# GNN-based Hierarchical Annotation for Analog Circuits

Kishor Kunal, Tonmoy Dhar, Meghna Madhusudan, Jitesh Poojary, Arvind K. Sharma, Wenbin Xu, Steven M. Burns, Jiang Hu, Ramesh Harjani, Sachin S. Sapatnekar

*Abstract*—**Analog designs consist of multiple hierarchical functional blocks. Each block can be built using one of several design topologies, where the choice of topology is based on circuit performance requirements. A major challenge in automating analog design is in the identification of these functional blocks, which enables the creation of hierarchical netlist representations. This can facilitate a variety of design automation tasks such as circuit layout optimization because the layout is dictated by constraints at each level, such as symmetry requirements, that depend on the topology of the hierarchical block. Traditional graph-based methods find it hard to automatically identify the large number of structural variants of each block. To overcome this limitation, this paper leverages recent advances in graph neural networks (GNNs). A variety of GNN strategies is used to identify netlist elements for circuit functional blocks at higher levels of the design hierarchy, where numerous design variants are possible. At lower levels of hierarchy, where the degrees of freedom in circuit topology are limited, structures are identified using graph-based algorithms. The proposed hierarchical recognition scheme enables the identification of layout constraints such as symmetry and matching, which enable high-quality hierarchical layouts. This method is scalable across a wide range of analog designs. An experimental evaluation shows a high degree of accuracy over a wide range of analog designs, identifying functional blocks such as low-noise amplifiers, operational transconductance amplifiers, mixers, oscillators, and band-pass filters in larger circuits.**

## I. Introduction

Despite many new developments over the years in the arena of design automation for analog circuit layout [1], [2], automation has found a low level of acceptance within the analog design community. The traditional model for layout design involves a human expert designer who translates a circuit from netlist to layout. This designer-driven approach has survived primarily because of the large number of layout styles as well as implicit and explicit rules that must be captured by the layout. Optimal techniques are internalized by the designer based on many years of experience and have been difficult for automated tools to replicate. While digital designs use a standardized set of lowest-level cells from a standard cell library and use Boolean algebra to guide transformations from one circuit topology to another, analog designs see large variations at even the lowest level across designs that implement the same functionality. For example, between textbooks [3] and research papers, there are well over 100 widely used operational transconductance amplifier (OTA) topologies of various types (e.g., telescopic, folded cascode, Miller-compensated). Moreover, analog designers are culturally likely to exercise such variations as this "secret sauce" may provide tangible improvements in performance.

A prerequisite for truly generalizable analog automation is to recognize all variants – including those that have not even been designed to date. Prior methods are *library-based* [4], [5], matching a circuit to prespecified templates, and requiring an enumeration of possible topologies in an exhaustive database, or *knowledge-based* [6]–[8], embedding rules for recognizing circuits; however, the rules must come from an expert designer who may struggle to provide a list (many rules are intuitively ingrained rather than explicitly stated). Moreover, it is difficult to capture rules for all variants of every circuit block.

Both classes of methods involve prohibitive costs in terms of designer input, and cannot be easily adapted to new topology variants. In contrast, an experienced expert designer can examine a large circuit and recognize substructures of known blocks; minor variations do not affect this judgment. For instance, an OTA circuit consists of a bias network, a current mirror, a differential pair, a differential load, and an output stage: one look at a schematic is enough for an expert to decipher these stages. Conventional algorithms cannot replicate this. Thus, in spite of the large body of publications on analog layout synthesis, these methods are seldom used in industry.

We seek to build a foundation for a generalizable analog design methodology with no human in loop. At the core of such a methodology is the ability to recognize blocks in a circuit, at the level of an expert human designer. We draw inspiration from machine learning (ML) methods that are adept at recognizing variants of a target and have been widely used for recognizing images. Unlike images, a netlist cannot be abstracted into a planar graph with a simple 2D topological embedding, and convolutional neural networks (CNNs) cannot be used directly.

Our approach inputs a transistor-level netlist, aggregates elements that implement a set of functionalities, and outputs a circuit hierarchy. We abstract the circuit netlist as a graph and leverage recent advances in the field of graph neural networks (GNNs) [9]–[11] to perform approximate subgraph isomorphism, classifying circuit elements into classes, depending on which subcircuit they belong to. Once smaller subcircuits are identified, we use graph-based approaches on their subgraphs and identify basic structures ("primitives") that compose these blocks. Primitives (e.g., differential pairs, current mirrors) contain a small number of transistors/passives and do not

K. Kunal, T. Dhar, M. Madhusudan, J. Poojary, A. K. Sharma, R. Harjani, and S. S. Sapatnekar are with the Department of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN; W. Xu and J. Hu are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX; S. M. Burns, is with Intel Labs, Hillsboro, OR. This research was supported in part by the DARPA IDEA program under SPAWAR contract N660011824048.

vary significantly across circuits and can be handled using graph-based methods. In contrast, larger blocks (OTAs, low noise amplifiers (LNAs), mixers, etc.) may appear in a number of variants, and can benefit from using trained GNNs, which can handle variants naturally. A GNN learns the rules during training, thus mimicking the human expert.

