI) Matrices

Let $\boldsymbol{X}$, of shape $|\mathcal{V}| \times d$, be the embedding matrix representing the d-dimensional embeddings for all $v \in \mathcal{V}$. The similarity metric between two nodes i, j is measured as $\langle \boldsymbol{X_i}, \boldsymbol{X_j} \rangle$, where $\boldsymbol{X_i}$ denotes the i-th row of $\boldsymbol{X}$. Therefore:

- The similarity pattern for all pairs (i, j) of nodes is fully captured in the matrix $\boldsymbol{XX}^\top$, termed the *PMI matrix* [29], [30]. The (i, j)-th entry of this matrix is equal to $\langle \boldsymbol{X_i}, \boldsymbol{X_j} \rangle$.
- Similarity values for any pair (i, j) are unchanged, and therefore the PMI matrix $\boldsymbol{XX}^\top$ is unchanged, if instead of $\boldsymbol{X}$, the embedding is $\tilde{\boldsymbol{X}} = \boldsymbol{XQ}$, where $\boldsymbol{Q} \in O_{d \times d}$ is orthogonal.

E. PMI Matrix eigendecomposition for network embedding

The PMI matrix $\boldsymbol{XX}^\top$ under the shift $\tilde{\boldsymbol{X}} = \boldsymbol{XQ}$ becomes $(\boldsymbol{XQ})(\boldsymbol{XQ})^\top = \boldsymbol{XQQ}^\top\boldsymbol{X}^\top = \boldsymbol{XX}^\top$. From this viewpoint, if we compare graphs via the PMI matrix, there is no need to search for a suitable $\mathbf{Q}$. Based on this idea, *Matrix-factorization methods* directly factor the PMI matrix – via some unique factorization process and usually approximately via the *top-k eigendecomposition* – to obtain embeddings. We have that $\boldsymbol{XX}^\top \approx \boldsymbol{USU}^\top$, with $\boldsymbol{US^{1/2}}$ being assigned as the embedding. Uniqueness can be guaranteed by taking care of the sign of the eigenvector. With the eigendecomposition factorization strategy, it can also be shown that a permutation of the nodes does not change the similarities calculated via embeddings. In a notable analysis, NETMF [30] unified three popular methods, Node2vec [14], LINE [15], and DeepWalk [16] as implicit factorizations of a PMI matrix. In

this framework, we calculate $\boldsymbol{X}\boldsymbol{X}^\top$ directly, and then conduct matrix factorization via truncated eigendecomposition. Unlike $\boldsymbol{X}$ which must be estimated by random walks, $\boldsymbol{X}\boldsymbol{X}^\top$ often has a closed form, and can be calculated using the adjacency matrix $\boldsymbol{A}$. Runtime is usually faster than random walk based approaches and no explicit embedding alignment is required.

F. Graph Neural Networks

We define a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of vertices, and $\mathcal{E}$ the set of edges. The adjacency matrix $\boldsymbol{A}$ has $\boldsymbol{A}_{ij} = 1$ if there is an edge $(i, j) \in \mathcal{E}$. The graphs are considered as undirected (the adjacency matrix $\boldsymbol{A}$ is symmetric) and homogeneous (all the $v \in \mathcal{V}$ are of the same type). Directed graphs and heterogeneous node sets arise in the literature and can also be solved via GNN variants, but they are not our primary focus in this work.

Similar to deep neural networks which repeatedly apply non-linear activation functions to form their latent representations, GNNs repeatedly apply four stages which are: neighborhood sampling, neighbor information extraction, aggregation, and representation update. The architecture is parametrized layer-wise by two separate weight matrices, $\boldsymbol{W}^{self}$ and $\boldsymbol{W}^{nbd}$. Nodes are given some initial attributes $\boldsymbol{X}$ of shape $|\mathcal{V}| \times d$ where d is the dimensionality of the initial attributes. Via multiple GNN layers, these initial attributes are propagated by the graph structure and transformed into the hidden representations of the GNN, and finally to the target labels. A sample workflow, consisting of the hidden layer iterates $\boldsymbol{h}_v^t$ for each node $v \in \mathcal{V}$ for layer depth $t = 0, \ldots, T$, with the aggregate function being the mean aggregation is shown in Algorithm 1. The schematic of a GNN predictor of our method in demonstrated in Figure 5.

