# TAG: Learning Circuit Spatial Embedding From Layouts

Keren Zhu[1], Hao Chen[1], Walker J. Turner[2], George F. Kokai[2], Po-Hsuan Wei[2],
David Z. Pan[1], and Haoxing Ren[2]
[1]ECE Department, The University of Texas at Austin, Austin, TX, USA
[2]NVIDIA Corporation, USA
keren.zhu@utexas.edu, dpan@ece.utexas.edu, haoxingr@nvidia.com

## ABSTRACT

Analog and mixed-signal (AMS) circuit designs still rely on human design expertise. Machine learning has been assisting circuit design automation by replacing human experience with artificial intelligence. This paper presents TAG, a new paradigm of learning the circuit representation from layouts leveraging **T**ext, self **A**ttention and **G**raph. The embedding network model learns spatial information without manual labeling. We introduce text embedding and a self-attention mechanism to AMS circuit learning. Experimental results demonstrate the ability to predict layout distances between instances with industrial FinFET technology benchmarks. The effectiveness of the circuit representation is verified by showing the transferability to three other learning tasks with limited data in the case studies: layout matching prediction, wirelength estimation, and net parasitic capacitance prediction.

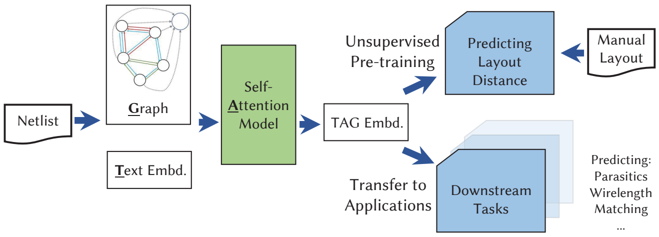


**Figure 1: An illustration of the proposed circuit representation learning paradigm.**

## 1 INTRODUCTION

The performance of analog and mixed-signal (AMS) integrated circuit designs are sensitive to parasitics, process variation, and layout-dependent effects. Today, AMS circuit design, from schematic to layout, is still mainly a manual, time-consuming, and error-prone task.

AMS circuits often impose specific parasitics and mismatch requirements on their layout implementation, where designers leverage their prior experience to place devices in specific patterns and configurations to reduce parasitics, the effects of local variation gradients, and layout-dependent effects. Lacking the techniques to mimic such an experience automatically is one of the main bottlenecks in automating AMS design flow [1].

Researchers have attempted to apply machine learning (ML) to AMS IC designs [2]. Several studies use graph neural network (GNN) on circuit graphs to learn the symmetry constraints in layouts [3, 4]. The work [5] uses GNN to identify the type of AMS circuits, such as amplifiers and filters, to select layout templates for each circuit. Researchers also represent schematics with graphs and use GNN for the analog device sizing problem [6, 7]. Wang et al. [8] and Li et al. [9] uses GNN to learn netlist representations based on their logic functionality. The GNN-based ML frameworks decide transistors' width and length parameters based on the feedback from pre-layout simulations. Netlists are essentially hyper-graphs, making GNN a solution candidate that many prior studies have adopted. The underlying GNNs, in some sense, are expected to learn to capture the circuit representation. Outside the AMS circuit domain, there are also attempts to learn the graph-structured circuit representation. Wu et al. [10] investigated learning on high-level synthesis codes with GNN. Several studies apply ML to learn source code representation [11, 12]. However, despite its wide adoption, circuit representation learning is seldom studied as an individual problem. The underlying neural network models are trained with different targets in the individual applications. In this work, we propose a new paradigm to learn AMS circuit representation without additional manual labeling by leveraging the layout/placement data directly.

Circuits are commonly represented as graphs, and existing learning algorithms apply GNN on the circuit graph. Nonetheless, the knowledge of the graph representation is limited to device connectivity only. However, detailed information is readily available in the circuit netlist in the form of the device, instance, and net names, where designers often use specific naming conventions to detail and organize the netlist to be more human readable. To leverage this information, We adopt the natural language model in the representation learning process. On the other hand, the graph convolution mechanism is usually limited in capturing a global view of the entire circuits. To address this, we also adopt a sub-circuit-wise self-attention mechanism to integrate the whole picture of the sub-circuit into the resulting embedding.

In this paper, we propose TAG (**T**ext, Self-**A**ttention and **G**raph), a framework that learns the AMS circuit representation from layout positions. Inspired by the success of the pre-trained model in natural language models (e.g., BERT [13]), the TAG framework pre-trains a learning model on a larger layout dataset first. Then the pre-trained circuit instance embeddings can be used for other learning tasks with limited data. Figure 1 shows our proposed circuit learning paradigm. A design database of netlists and corresponding manual layouts are used to pre-train a TAG model. The pre-trained TAG models are then transferable to multiple applications in analog

*ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA*
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549384

CAD. This work attempts to establish a common learning representation for analog circuits. The main contribution of this work is summarized as follows.

- A framework, TAG, to learn and embed the circuit instance representation from layout data without additional manual labeling is presented. It learns the spatial relations of instances in the embedding and assists in transferring to other learning tasks.
- A novel methodology of incorporating the circuit netlist text information, such as the instance and type names, from netlists into the learning task is proposed.
- A circuit embedding network combining a multi-head self-attention layer with GNN is presented. The proposed usage of the self-attention mechanism allows the resulting instance embedding to reflect a better global view of the circuits.
- Experimental results show TAG significantly outperforms the existing methods in the accuracy of predicting relative layout distance. TAG also demonstrates great effectiveness in transferring to three other learning tasks: layout matching prediction, wirelength estimation, and net parasitic capacitance prediction.

The remainder of this paper is organized as follows. Section 2 gives the preliminaries. Section 3 details the proposed TAG framework. Section 4 presents the experimental results, and Section 5 concludes the paper.

## 2 PRELIMINARIES

In this section, we introduce the convolutional graph neural network (Section 2.1). Then we describe the circuit hierarchical structure and formulate our learning target: the relative instance distance (Section 2.2). In the end, we overview three applications that are used for case studies in the experiments (Section 2.3).

### 2.1 Convolutional Graph Neural Network

Convolutional graph neural networks (ConvGNN) are widely used for graph-structured data. ConvGNNs perform *convolution* on graph structures to obtain new node embeddings. For a node $n_i$, a graph convolution operation aggregates the current embedding or feature of $n_i$'s neighboring nodes as shown in the Equation (1),

$$\begin{aligned}\mathbf{a}_i^{l+1} &= \text{AGGREGATE}^l\left(\left\{\mathbf{h}_j^l : u \in \mathcal{N}_i\right\}\right),\\ \mathbf{h}_i^{l+1} &= \text{COMBINE}^l(\mathbf{h}_i^l, \mathbf{a}_i^{l+1}),\end{aligned} \tag{1}$$

where $\mathbf{h}_i^l$ is the $l^{\text{th}}$ layer output embedding for $n_i$, $\mathcal{N}_i$ indicates the neighbors of node $n_i$. The choice of AGGREGATE$(\cdot)$ and COMBINE$(\cdot)$ functions vary in ConvGNN layer designs. A typical practice is to use pooling functions, such as max and mean, and linear transformations. After passing through the graph convolution layers, the node embedding can be used for some downstream prediction tasks.

While ConvGNNs have demonstrated success in many applications, there are several limitations in representative ConvGNN architectures. First, the knowledge learned from ConvGNNs tends to have a strong locality. ConvGNNs usually work by aggregating neighboring information, which results in similar behavior to low-pass filters on the graph spectral domain [14]. Therefore, ConvGNNs sometimes may lack a global view of the graph. Second, ConvGNNs can hardly distinguish locally isomorphic structures. It becomes an issue as AMS designs frequently contain symmetric or parallel local structures.

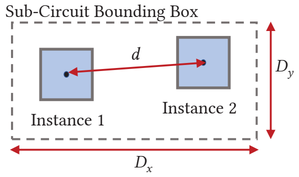


**Figure 2: An example of the layout distance.**

### 2.2 Sub-Circuits and Relative Layout Distance

AMS circuit designs are intrinsically hierarchical. Both the schematic and the layout design are usually implemented hierarchically. A sub-circuit, such as a current mirror and an OTA, can function individually and be used as the building blocks for different top-level circuits. On the other hand, normalizing the learning task to the sub-circuit scale allows a more general inductive basis in the learning model. It benefits the transferability of the ML model to circuits with different scales. Therefore we focus on the sub-circuit-level in our unsupervised training scheme.

AMS layout requires careful considerations of parasitics, matching, area, power, etc. Specifically, layout constraints are commonly employed to ensure proper matching, device interdigitation, symmetry placement, and distances to critical signal paths. We propose learning distance between instances as it is a crucial measurement from layout implementation and containing the design expertise. Depending on the hierarchy level and the design, the instances can be either a primitive device (e.g., transistor) or a sub-circuit (e.g., OTA). To allow the learning model to work for arbitrary circuits, we normalize the distance by its parent circuit bounding box so that its value is between 0 and 1. Figure 2 shows an example of the layout distance. The distance $d$ between every instance pair is normalized to $\hat{d} = d/\sqrt{D_x^2 + D_y^2}$, where $D_x$ and $D_y$ denotes the width and height of its parent sub-circuit layout bounding box. The normalized distance is used as our training target. Such practice also motivates our ML model to homogeneously learn the knowledge between different hierarchy levels.

The relative layout distance prediction learning task is formulated as follows.

**Problem 1** (Relative Layout Distance Prediction)**.** Given a circuit design $D$ with hierarchy tree structure with a set of sub-circuits $C$, predict the relative distance $\hat{d}$ in the manual layout implementations between all instance pairs $(\{I_i, I_j\} \in I)$ in the same sub-circuit $C_i \in C$.

### 2.3 Applications in Analog CAD

In this paper, we introduce three downstream applications as case studies to evaluate our proposed circuit embedding.

*2.3.1 Layout Matching Detection.* Identifying matching constraints in sub-circuits is crucial for fully-automated layout syntheses [15]. The matched instances are placed in certain matching patterns, such as symmetry and common-centroid. The identification problem can be formulated as a binary classification problem, for which GNN is recently leveraged to solve [3, 4, 16].

In the case study, we formulate the layout matching detection problem as follows.

**Problem 2** (Layout Matching Detection)**.** Given a circuit design $D$ with a hierarchy tree structure with a set of sub-circuits $C$, for every instance pairs $(\{I_i, I_j\} \in I)$ that in the same sub-circuit $C_i \in C$, predict whether it is forming a symmetry, common centroid, or interdigitation patterns in the human layout implementation.

*2.3.2 Wirelength Estimation.* A priori wirelength estimation is a classical problem in VLSI design automation [17]. It guides the early design stages. Modern algorithms leverage ML techniques to increase the accuracy of the estimator [18].

In the current analog layout synthesis framework, the weights of nets and the proximity of instances are usually treated as human-specified parameters. Finding a suitable set of parameters requires design expertise and trial and error [19]. A wirelength estimator can assist this process.

In the case study, we formulate the wirelength estimation problem as follows.

**Problem 3** (Wirelength Estimation)**.** Given a circuit design $D$ with a hierarchy tree structure with a set of sub-circuits $C$, for every net $n_i$ in the same sub-circuit $C_i \in C$, predict its half-perimeter wirelength (HPWL) in the human layout implementation.

*2.3.3 Net Parasitic Capacitance Prediction.* Predicting post-layout parasitics from the schematic is an important problem in advanced technology nodes where the mismatch of pre-layout simulation and post-layout performance is significant. Researchers have introduced ML methods to tackle the problem [20, 21]. By predicting the post-layout parasitics from schematics, those methods reduce the error of pre-layout simulation and accelerate the design cycle.

The state-of-the-art algorithm, ParaGraph [21], introduces GNN to the problem. For each parasitic type, such as net capacitance, it trains multiple models for a different range of values. Each model is specified with a maximum prediction value ($max_v$). The models are then merged using the ensemble modeling technique to produce the final prediction. Such methodology benefits the overall accuracy by allowing the models to focus on a small range of magnitude of regression targets.

We apply our TAG embedding in our experiments to the most representative parasitics prediction task: the net capacitance prediction problem. We formulate the problem as follows.

**Problem 4** (Net Parasitic Capacitance Prediction)**.** Given a circuit design $D$ with a hierarchy tree structure with a set of sub-circuits $C$, for every net $n_i$ in the flatten netlist $\hat{C}$, predict its post-layout total parasitic capacitance in human layout implementation.

## 3 TAG ALGORITHMS

TAG's circuit embedding network architecture comprises a GNN and a multi-head self-attention layer (MSA). The GNN model works on the entire hierarchical circuit to obtain the initial embeddings. To mitigate the locality of the GNN model, we use the MSA layer on the sub-circuit instances to allow the resulting embeddings to consider the entire sub-circuit. We add instance text embeddings during the MSA step to provide an additional dimension of knowledge. We train the embeddings by regressing to relative layout distance.