Algorithm 1 Training loop for a GNN using degree, embedding and node attribute features

Input : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, initial input node attributes $\boldsymbol{h}_v^0$, layer depth T

Output : A final set of GNN node representations after T GNN layers $\boldsymbol{h}_v^T, \forall v \in \mathcal{V}$

for $i \leftarrow 1, 2, \ldots T$ **do**
  for $v \in \mathcal{V}$ **do**
    AGGREGATE : $\boldsymbol{m_v^i} = \sum_{u \in \mathcal{N}(v)} \frac{\boldsymbol{h_u^{i-1}}}{|\mathcal{N}(v)|}$
  end for
  for $v \in \mathcal{V}$ **do**
    UPDATE : $\boldsymbol{h_v^i} = \sigma(\boldsymbol{W_v^{self}} \boldsymbol{h_v^{i-1}} + \boldsymbol{W_v^{nbd}} \boldsymbol{m_v^i})$
  end for
end for

IV. Dataset Generation for Congestion Prediction

As a first step, we frame the congestion prediction problem as a problem solvable by GNNs — the *node regression problem*, where continuous-valued labels are provided on some nodes (the training set) and GNNs predict this label on other nodes (the test set) with a held-out node set (validation set).

TABLE I
Details of the Superblue dataset (DAC 2012)

Circuit name	Nodes	Terminals	Nets
Train set			
Superblue2	1014029	92756	990899
Superblue3	919911	86541	898001
Superblue6	1014209	95116	1006629
Superblue7	1364958	93071	1340418
Superblue9	846678	57614	833808
Superblue11	954686	94915	935731
Superblue14	634555	66715	619815
Train graph for ablation study - also in normal train set			
Superblue16	698741	18291	697458
Validation set			
Superblue12	1293433	15349	1293436
Test set			
Superblue19	522775	16678	511685

TABLE II
Details of the OpenRoad dataset

Circuit name	Nodes	Terminals	Nets
Train set			
gcd	343	54	414
ibex	22279	264	24368
aes	23669	388	24458
tinyRocket	33225	269	37250
jpeg	89737	47	94139
bp_multi	93528	1453	110847
Dynamic_node	12112	693	14736
bp_fe	32453	2511	38672
Train graph for ablation study - also in normal train set			
bp	151415	24	179901
Validation set			
bp_be	54591	3029	64829
Test set			
swerv	102034	2039	114079

- We extract two publicly available netlist sets: Superblue circuit line from DAC 2012 [31] which we place via DREAMPLACE [32] as well as a collection of circuits provided with the OPENROAD framework [33]. We convert them into graphs. (Details in Tables I and II.)
- During the global routing phase, we save the position of a cell during an iteration along with the 2-D congestion label and convert the grid congestion value to cell label.
- The dataset is then divided into test and train graphs. We train a GNN to create accurate predictions (in correlation terms) for the test graphs' congestion.

The synthesized design is represented by a netlist of circuit elements. Every circuit element contained in the netlist is regarded as a node in the graph. The edges represent interconnections between circuit elements as defined in the nets of the netlist. We assume the set of nodes in each net is fully connected. Macros and terminals are removed from the graph, because they are normally manually placed in practice and their high node degrees compared to standard cells reduce training efficiency. This leaves only cells as circuit elements. Nets with degree more than 10 are excluded from the final graph as they introduce cliques too large to work with efficiently. The informative cell features (pin number, cell size)