In the rest of this section, we present the details of the algorithms. The graph structure for GNN learning is shown in Section 3.1. The instance input features are described in Section 3.2. The embedding network architecture is presented in Section 3.3. Finally, we introduce the learning algorithm for relative layout distance regression in Section 3.4.

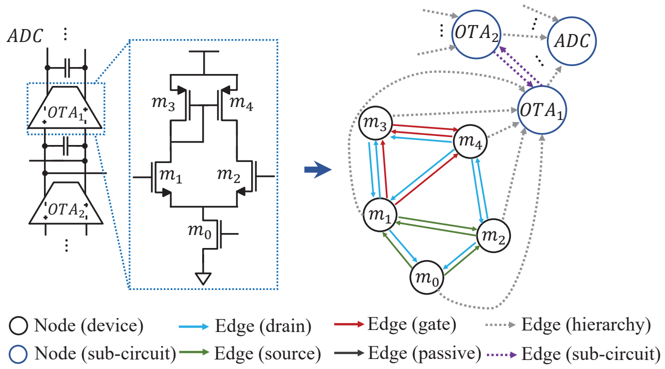


**Figure 3: An example of the graph representation for an AMS circuit.**

### 3.1 Heterogeneous Hierarchical Graph Construction

We propose to use a heterogeneous hierarchical Graph $G = (V, E)$ to represent a circuit. At the device level, we adopt a similar approach to the work [3]. Each device is represented as a node, and the nets are decomposed into two-pin pairs. A directed edge $e = (u, v, \tau_v) \in E$ indicates the interconnection from vertex $u$ to $v$ with edge type $\tau_v$. The edge type denotes the type of the connected port of $v$ in $e$. The port type can be the gate, drain, source, passive device, and sub-circuit. The power and ground nets are not extracted into the graph as they trivialize the graph by connecting most of the nodes. We exclude the dummy and decap devices in the graph and learning process as they are mainly used to compensate for layout effects instead of functioning in circuits.

Different from the work [3], the circuit hierarchy is incorporated in the graph. Each sub-circuit is also represented as a node in the graph. A directed edge $e = (u, v, \tau_{hier})$ is added from the child node $u$ to its parent parent$(v)$. The backward parent-children edges are not added with the assumption that the circuit implementations are bottom-up.

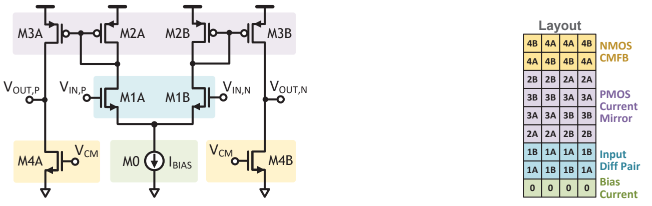
Figure 3 illustrates an example of the proposed graph representation. The example includes three hierarchy levels. Nodes $m_0, \ldots, m_4$ represent the transistors composing $OTA_1$. One hierarchy edge directs from every transistor node to the $OTA_1$ sub-circuit node denoting the hierarchy. On a higher level, the interconnections between $OTA_1$ and $OTA_2$ are represented with the two-pin pair model similar to the leaf nodes. The proposed graph representation maintains a homogeneous structure between different hierarchy levels and enables our learning model to apply to device-level and circuit-level prediction.

### 3.2 Instance Feature Initialization

In TAG, each instance has two sets of features. The first feature set includes the conventional features such as node type, sizing, and area. We use this set of features in the GNN operation and name it graph node features. The other set is the text embeddings of instance names and instance types. We pre-train a word embedding

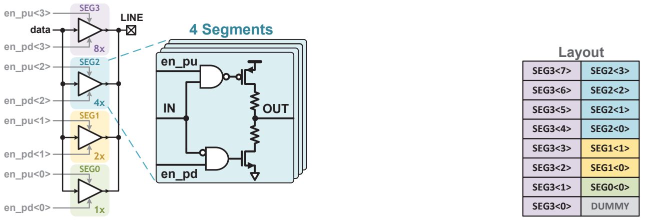
**Table 1: List of Instance Parameters.**

| Type | Feature | Definition |
|---|---|---|
| Transistors | L | Gate poly length |
| | NF | Number of fingers |
| | NFIN | Number of fins |
| Resistors | L | Length of resistors |
| | W | Width of resistors |
| Capacitors | M | Multipliers |



(a) (b)

**Figure 4: An OTA design with symmetric structure. (a) The schematic. (b) Manual layout abstraction.**



(a) (b)

**Figure 5: A line driver design with array structure. (a) The schematic. (b) Manual layout abstraction.**

model and use the learned text embeddings to provide additional information to the graph node features.

*3.2.1 Device Parameter Features.* We initialize the graph's first feature set for device and sub-circuit nodes. The first part of the node feature is a one-hot vector of node types. In this work, the node types include regular NMOS, regular PMOS, thick gate NMOS, thick gate PMOS, resistor, capacitor, and sub-circuit. The second part is an instance's width, height, and area. For sub-circuits, we sum up the area of children instances to approximate the sub-circuit areas. The widths and heights are then calculated, assuming the aspect ratio is 1. The third part of the feature is the sizing parameters. They define the device geometries and influence the circuit functionalities. Table 1 lists the parameters included. All the parameters are normalized. We average their children's instances to obtain the sizing feature fields for sub-circuit nodes.

*3.2.2 Texts Features.* In addition to the graph node feature, we also incorporate the word embedding of the instance name and device/sub-circuit type name in our framework.

Circuit netlists describe instances using an associated instance name and a device type, where the use of this information has been usually overlooked so far. In typical AMS circuit netlists, an instance is associated with an instance name from the circuit netlist and an instance type.

The instance names empirically incorporate the purpose and the position of the instances. Intuitively, the designers select the names to help them understand the circuit design, e.g., `NMOS0`, `INV0`, and `NDIFF`. Although not always deliberately planned, the name of an instance usually provides prior knowledge from design expertise. We observe in real-world designs that there is a correlation between the naming similarity and placement proximity. The instance names can be utilized as supplementary knowledge in circuit representation learning.