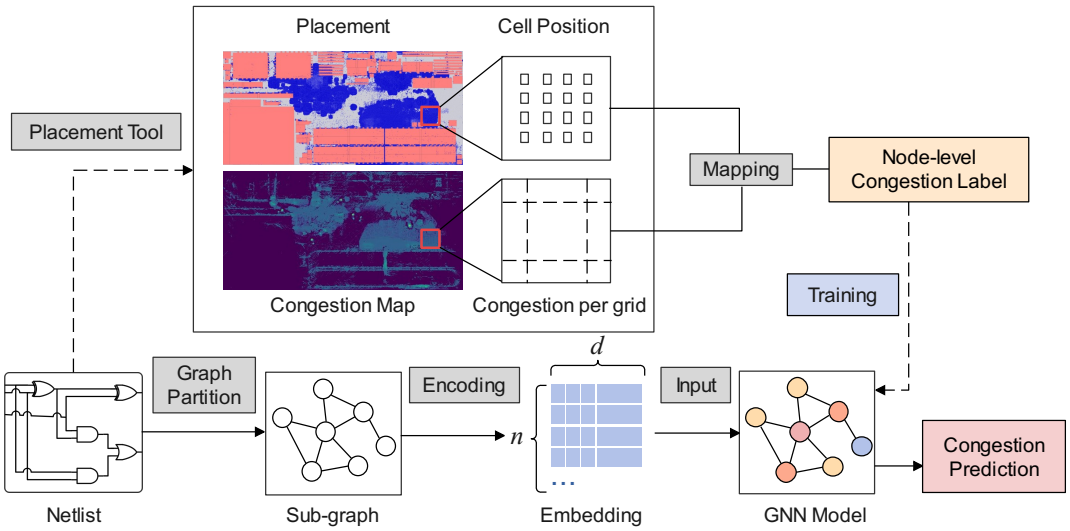

Fig. 4. The training (dashed arrows) and inference (solid arrows) flow of our GNN framework.

are stored as node features and can be used in combination with structural embeddings as GNN inputs. This follows the flow of CongestionNet [7].

The workflow to generate labels for training is shown in Figure 4. For superblue circuit lines, we use DREAMPLACE for placement and NCTUGR 2.0 for congestion map generation, while for OPENROAD datasets, we use the built-in placer RePlAce for placement and FastRoute for congestion map generation [5], [6], [19].

The board is partitioned into grids of size C_x, C_y, with the congestion value for each grid cell computed as the wiring demand divided by the routing capacity. The output along the z-axis is reduced by a max function, with the congestion of layer 0 removed due to lack of routing capacity. The position of every cell (x, y) with grid spacing C_x, C_y can be converted into $(\lfloor \frac{x}{C_x} \rfloor, \lfloor \frac{y}{C_y} \rfloor)$, and we set the label of the cell as the congestion at $(\lfloor \frac{x}{C_x} \rfloor, \lfloor \frac{y}{C_y} \rfloor)$. All labels are normalized to a fixed range of $[-6, 6]$. Our focus is on predicting congestion due to local logic structure, which manifests itself on lower metal layers. Therefore, we use congestion labels from the lower half of the metal layers to train and evaluate the model. Results using overall congestion are also presented for comparison. We now describe the partitioning and embedding steps; it is in these steps where our approach differs most significantly from CongestionNet.

V. Embedding-enhanced GNN training framework

The overview of our proposed framework is shown in Figure 5. We now sequentially describe all the sub-processes for training and inference of a complete GNN model.

A. Sub-graph partition and batch-training

The graph from the netlist of a complete circuit is too large for direct matrix factorization and must be partitioned into clusters. We use the METIS partitioning tool used in the ClusterGCN [34] framework to partition graphs for training. In METIS, the graph is coarsened by repeatedly merging nodes that are connected by highly weighted edges. After coarsening, the graph is bisected and resolved to its original position iteratively. The bisection is expanded to obtain a K-way partition. We set the clustering configuration to yield clusters of $\approx$ 5000 nodes each, with all superblue circuits being partitioned 150 times and OPENROAD circuits set to the closest integer to reach the target cluster size. The combination of matrix factorization with clustering in a GNN is, to our knowledge, novel even outside the EDA context.

B. Matrix Factorization for embedding generation

Given a partition P with associated adjacency, degree and Laplacian matrices $\boldsymbol{A_P}, \boldsymbol{D_P}, \boldsymbol{L_P}$, we use a modified version of the InfiniteWalk [18] matrix factorization method to form the embedding, with $\mathbf{1}\mathbf{1}^\top$ the all-ones matrix, and $L, T > 0, H \geq L$ being hyperparameters:

- $\boldsymbol{M'_P} = \mathbf{1}\mathbf{1}^\top + Tr(\boldsymbol{D_P})\boldsymbol{D_P^{-1/2}}\boldsymbol{L_P}\boldsymbol{D_P^{-1/2}}$.
- $\boldsymbol{M''_P} = \mathbf{1}\mathbf{1}^\top + \frac{\boldsymbol{M'_P}}{T}$.
- Clamp $\boldsymbol{M''_P}$ to range $[L, H], L > 0$.
- $\boldsymbol{M''_P} \leftarrow \log(\boldsymbol{M''_P})$ (entrywise log).

The matrix $\boldsymbol{M''_P}$ is the PMI matrix, and a truncated eigen-decomposition yields the embedding. The hyperparameter choice depends on the graph structure. While L, H control the numerical stability, T controls the extent to which a node may influence its neighbour. High values of T generally lead to similar embeddings graph-wide, an undesirable outcome termed **oversmoothing**, which harms performance.