We also find that the instance names are beneficial for offsetting some of the limitations of ConvGNNs. The instance names help distinguish locally isomorphic structures while retaining an inductive basis across different circuits. Figure 4 shows an example of an operational transconductance amplifier (OTA). Its schematic has a highly symmetric structure. As a result, typical ConvGNNs usually are challenged in distinguishing the `A` branch nodes from the `B` branch in the embedding space, such as `M4A` and `M4B`. On the other hand, by examining the instance names, a human can quickly identify the circuit structures. Because those names contain prior knowledge of pair-wise symmetry relations and instance positions in the circuit, the naming convention plays an even more critical role in mixed-signal designs where parallel structures are common. Figure 5 shows a line driver design. The driver consists of unit-sized driver segments configured for binary-weighted digital control of drive strength. The only difference between the segments is the connection to different control signals (e.g., `en_pu<3:0>` and `en_pd<3:0>`), which are also coming from control circuits with similar structures. Such differences are challenging for ConvGNNs to learn, while humans can easily understand by observing the instance naming.

Device/sub-circuit type names are also often neglected in existing AMS circuit learning schemes. A common approach (e.g., [21]) is to group the devices into several groups, such as PMOS, NMOS, and capacitors. However, such a paradigm omits the detailed difference between device types, such as low and high threshold voltage transistors. On the other hand, the work [22] separates every device type with a one-hot encoding scheme. However, it adds additional complexity for the network to learn the behavior of every single device. Besides, there are few considerations of sub-circuit names in circuit learning models. Although circuit type identification techniques exist [5], a principal method to vectorize sub-circuit type for learning tasks is yet to be explored. On the other hand, the device types and circuit types are usually well described in their naming for circuit designers to understand. Similar device types also have parasitics that are correlated.

In TAG, we consider the texts in the circuit representation learning tasks. In [23], names of module hierarchies are encoded with trie (suffix graph) to assist placement. The modules with common ancestors in the hierarchy tree have similar hierarchical encoding. The similarity of hierarchies benefits the ML-assisted placement to find better clustering of the modules. However, the trie-based encoding method does not leverage the semantic information of the texts and is hard to be extended to new designs. A new paradigm is proposed to vectorize instances and type names using word embeddings.

We adopt the fastText framework [24] to vectorize the texts. It extracts the subword information to enable learning from words having similar subwords (e.g., `nch_ulvt_mac` and `nch_lvt_mac`). It uses a hashing function to store the dictionary that allows producing word embedding for unseen words. We extract the sentences from the netlists of 1490 industrial AMS circuit designs. The following words are treated as one sentence: (1) the current circuit name

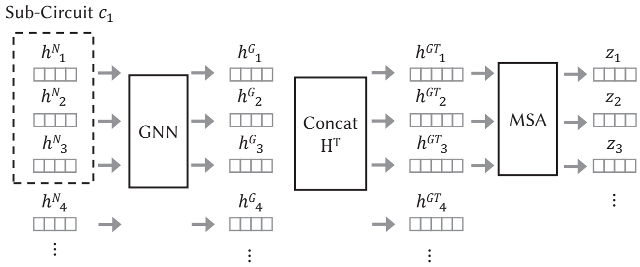


**Figure 6: Illustration of the proposed instance representation embedding network.**

**Algorithm 1** Instance Embedding Algorithm in TAG

```
Input: A heterogeneous hierarchical graph representation $G = (V, E)$, node features $\mathbf{H}^N$, text embedding features $\mathbf{H}^T, \forall i \in V$ and a set of sub-circuits $C$.
Output: The instance embeddings $\mathbf{Z}$.
1: GNN forward operation $\mathbf{H}^G = \text{GNN}(G, \mathbf{H}^N)$.
2: Concatenate with text embeddings $\mathbf{H}^{GT} = \text{concat}(\mathbf{H}^G, \mathbf{H}^T)$
3: Transform to embedding dimension $\mathbf{H}^{GT} = \mathbf{W}\mathbf{H}^{GT}$
4: Initialize an empty matrix $\mathbf{Z} = \text{zeros}(|V|, d)$
5: for each Sub-circuit $C_i \in C$ do
6:     Extract the instances embeddings $\mathbf{H}_{C_i} = \{\mathbf{H}^{GT}_k, \forall k \in C_i\}$
7:     MSA forward operation $\mathbf{Z}_{C_i} = \text{MSA}(\mathbf{H}_{C_i})$
   return Z
```

and the device/sub-circuit type names of its children instances, and (2) the instance name, its device/sub-circuit type names, and the net names connecting to this instance. We then combine the extracted sentences with the first 1 billion bytes of English Wikipedia corpus [25]. We train the word embedding model using the fastText framework with a word embedding dimension of 64, a context window size of 10, and a maximum length of character N-gram of 15.

## 3.3 Instance Representation Embedding Network

The embedding network in TAG contains two stages: the GNN and the multi-head MSA stage. Algorithm 1 sketches the procedures. We first compute the graph embeddings $\mathbf{H}_G$ using a GNN model (Line 1). Then we concatenate the graph embedding $\mathbf{H}^G$ with the instance text embedding $\mathbf{H}^T$ and apply linear transform on it to form a combined embedding $\mathbf{H}^{GT}$ (Line 2). We then treat the combined embeddings $\mathbf{H}^{GT}$ from the same sub-circuit as a collection (Line 6). This collection of embedding vectors is sent to an MSA layer (Line 7). The MSA layer enables all the instances in their sub-circuit to be considered together, adding a global context to the final instance embeddings. Figure 6 shows an illustration of the proposed TAG network. After the MSA layer, the TAG embedding vectors $\mathbf{Z}$ are then fed to the distance prediction network or the other downstream tasks.

The GNN network in TAG has two convolution layers. The first layer uses different linear transform matrices for different edge types to distinguish different edge connections. The convolution operation on this layer is shown in Equation (2)

$$\mathbf{h}_i^{l+1} = \text{ReLU}\left(\mathbf{W}^l_{self}\mathbf{h}^l_i + \text{mean}\left(\sum_{j \in \mathcal{N}_i} \mathbf{W}^l_{e_{ij}}\mathbf{h}^l_j\right)\right), \tag{2}$$

where $\mathbf{h}_i^{(l)}$ is the $l^{\text{th}}$ layer output for node $n_i$, $\mathcal{N}_i$ indicates the neighbors of node $n_i$, $\mathbf{W}^l_{self}$ is the weight for transform on the node $n_i$ itself and $\mathbf{W}^l_{e_{ij}}$ is the weight for edge type $e_{ij}$. The second layer is a graph isomorphism network (GIN) layer [26]. It is provably as powerful as the Weisfeiler-Lehman graph isomorphism test and is shown in Equation (3).

$$\mathbf{h}_i^{(l+1)} = \mathbf{W}\left((1+\epsilon)\mathbf{h}^l_i + \sum_{j \in \mathcal{N}_i}\left\{\mathbf{h}^l_j\right\}\right), \tag{3}$$

where $\mathbf{W}$ is a weight matrix, and we use $\epsilon = 0$ in the experiments. In our implementation, we set the hidden layer dimension and the output embedding dimension in the GNN to be 64 and 32, respectively. We also experiment with the variants of GNN with popular ConvGNN layers and change the number of layers. The impact of GNN architecture choice is relatively minor compared to our other proposed techniques.

The graph embedding and pre-trained text embedding of the same node are concatenated together and sent to an MSA layer. We embed instances within a sub-circuit as an unordered sequence, on which we apply the self-attention mechanism. Self-attention (SA) [27] is a popular building block for machine learning on sequences. Equation (4) shows its computation equations,

$$\begin{aligned} [\mathbf{q}, \mathbf{k}, \mathbf{v}] &= \mathbf{z}\mathbf{U}, & \mathbf{U} &\in \mathbb{R}^{D \times 3D_h}, \\ A &= \text{softmax}(qk^T/\sqrt{D_h}), & A &\in \mathbb{R}^{N \times N}, \\ \text{SA}(\mathbf{z}) &= A\mathbf{v}, \end{aligned} \tag{4}$$

where $q$, $k$, and $v$ are query, key, and value matrices of each embedding, $N$ is the sequence length, $A$ is the attention of each query-key pair, and $\text{SA}(\mathbf{z})$ is the final embedding of each node based on the attention over the value matrices of other nodes. The SA mechanism can be applied to an arbitrary input sequence length. MSA extends the SA mechanism to run $k$ SA operations, called "head", in parallel. The MSA operation is shown in Equation (5).

$$\text{MSA}(\mathbf{z}) = [\text{SA}_1(\mathbf{z}); \text{SA}_2(\mathbf{z}); ...; \text{SA}_k(\mathbf{z})]\mathbf{U}, \quad \mathbf{U} \in \mathbb{R}^{k \cdot D_h \times D}. \tag{5}$$

We use $k = 4$, $D = 64$, and $D_h = 16$ in the experiments.

## 3.4 Layout Instance Distance Prediction Loss

After obtaining the embeddings, we iterate through all instance pairs in the same sub-circuit and predict their relative distances in the manual layout implementation. This learning task allows the embedding network to extract the knowledge from human layout implementation without additional manual labeling.

An ad-hoc approach to predict distance based on two embedding vectors is to concatenate them and feed-forward using a fully connected network (FC) as shown in Equation (6).

$$y_{ij} = \text{FC}([\mathbf{z}_i; \mathbf{z}_j]) \tag{6}$$

We also propose a more direct approach for predicting the relative distance in TAG. We assume an instance embedding vector space exists where the distance in this space is proportional to the

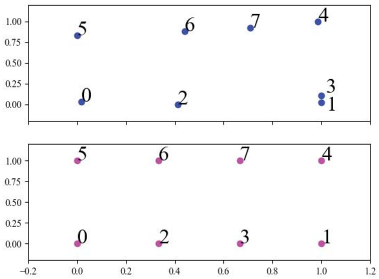


**Figure 7: Example of the spatial embeddings and their corresponding layout locations. Above: The two dimensional principal component analysis of the embeddings. Bottom: The layout locations.**

expected placement distance in manual layout. Equation (7) shows our proposed method.

$$\mathrm{NORM}(i, j, C, \mathbf{H}) = \frac{||\mathbf{h}_i - \mathbf{h}_j||}{\max_{k,l}(||\mathbf{h}_k - \mathbf{h}_l||)}, \quad \forall k, l \in C, k \neq l, \qquad (7)$$

where $\mathbf{H}$ denotes the embedding space, $\mathbf{h}_i$ indicates the embedding of instance $I_i$ and $C$ is a collection of instances, which is the sub-circuits in our scenario. The denominator term finds the maximum distance between the instance pair in this sub-circuit, approximating the sub-circuit diameters. Intuitively, we measure the relative layout distance of two instances by computing their distance normalized by the sub-circuit diameters. Figure 7 shows an illustration of our learned $\mathbf{H}$ and the corresponding instance relative positions in a sub-circuit. In the implementation, we use the $\mathrm{LogSumExp}(x_i, \ldots, x_n) = \log(\exp(x_i) + \cdots + \exp(x_n))$ function to smooth and approximate the $\max(\cdot)$ function for more robust and efficient training. TAG adopts the scheme and adds a layer normalization step and an FC network before the distance norm computation, as shown in Equation (8).

$$\mathrm{DIST}(i, j, C, \mathbf{Z}) = \mathrm{NORM}(i, j, C, \mathrm{FC}(\mathrm{LayerNorm}(\mathbf{Z}))), \qquad (8)$$

where $Z$ is the embedding after MSA layer. We choose to add an additional fully connected (FC) layer before the NORM layer because we empirically find doing so leads to better transferability to the downstream tasks. $L_2$ norm and FC networks with 1 hidden layer of dimension 128 in our experiments are used. The mean squared error loss is used to train the model.

# 4 EXPERIMENTAL RESULTS

We implement the framework in Python with the PyTorch library. All models are trained on a single NVIDIA Tesla V100 GPU with 32GB memory. All models are trained with an ADAM optimizer.

The proposed method is evaluated on a dataset of 447 industrial AMS circuits in sub-10nm technology. The size of the circuits ranges from 20 to 2000 instances. We exclude the sub-circuits under four instances to avoid the results being dominated by naïve cases and sample 20 instances from one sub-circuit for large sub-circuits. To extract the placement coordinates of each device in the layout view, we use the StarRC extraction tool.

The circuits in the dataset are randomly shuffled and split into training, validation, and test sets with 60%, 20%, and 20% allocation, respectively. Because different circuits sometimes share common sub-circuits, to avoid data leakage, we exclude all the sub-circuits that appear in the training set when doing the validation and testing.

**Table 2: Comparisons of $R^2$, MAE and MAPE for directly predicting instance relative distance with text embedding distance norm.**

| Method | $R^2$ | MAE | sMAPE |
|---|---|---|---|
| NLP Pre-trained Model [28] | -0.783 | 0.291 | 0.565 |
| Our model | **0.205** | **0.186** | **0.452** |

We report the test set results in the experiments at the epoch with the lowest validation loss.