The factorization generates a K-dimensional embedding vector for every cell. This is run for every partition generated by the METIS step and the embeddings are cached. The embeddings are comparable between partitions due to the usage of PMI matrices and require no extra alignment.

For use with node attributes $\boldsymbol{X}$, the embeddings $\boldsymbol{E}$ and attributes are concatenated. These are fed into a special GNN

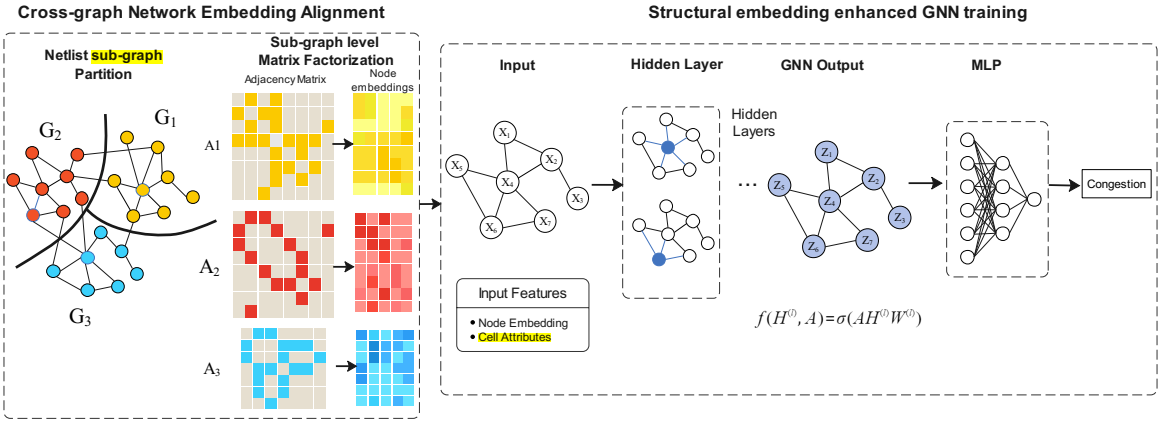

Fig. 5. Our system design for structural embedding-enhanced GNN with a MLP post predictor for congestion prediction

architecture called SAGE [10]. For the detailed formulation of SAGE please refer to Section III-F. To further refine the final representation with information passed directly from the original node features, the embedding and the attribute are concatenated with the final layer of SAGE output and fed into an MLP to generate the final congestion estimate $\hat{Y}$:

$$\hat{Y} = \text{MLP}\Big([[\boldsymbol{X};\boldsymbol{E}];\text{SAGE}([\boldsymbol{X};\boldsymbol{E}])]\Big),$$

where $[\ ;\ ]$ represents concatenation.

VI. Training and Inference Details and Results

A. GNN architecture details

Graph neural networks are prone to oversmoothing, where the predictions for all nodes become very similar, and this arises more often in deeper GNNs. Accordingly, as opposed to CongestionNet which uses 16 hidden states and a depth 8 GAT [7], [35], we use SAGE [10] (SAmple and aGgregatE) convolutions to construct our GNNs and set them to have 2 hidden layers of size 200 and 160. The input to the GNN is the node embedding concatenated with the attributes. The output of the last hidden layer is concatenated with the input and fed to an MLP with two hidden layers, each of size 150, to obtain the final congestion predictions. Squared error loss is used to train with ADAM [36] as the optimizer, with each minibatch containing one cluster. The parameters L, H are set to $10^{-10}, 10^{6}$ respectively. The dimension of the embedding is set to 4.

B. Training and inference

During training, the loss is optimized over the nodes of the graph, and accuracy on the nodes defines our graph level metrics. During inference, an unseen graph is provided and the embedding is computed. We do not need the additional alignment step required for using proximity based embedding here. Once the new graph's embedding is computed, we concatenate it with the new graph's attributes (if available) and output a graph level prediction. The unseen graph has a ground truth position provided in the evaluation phase. For evaluation, along with graph level we consider a **reconstructed grid**, which consists only of the grid cells with at least one non-terminal node, and is reconstructed by averaging the labels and predictions of nodes in a grid cell.

In our experiments, we consistently found that :

- The usage of average labels, where each cell instead inherits the congestion in $(\lfloor\frac{x}{C_x}\rfloor, \lfloor\frac{y}{C_y}\rfloor)$ divided by the total number of cells in $(\lfloor\frac{x}{C_x}\rfloor, \lfloor\frac{y}{C_y}\rfloor)$, is not superior in terms of final performance, although it does help in terms of rapidly converging to the optimum.
- The usage of pin features, beyond simply counting the pins of each cell, does not help the outcome, and usually prevents good fitting of models.
- The addition of terminals back to the graph also hinders training of GNNs. This may be understood to arise from the size of the macros causing problems in label creation for the graph.

C. Correlation metrics for evaluation

We use the following three metrics of correlation to measure performance, evaluated both on graph and grid level.

- Pearson correlation coefficient (PCC). Given two random variables X, Y, the PCC between X, Y is defined as $E[\tilde{X}\tilde{Y}]$, where $\tilde{X} = \frac{X - E[X]}{\sqrt{Var(X)}}$ and similarly for Y.
- Spearman correlation, which calculates PCC but replaces X, Y with their rank among all X, Y in observed sample.
- Kendall correlation : Let N samples be present of X, Y. Two pairs (X_i, X_j) and (Y_i, Y_j) are concordant if $(X_i - X_j)(Y_i - Y_j) > 0$. That is, if $X_i > X_j$ then $Y_i > Y_j$ and vice versa. Kendall correlation is the difference between concordant and non-cordant pairs. divided by the total number of pairs $\binom{N}{2}$. This is the only metric used in [7]. We add the other two metrics for more complete evaluation as especially Pearson can capture raw values which Spearman and Kendall cannot.

Before evaluation, both the prediction and the label have some (very low) noise added to them. This is to break ties randomly, as the demand values are discrete and in this situation the concordance of pairs is not meaningful. ???

TABLE III
RESULTS OBTAINED ON ALL SUPERBLUE CIRCUITS (DAC 2012)

Congestion prediction study correlation metrics												
	Lower level congestion						Overall congestion					
Methods	Pearson		Spearman		Kendall		Pearson		Spearman		Kendall	
	Node	Grid	Node	Grid	Node	Grid	Node	Grid	Node	Grid	Node	Grid
Adhesion metric	0.09	0.16	0.06	0.20	0.06	0.14	0.08	0.16	0.07	0.20	0.05	0.15
Neighbourhood metric	0.02	0.04	0.18	0.27	0.13	0.18	0.03	0.08	0.18	0.27	0.13	0.18
GTL metric	0.02	0.01	0.14	0.23	0.10	0.16	0.01	0.05	0.14	0.24	0.10	0.16
CongestionNet	0.26	0.35	0.27	0.33	0.19	0.24	0.19	0.27	0.24	0.26	0.17	0.22
Embedding-enhanced GNN (ours)	0.31	0.43	0.34	0.44	0.25	0.31	0.24	0.33	0.26	0.38	0.20	0.27
Embedding ablation findings - train set reduced to one graph												
GNN (no embedding)	0.27	0.36	0.30	0.40	0.19	0.27	0.21	0.28	0.22	0.32	0.15	0.23
Embedding-enhanced GNN (ours)	0.30	0.41	0.32	0.42	0.24	0.30	0.22	0.30	0.24	0.35	0.18	0.26
Comparing embeddings only												
LINE (aligned)	0.10	0.14	0.15	0.21	0.09	0.15	0.06	0.09	0.12	0.22	0.09	0.16
LINE (unaligned)	0.02	0.06	0.03	0.06	0.02	0.04	0.03	0.06	0.04	0.05	0.01	0.03
Node2Vec(aligned)	0.10	0.15	0.10	0.14	0.07	0.10	0.09	0.13	0.10	0.12	0.08	0.10
Node2Vec(unaligned)	0.03	0.07	0.02	0.06	0.01	0.04	0.05	0.07	0.04	0.05	0.03	0.04
DeepWalk(aligned)	0.09	0.15	0.12	0.19	0.06	0.11	0.08	0.12	0.10	0.13	0.09	0.12
DeepWalk(unaligned)	0.00	0.04	0.02	0.04	0.02	0.03	0.02	0.02	0.03	0.02	0.02	0.01
GNN(embedding only)	0.16	0.25	0.17	0.26	0.15	0.20	0.14	0.17	0.16	0.25	0.13	0.17