The training time takes about 10 hours on the dataset. The inference time for each circuit takes an average of 0.09 seconds and a max of 0.8 seconds. Most of the inference time is spent reading the files instead of model inference. As the training is a one-time job, the runtime for TAG is considered negligible in usual applications.

To evaluate the solution quality, we adopt two sets of statistical measurements. For regression tasks, we use R-squared ($R^2$), Mean Absolute Error (MAE), and symmetric Mean Absolute Percentage Error (sMAPE) as the metrics. For binary classification tasks, we adopt accuracy (ACC), true positive rate (TPR), false positive rate (FPR), positive predictive value (PPV), and $F_1$-score over the valid pairs. Higher $R^2$, ACC, TPR, PPV, and $F_1$ scores are better, while lower MAE, sMAPE, and FPR scores are better.

To evaluate the effectiveness of our learned circuit representation in applications, we obtained the source codes of the AncstrGNN [3] and Paragraph [21] from the authors. We train these frameworks on our dataset and compare our circuit representation in the case studies.

In the rest of this section, we evaluate our text embedding quality and the circuit representation learning scheme and conduct two case studies for using our model in other two learning tasks: detecting matching in layouts and predicting the wirelength.

## 4.1 Circuit Text Embedding

We first evaluate the quality of the word embedding model. The word embedding ideally shall provide a meaningful similarity measurement between the instances. We verify this property by directly applying the distance norm method (Equation (7)) on the word embedding to predict the relative placement distances. As shown in Table 2, our model yields an $R^2$ of 0.205. This result is already better than the vanilla GNN-only approach, even without layout data or additional trainable parameters. It shows the proposed text embedding contains valuable information for learning. In comparison, the same model pre-trained with natural language corpus (Common Crawl) alone can only result in a $R^2$ of -0.783. The improvement of our word embedding model shows the benefits of training the word embedding model with sentences extracted from netlists.

To investigate the meaning of the embeddings, we visualize the high-dimensional embedding vector with the t-SNE algorithm [29]. Figure 8 shows the t-SNE plots for the words in 30 circuits. It is observed that two PMOS (`pch_ulvt_mac` and `pch_lvt_mac`) device types are close and well separated from the NMOS device type (`nch_ulvt_mac`). The t-SNE plot illustration shows that our word embedding model captures the similarities in texts and provides a new dimension of information and the conventional graph representation.

Within our benchmark circuits, many instance names are, in fact, not explicitly named, e.g., `M_I1` and `XI0`. However, from our experience, the important instances are usually well named.

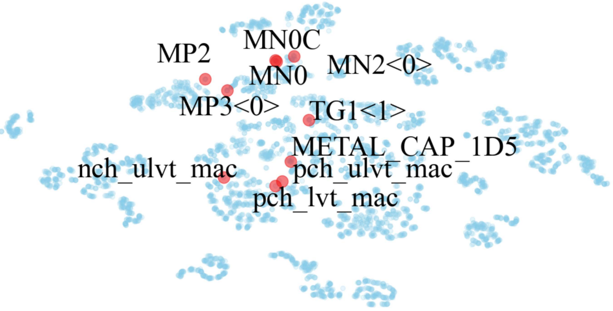


**Figure 8: t-SNE plots of embeddings of the word embeddings.**

Our experimental results demonstrate the overall effectiveness of the text embeddings even with the existence of arbitrary naming conventions in some netlists.

## 4.2 Instance Relative Distance Prediction

We evaluate our model accuracy on instance relative distance prediction. We predict the distances between all the instance pairs within the same sub-circuit. The distance is normalized to the sub-circuit placement bounding box. Table 3 shows the regression results compared with different variants of the models and training methods. The model types are denoted before the dash in the method names. “T”, “A” and “G” indicate the model contains text embedding, self-attention layers, and a graph neural network, respectively. The prediction method is labeled after the dash. “CAT” denotes concatenating the instance pair embeddings and predicting the distance with an FC network (Equation (6)). “NORM” indicates to use our proposed distance measuring method (Equation (8)). For example, TG-CAT denotes the model using GNN and text embedding and predicts the distance with the concatenation method. T-NORM denotes using only text embedding without graph with distance norm method.

Our proposed TAG framework outperforms the other methods in all metrics and achieves $R^2$ of 0.64. Meanwhile, the conventional GNN-only structure can hardly produce meaningful predictions above 0 $R^2$. Because layout instance distances are intrinsically noisy due to the manual implementation, our proposed model demonstrates a strong capability to learn the circuit representation. From the ablation study with different model variants, we also observe that each component of our proposed framework benefits the learning task. These observations demonstrate the effectiveness of our proposed techniques.

We also compare with the AncstrGNN [3]. “AncstrGNN-A” denotes using the AncstrGNN. AncstrGNN uses a Gated-GNN layer to generate node embeddings based on a contrastive loss between nodes. We adopt the "CAT" approach for relative distance prediction to predict pair-wise distance from pair embeddings. The sub-circuit embeddings are obtained by mean aggregating their children’s embeddings. “AncstrGNN-B”, on the other hand, uses the AncstrGNN network architecture but with our proposed hierarchical graph representation. In both cases, AncstrGNN can not learn the relative distance effectively. The results are similar to our G-only model.

Note that the “G-NORM” and “AG-NORM” options both fail in training. The reason is rooted in the graph formulation and the ConvGNN mechanism. As discussed in Section 2, the local isomorphic structure will make nodes indistinguishable. As a result, the distance between two node embeddings might be close or equal to zero. When we use the distance norm method, the backward gradient in such a case will be very large and cause the training process to diverge. This observation also shows the importance of text embedding.

**Table 3: Comparisons of $R^2$, MAE and sMAPE for instance relative distance prediction.**

| Method | $R^2$ | MAE | sMAPE |
|---|---|---|---|
| AncstrGNN [3]-A | -0.091 | 0.225 | 0.508 |
| AncstrGNN [3]-B | 0.068 | 0.191 | 0.502 |
| G-CAT | 0.075 | 0.134 | 0.502 |
| T-CAT | 0.194 | 0.187 | 0.489 |
| TA-CAT | 0.452 | 0.164 | 0.453 |
| TG-CAT | 0.335 | 0.177 | 0.542 |
| AG-CAT | 0.367 | 0.184 | 0.508 |
| TAG-CAT | 0.585 | 0.134 | 0.404 |
| G-NORM | FAIL | FAIL | FAIL |
| T-NORM | 0.321 | 0.177 | 0.458 |
| TA-NORM | 0.530 | 0.140 | 0.409 |
| TG-NORM | 0.470 | 0.154 | 0.442 |
| AG-NORM | FAIL | FAIL | FAIL |
| TAG-NORM | **0.640** | **0.122** | **0.364** |

## 4.3 Application Case Study 1: Layout Matching Prediction

The effectiveness of our learned circuit representation is evaluated for predicting the matching patterns in the layout. We use the same dataset in the pre-training process. Instructed by the designers, the labels of matched instances are extracted based on layout coordinates. The matching conditions include interdigitation pattern, common-centroid pattern, and symmetry pattern. The methods are evaluated to detect those matching pairs. This task is similar to the symmetry constraint detection problem.