TABLE IV
RESULTS OBTAINED ON ALL OPENROAD CIRCUITS

Congestion prediction study correlation metrics												
	Lower level congestion 底层						Overall congestion 所有layer					
Methods	Pearson		Spearman		Kendall		Pearson		Spearman		Kendall	
对比实验	Node	Grid	Node	Grid	Node	Grid	Node	Grid	Node	Grid	Node	Grid
Adhesion metric	0.12	0.13	0.09	0.14	0.07	0.10	0.10	0.12	0.06	0.12	0.05	0.09
Neighbourhood metric	0.13	0.12	0.08	0.13	0.10	0.06	0.12	0.10	0.06	0.13	0.05	0.10
GTL metric	0.10	0.12	0.08	0.14	0.06	0.10	0.09	0.10	0.06	0.13	0.05	0.10
CongestionNet	0.20	0.23	0.16	0.16	0.13	0.11	0.17	0.19	0.15	0.14	0.12	0.10
Embedding-enhanced GNN (ours)	0.23	0.26	0.20	0.19	0.18	0.17	0.19	0.22	0.16	0.16	0.13	0.12
Embedding ablation findings - train set reduced to one graph												
GNN (no embedding)	0.18	0.20	0.15	0.14	0.12	0.10	0.13	0.15	0.11	0.09	0.08	0.10
Embedding-enhanced GNN (ours)	0.20	0.23	0.17	0.17	0.13	0.11	0.14	0.18	0.12	0.14	0.09	0.10
Comparing embeddings only												
LINE (aligned)	0.10	0.11	0.09	0.07	0.07	0.05	0.10	0.11	0.09	0.08	0.07	0.06
LINE (unaligned)	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.01	0.01	0.01	0.01	0.01
Node2Vec(aligned)	0.11	0.12	0.09	0.07	0.07	0.05	0.12	0.12	0.10	0.07	0.08	0.05
Node2Vec(unaligned)	0.01	0.01	0.02	0.00	0.01	0.00	0.02	0.03	0.03	0.01	0.02	0.01
DeepWalk(aligned)	0.10	0.11	0.08	0.06	0.06	0.04	0.12	0.13	0.10	0.07	0.08	0.05
DeepWalk(unaligned)	0.02	0.01	0.01	0.02	0.01	0.02	0.01	0.01	0.01	0.02	0.01	0.02
GNN(embedding only)	0.13	0.15	0.11	0.15	0.09	0.11	0.12	0.13	0.10	0.14	0.08	0.10

D. Learning-free structure-based benchmarks

We use as benchmarks three methods which are also found in [7]. In each, a value is computed per cell and the corresponding correlation metrics computed using this value as a congestion prediction. In case of cross-validation of parameters, lower-level grid Kendall correlation coefficient is used to order settings of computation.

- Neighborhood size: The k-th neighborhood size of a node v is the number of nodes, reachable from v, within geodesic distance k. k is varied from 1 to 5 and the best performing value is taken.
- GTL (Graph Tangled Logic) : This refers to a measure that examines the structure of the graph cut around node v. Parametrized by a term called the **rent exponent**, it seeks to adjust for the effect of cluster size. We cross-validate and report only the result with the best rent exponent.
- Adhesion : We grow a local neighbourhood around a node v and compute all min-cuts between the node and other nodes in the neighbourhood, and take the maximum min-cut as indicative of connectivity around v. The neighbourhood size is varied between up to 1 and 5 nodes away and the best value is reported.

E. Key results on Superblue and OPENROAD

The aim of our results is to compare with the un-augmented graph convolutional network of the same structure, as well as to compare with the previously proposed CongestionNet [7]. In comparison with these benchmarks, our augmented network has the best performance. Results on Superblue and OPENROAD are respectively provided in tables III and IV. All results are reported using a Tesla V100 GPU and run on a pytorch [37] + Deep Graph Library (DGL) [38] codebase.

TABLE V
RUNTIME COMPARISON (SECONDS) BETWEEN SUBGRAPH-LEVEL TRAINING AND BLOCK SAMPLING

Architecture runtime comparisons (per graph per epoch)				
	Superblue		OPENROAD	
GNN architecture	Training time	Inference time	Training time	Inference time
No partitioning (ours + block sampler)	56.2	65.4	9.8	14.1
With partitioning				
Our architecture	2.2	6.1	0.22	2.5
CongestionNet	6.4	8.7	0.78	3.8

TABLE VI
MATRIX FACTORIZATION RUNTIME VS OTHER EMBEDDING METHODS IN SECONDS

Embedding runtime comparisons				
Embedding method	Train time	Alignment time	Train time	Alignment time
Node2vec	250.6	1750.4	45.2	733.5
LINE	167.8	1355.2	19.7	802.1
DeepWalk	143.7	1566.5	22.1	783.4
Ours	80.4	-	18.2	-

We also provide comparison of the InfiniteWalk embedding as an input to the GCN versus embeddings from Node2vec, LINE, and DeepWalk, all aligned using the CONE-ALIGN method. The InfiniteWalk embedding is shown to provide superior results. We provide an unaligned case as a simple comparison to highlight the need for alignment. For multi-graph alignment, the choice of anchor graph may not be clear. As such, the training and alignment for the embedding-only case is done with exactly one training graph — superblue16 and bp for DREAMPLACE and OPENROAD respectively. The results are thus a lower bound on the performance achievable with InfiniteWalk-like embedding alone. Each embedding dimension is set to 32. InfiniteWalk-like embedding generally achieves the best performance even over the aligned proximity embeddings, as well as improving attribute-equipped GNNs. The improvement on top of attributes ranges from 10 to 20 %. Relative to aligned embeddings, InfiniteWalk can be up to 60% more effective outperforms non-learning benchmarks.

F. Runtime comparison

Using clustering and partitioning the graph leads to significant improvements in runtime when compared to conventional samplers. These comparisons are made in Tables V and VI. The without-partition version is constructed using a Multi-layer block sampler. In particular, note that in terms of embedding comparisons, our InfiniteWalk-derived embedding is far faster due to requiring no alignment time. This is actually an **underestimate** of the alignment time, due to the numerical instability of the optimization problem requiring extra time for tuning the correct regularizer value. Clustering by itself can cut up to 90% of epoch time or more, and avoiding the alignment step offers a similar speedup.

attribute of cells???

VII. CONCLUSION, COMPARISON, POSSIBLE EXTENSIONS

In this work, we provide the GNN-based method for predicting congestion from a netlist that can operate without any useful attribute of cells, and solely on netlist structure. Our approach is inspired by the fact that classic methods for congestion estimation such as GTL [11], which rely solely on structural properties, are able to pinpoint congested cells. This suggested that embedding learning for circuits might be able to capture the aspects that allowed these previous graph-based predictors to do well. Crucially, by adding the Pearson correlation, we also seek to measure the amount of prediction for the raw congestion value, and not just the rank.

Our most important finding is that the most popular embedding learning methods, which are otherwise dominant in GNN-related works, do not perform particularly well in the EDA task. The success of matrix factorization paired with clustering is a novel finding in the EDA context, and potentially even outside of it. Comparison with previous methods, such as CongestionNet, shows that we improve on key aspects such as correlation metrics and runtime. This makes our method attractive even when there are attributes available.

There are currently no theoretical explanations for the success of matrix factorization when clustering is employed. We believe that understanding the types of graph structures which allow this kind of processing is important. From a practical perspective, it should be noted that proximity based methods constitute the majority of embedding research in the machine learning community, and alignment methods are also rapidly improving. Therefore, there are possible pairings of proximity based random walk embedding methods with alignments that might compete with matrix factorization. While we achieve superior performance with matrix factorization methods, random walk methods are superior in terms of scalability in memory, while matrix factorization methods are superior in terms of runtime scalability. There should be a smooth tradeoff between these two methods, which deserves further exploration. 未来工作

Recently, there has been interest in not only the prediction of congestion, but using such prediction as feedback [39] to placement algorithms [40] to control the placement process. Although we have demonstrated successful prediction, it remains an open challenge to integrate our predictor for the control problem and improve metrics such as wirelength in the final (post-detailed routing and placement) design.

REFERENCES

[1] T. Lin and C. Chu, "Polar 2.0: An effective routability-driven placer," in *Proceedings of the 51st Annual Design Automation Conference*, 2014, pp. 1–6.

[2] C. Li, M. Xie, C.-K. Koh, J. Cong, and P. H. Madden, "Routability-driven placement and white space allocation," *IEEE Transactions on Computer-aided design of Integrated Circuits and Systems*, vol. 26, no. 5, pp. 858–871, 2007.

[3] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren, S.-Y. Fang, Y. Chen, and J. Hu, "Routenet: Routability prediction for mixed-size designs using convolutional neural network," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.

[4] P. Spindler and F. M. Johannes, "Fast and accurate routing demand estimation for efficient routability-driven placement," in *2007 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2007, pp. 1–6.

[5] K.-R. Dai, W.-H. Liu, and Y.-L. Li, "Nctu-gr: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-d global routing," *IEEE Transactions on very large scale integration (VLSI) systems*, vol. 20, no. 3, pp. 459–472, 2011.