We randomly selected 10% of the entire dataset from the training set to train the models. Another 10% of circuits from the validation set in the previous stage are used to validate the task. The entire test set (20% of circuits) is used for testing. The task is treated as a binary classification task by concatenating two instance embeddings and forwarding with a two-layer FC network and is trained with the cross-entropy loss.

We compare the proposed pre-trained circuit representation with training from scratch and the state-of-the-art symmetry detection framework AncstrGNN [3]. Table 4 shows comparisons of evaluation metrics. “AncstrGNN-A-CAT” concatenates the pre-trained AncstrGNN embedding and trains an FC network to do binary classification. “AncstrGNN-A-COS” uses the cosine similarity criteria to predict the symmetry constraint as proposed in the original paper. “AncstrGNN-B-CAT” and “AncstrGNN-B-COS”, on the other hand, are using our proposed hierarchical graph representation. “G trained from scratch” and ‘TAG trained from scratch” train the network without pre-trained weights. “TAG” is with our proposed pre-trained embeddings. The embedding network weights in this configuration are fixed in the training so that its results measure the generality of our pre-trained embeddings. Our proposed method outperforms training from scratch and AncstrGNN with an $F_1$ score of 0.842. This observation shows the effectiveness of applying the proposed pre-trained circuit representation in other tasks.

**Table 4: Comparisons of ACC, TPR, FPR, PPV and $F_1$ for layout matching prediction.**

| Method | ACC | TPR | FPR | PPV | $F_1$ |
|---|---|---|---|---|---|
| AncstrGNN [3]-A-CAT | 0.677 | 0.802 | 0.434 | 0.621 | 0.706 |
| AncstrGNN [3]-A-COS | 0.805 | 0.724 | **0.086** | **0.919** | 0.810 |
| AncstrGNN [3]-B-CAT | 0.750 | 0.701 | 0.203 | 0.765 | 0.731 |
| AncstrGNN [3]-B-COS | 0.720 | 0.740 | 0.305 | 0.738 | 0.739 |
| G Trained from scratch | 0.731 | 0.706 | 0.246 | 0.730 | 0.718 |
| TAG Trained from scratch | 0.730 | 0.666 | 0.208 | 0.751 | 0.706 |
| TAG Pre-trained | **0.833** | **0.915** | 0.244 | 0.780 | **0.842** |

**Table 5: Comparisons of $R^2$, MAE and sMAPE for relative HPWL prediction.**

| Method | $R^2$ | MAE | sMAPE |
|---|---|---|---|
| AncstrGNN [3]-A | -0.177 | 0.146 | 0.550 |
| AncstrGNN [3]-B | 0.203 | 0.198 | 0.520 |
| G Trained from scratch | 0.027 | 0.219 | 0.566 |
| TAG Trained from scratch | 0.153 | 0.212 | 0.543 |
| TAG Pre-trained | **0.570** | **0.139** | **0.469** |

## 4.4 Application Case Study 2: Wirelength Estimation

Another case study is to estimate the net HPWL. We also use the same dataset in the pre-training process. Like the instance distance prediction task, we normalize HPWL with respect to the sub-circuit layout bounding box to allow inductive learning. The 10%/10%/20% data splitting is used for training, validation, and test sets similar to the matching prediction task.

We use a self-attention layer with four heads and mean aggregation on the instance embeddings and predict the HPWL using a two-layer FC network, as shown in Equation (9).

$$WL = \mathrm{FC}\left(\mathrm{mean}\left(\mathrm{MSA}\left(\mathbf{Z}_N\right)\right)\right), \tag{9}$$

where $\mathbf{Z}_N$ denotes a collection of instance embeddings connected by net $N$.

Table 5 shows the comparisons of HPWL prediction results. “AncstrGNN-A” denotes using the AncstrGNN embedding with the original flatten graph, while “AncstrGNN-B” uses the TAG configurations. The pre-trained TAG model achieves the best result in all evaluation metrics. The observation in this case study aligns with the results from the matching prediction task that TAG outperforms the baselines. It is observed that there is a performance gap between “AncstrGNN-A” and “AncstrGNN-B”. We believe that adding hierarchy knowledge with our proposed graph representation benefits the learning task.

## 4.5 Application Case Study 3: Net Parasitic Capacitance Prediction

We also evaluate the effectiveness of our pre-trained embeddings in the net parasitic capacitance prediction task. This task uses a dataset of 385 industrial AMS circuits in sub-10nm technology. Based on the recommendation from the designers, we use 17 designs in the dataset as the testing set and the rest as the training set. We verify that there is no overlap between this testing set and the training set in the model pre-training.

The TAG embeddings are integrated with the state-of-the-art parasitics prediction algorithm, ParaGraph [21]. The TAG embedding vectors for all the instances are first generated with a pre-trained TAG model. Then we augment the ParaGraph input features with these TAG embeddings. Five models are trained at one time with different maximum prediction values ($max_v$) of 0.5fF, 1fF, 10fF, 100fF and 1pF. The final prediction is obtained with the ensemble modeling technique as suggested in the original paper.