[6] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "Nctu-gr 2.0: Multithreaded collision-aware global routing with bounded-length maze routing," *IEEE Transactions on computer-aided design of integrated circuits and systems*, vol. 32, no. 5, pp. 709–722, 2013.

[7] R. Kirby, S. Godil, R. Roy, and B. Catanzaro, "Congestionnet: Routing congestion prediction using deep graph neural networks," in *2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)*. IEEE, 2019, pp. 217–222.

[8] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[9] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[10] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proc. Adv. Neural Inf. Proc. Systems*, 2017.

[11] T. Jindal, C. J. Alpert, J. Hu, Z. Li, G.-J. Nam, and C. B. Winn, "Detecting tangled logic structures in vlsi netlists," in *Proceedings of the 47th Design Automation Conference*, 2010, pp. 603–608.

[12] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *arXiv preprint arXiv:1310.4546*, 2013.

[13] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[14] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, 2016, pp. 855–864.

[15] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th international conference on world wide web*, 2015, pp. 1067–1077.

[16] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2014, pp. 701–710.

[17] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Proceedings of the eleventh ACM international conference on web search and data mining*, 2018, pp. 459–467.

[18] S. Chanpuriya and C. Musco, "Infinitewalk: Deep network embeddings as laplacian embeddings with a nonlinearity," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1325–1333.

[19] Y. Xu, Y. Zhang, and C. Chu, "Fastroute 4.0: Global router with efficient via minimization," in *2009 Asia and South Pacific Design Automation Conference*. IEEE, 2009, pp. 576–581.

[20] M. Saeedi, M. S. Zamani, and A. Jahanian, "Prediction and reduction of routing congestion," in *Proceedings of the 2006 international symposium on Physical design*, 2006, pp. 72–77.

[21] P. Kudva, A. Sullivan, and W. Dougherty, "Metrics for structural logic synthesis," in *IEEE/ACM International Conference on Computer Aided Design, 2002. ICCAD 2002.* IEEE, 2002, pp. 551–556.

[22] C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec, "Learning structural node embeddings via diffusion wavelets," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1320–1329.

[23] N. K. Ahmed, R. Rossi, J. B. Lee, T. L. Willke, R. Zhou, X. Kong, and H. Eldardiry, "Learning role-based graph embeddings," *arXiv preprint arXiv:1802.02896*, 2018.

[24] L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo, "struc2vec: Learning node representations from structural identity," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 385–394.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[26] M. Heimann, H. Shen, T. Safavi, and D. Koutra, "Regal: Representation learning-based graph alignment," in *Proceedings of the 27th ACM international conference on information and knowledge management*, 2018, pp. 117–126.

[27] X. Chen, M. Heimann, F. Vahedian, and D. Koutra, "Cone-align: Consistent network alignment with proximity-preserving node embedding," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1985–1988.

[28] R. Singh, J. Xu, and B. Berger, "Global alignment of multiple protein interaction networks with application to functional orthology detection," *Proc. of the National Academy of Sciences*, vol. 105, no. 35, pp. 12 763–12 768, 2008.

[29] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," in *Proc. Adv. Neural Inf. Proc. Systems*, 2014.

[30] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec," in *Pro. of Int. conf. on web search and data mining*, 2018.

[31] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei, "The dac 2012 routability-driven placement contest and benchmark suite," in *DAC Design Automation Conference 2012*. IEEE, 2012, pp. 774–782.

[32] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020.

[33] T. Ajayi and D. Blaauw, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," in *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

[34] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, "Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 257–266.

[35] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *Proc. Int. Conf. Learning Representations*, 2018.

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

[37] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[38] M. Wang, L. Yu, D. Zheng, Q. Gan, Y. Gai, Z. Ye, M. Li, J. Zhou, Q. Huang, C. Ma *et al.*, "Deep graph library: Towards efficient and scalable deep learning on graphs." 2019.

[39] J. Chen, J. Kuang, G. Zhao, D. J.-H. Huang, and E. F. Young, "Pros: A plug-in for routability optimization applied in the state-of-the-art commercial eda tool using deep learning," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2020, pp. 1–8.

[40] S. Liu, Q. Sun, P. Liao, Y. Lin, and B. Yu, "Global placement with deep learning-enabled explicit routability optimization," *IEEE/ACM Proceedings Design, Automation and Test in Eurpoe (DATE)*, 2021.