**Table 6: Comparisons of $R^2$ and MAE of different $max_v$ for the net parasitic capacitance prediction task.**

| Metrics | $max_v$ | ParaGraph [21] | TAG |
|---|---|---|---|
| $R^2$ | 0.5fF | 0.495 | **0.678** |
| | 1fF | 0.830 | **0.876** |
| | 10fF | 0.854 | **0.856** |
| | 100fF | **0.872** | 0.870 |
| | 1pF | 0.308 | **0.411** |
| MAE | 0.5fF | $2.71e-17$ | $\mathbf{2.25e-17}$ |
| | 1fF | $5.14e-17$ | $\mathbf{3.41e-17}$ |
| | 10fF | $2.01e-16$ | $\mathbf{1.83e-16}$ |
| | 100fF | $\mathbf{3.23e-16}$ | $3.39e-16$ |
| | 1pF | $9.93e-16$ | $\mathbf{9.07e-16}$ |

**Table 7: Comparisons of mean and geometric mean of the errors in simulated performance with predicted net parasitic capacitance.**

| Method | Mean | Geometric Mean |
|---|---|---|
| ParaGraph [21] | 18.6% | 4.97% |
| TAG | **11.8%** | **4.17%** |

Table 6 shows the comparisons of the prediction accuracy. With the augmented TAG embedding, the net parasitic capacitance prediction achieves significant improvement in accuracy for the 0.5fF and 1pF models. It also produces similar accuracy for the 1fF, 10fF, and 100fF models. We believe that it is because the pre-trained TAG embedding incorporates valuable spatial information of the circuits. The additional spatial information allows the model to make more accurate predictions considering layout effects. Table 7 shows the corresponding errors on simulated performance. With the more accurate net capacitance predictions, the TAG embedding helps to reduce the mean performance error from 18.6% to 11.8%. The results demonstrate the effectiveness of our proposed TAG model.

## 5 CONCLUSION

This paper has presented TAG, a new paradigm and framework to learn and pre-train circuit instance representations. Using relative layout distance for the training target, TAG embeds the high-level knowledge into the representation by fitting the spatial information. It leverages the netlists and introduces sub-circuit-wise MSA to assist the training. A comprehensive algorithm set has been presented, including feature extraction, network architecture, and learning algorithm. Experimental results have demonstrated the efficiency and effectiveness of TAG on learning spatial knowledge and the ability to transfer the learned embeddings to other learning tasks.

## ACKNOWLEDGEMENT

This work is supported in part by NVIDIA Corporation, the DARPA IDEA program, and the NSF under Grant No. 1704758 and 2112665. The authors would like to thank Xuyang Jin from The University of Texas at Austin and Yanqing Zhang from NVIDIA Corporation for helpful comments and discussions.

## REFERENCES

[1] H. Chen, M. Liu, X. Tang, K. Zhu, N. Sun, and D. Z. Pan, "Challenges and opportunities toward fully automated analog layout design," *Journal of Semiconductors*, vol. 41, no. 20070021, p. 111407, 2020.

[2] E. Afacan, N. Lourenço, R. Martins, and G. Dündar, "Review: Machine learning techniques in analog/rf integrated circuit design, synthesis, layout, and test," *Integration, the VLSI Journal*, vol. 77, pp. 113–130, 2021.

[3] H. Chen, K. Zhu, M. Liu, X. Tang, N. Sun, and D. Z. Pan, "Universal symmetry constraint extraction for analog and mixed-signal circuits with graph neural networks," in *Proc. DAC*, 2021.

[4] X. Gao, C. Deng, M. Liu, Z. Zhang, D. Z. Pan, and Y. Lin, "Layout symmetry annotation for analog circuits with graph neural networks," in *Proc. ASPDAC*, 2021.

[5] K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. Sharma, W. Xu, S. M. Burns, J. Hu, R. Harjani, and S. S. Sapatnekar, "GANA: Graph convolutional network based automated netlist annotation for analog circuits," in *Proc. DATE*, 2020.

[6] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolić, "AutoCkt: Deep reinforcement learning of analog circuit designs," in *Proc. DATE*, 2020.

[7] H. Wang, K. Wang, J. Yang, L. Shen, N. Sun, H.-S. Lee, and S. Han, "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. DAC*, 2020.

[8] Z. Wang, C. Bai, Z. He, G. Zhang, X. Qiang, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proc. DAC*, 2022.

[9] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proc. DAC*, 2022.

[10] N. Wu, H. Yang, Y. Xie, P. Li, and C. Hao, "High-level synthesis performance prediction using gnns: Benchmarking, modeling, and advancing," in *Proc. DAC*, 2022.

[11] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "Code2vec: Learning distributed representations of code," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, jan 2019.

[12] V. J. Hellendoorn, P. Maniatis, R. Singh, C. Sutton, and D. Bieber, "Global relational models of source code," in *Proc. ICLR*, 2020.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2019.

[14] F. Wu, A. Souza, T. Zhang, C. Fifty, T. Yu, and K. Weinberger, "Simplifying graph convolutional networks," in *Proc. ICML*, 2019.

[15] K. Zhu, H. Chen, M. Liu, and D. Z. Pan, "Automating analog constraint extraction: From heuristics to learning," in *Proc. ASPDAC*, 2022.

[16] K. Kunal, P. Poojary, T. Dhar, M. Madhusudan, R. Harjani, and S. Sapatnekar, "A general approach for identifying hierarchical symmetry constraints for analog circuit layout," in *Proc. ICCAD*, 2020.

[17] A. Kahng and S. Reda, "Intrinsic shortest path length: a new, accurate a priori wirelength estimator," in *Proc. ICCAD*, 2005.

[18] D. Hyun, Y. Fan, and Y. Shin, "Accurate wirelength prediction for placement-aware synthesis through machine learning," in *Proc. DATE*, 2019.

[19] M. Liu, W. Li, K. Zhu, B. Xu, Y. Lin, L. Shen, X. Tang, N. Sun, and D. Z. Pan, "S$^3$DET: Detecting system symmetry constraints for analog circuits with graph similarity," in *Proc. ASPDAC*, 2020.

[20] B. Shook, P. Bhansali, C. Kashyap, C. Amin, and S. Joshi, "Mlparest: Machine learning based parasitic estimation for custom circuit design," in *Proc. DAC*, 2020.

[21] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proc. DAC*, 2020.

[22] H. Chen, M. Liu, X. Tang, K. Zhu, A. Mukherjee, N. Sun, and D. Z. Pan, "MAGICAL 1.0: An open-source fully-automated ams layout synthesis framework verified with a 40-nm 1 GS/s $\Delta\Sigma$ ADC," in *Proc. CICC*, 2021.

[23] Y.-C. Lu, S. Pentapati, and S. K. Lim, "The law of attraction: Affinity-aware placement optimization using graph neural networks," in *Proc. ISPD*, 2021.

[24] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *arXiv preprint arXiv:1607.04606*, 2016.

[25] M. Mahoney, "Wikipedia corpus." [Online]. Available: http://mattmahoney.net/dc/enwik9.zip

[26] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *Proc. ICLR*, 2019.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NIPS*, 2017.

[28] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC)*, 2018.

[29] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.