This paper applies a variety of available GNN methodologies for the task of identifying commonly encountered sub-blocks in an analog design. We use supervised learning to train the GNN network on a set of available circuit topologies and then use the trained network to identify similar (but not necessarily identical) structures in new unseen circuits using node classification. This manuscript improves upon a preliminary version of this work [12] in several ways. The differences with our preliminary work are listed in Table I. First, it uses a wider range of GNNs for feature aggregation, and shows a comparative analysis of the performance of various GNNs for the task of circuit identification. Second, the graph representation of the circuit is modified from the bipartite graph in [12] to a bipartite multigraph. This provides a better representation of a circuit, where a net can connect to multiple terminals of devices: we demonstrate a substantial gain in accuracy of the GNN models using this new abstraction. This helps us in reducing the class-based postprocessing thus making it more generalizable to new classes of circuits.

**Table I:** A comparison of this work with our preliminary work.

| | **GANA** [12] | **This work** |
|---|---|---|
| Circuit Representation | Bipartite graph | Bipartite multigraph |
| Node-features | 18 | 18 (modified) |
| GNN method | ChebNet [10] | Multigraph GNN [13] |
| Training | Supervised | Supervised |
| GNN accuracy | Good | Better |
| Post-processing I | Graph-based | Graph-based |
| Post-processing II | Class-based | None |
| New designs | NA | True |

Our extracted circuit hierarchy can be used in various ways: for automated layout using hierarchical block assembly using the identified hierarchies; for automated constraint annotation (e.g., identifying symmetry, matching, common centroid, guard-ring around hierarchies, compact-placement); or constraint budgeting to each block while meeting system-level constraints.

While it is obvious that the extraction of hierarchies is useful for flat netlists, counterintuitively, it can even be useful for layout generation of netlists where hierarchies are specified by the schematic designer. This is because schematic designer's view of circuit hierarchy may not necessarily be optimal for layout hierarchy. For example, in OTA circuits, a schematic designer will typically keep the biasing circuit, which shares current paths with the differential pair, in a single hierarchy corresponding to the OTA signal paths. However, from a layout standpoint the current mirror transistor that lies in the signal path should be placed in a current mirror hierarchy, which enables the entire current mirror structure to be placed using a common-centroid or interdigitated methodology [14], [15]. Thus, creating a separate layout hierarchy for bias and signal paths provides better layout solutions. Another example corresponds to a public-domain successive approximation register (SAR) based ADC circuit design [16], where the netlist mixes the control logic with the capacitor array. However, a better layout solution is obtained by separating the two, using a separate hierarchy for the capacitor array and generating a common centroid layout for the capacitors. The objective of this work is to create hierarchies that enable optimal solutions for automated analog layout generation.

*Related work:* Early work on using ML for subcircuit identification includes a compiler-inspired learning approach in [17] that applies recursive compositions of prespecified rules to recognize structures from a netlist, but inherits the limitations of rule-based methods, and a CNN-based method [18] that translates a matrix embedding of a circuit to a set of templates. Both works operate on small structures with $< 50$ transistors, and face challenges in scaling to larger netlists; our method is demonstrated on blocks with $> 500$ transistors.

Motivated by the advances in ML techniques, several recent approaches have used ML methods to address problems in the domain of analog layout automation. The work in [19] solves the parasitic extraction method using graph neural networks. Research on extracting geometric constraints automatically has also evolved rapidly in recent years [20]–[22]. Approaches based on array recognition [23] recognize array structures and extract their symmetry constraints, without attempting to specifically target subcircuit recognition. Other methods work at the upper layers of the design hierarchy and extract system-level symmetries, leveraging statistical techniques [24]. The approaches in [25], [26] focus on the identification of symmetry pair constraints to match devices with similar net connections. However, methods for extracting geometric constraints cannot easily be generalized for subcircuit identification, e.g., the supervised learning method in [26] for clustering requires the number of clusters to be prespecified, but this information is unavailable during subcircuit identification.

## II. Hierarchical recognition

### A. Representing circuits hierarchically

Our recognition scheme creates a hierarchical representation of the netlist, identifying smaller structures composed into larger structures. At various levels of the hierarchical representation, we have

- *Elements*, i.e., transistors (NMOS/PMOS) and passives (resistors, capacitors, inductors), lie at the lowest level.
- *Primitives*, composed of a few elements form basic building blocks of a circuit. These are typically simple structures, largely invariant across circuits, e.g., differential pairs, current mirrors.
- *Sub-blocks* form multiple levels of the design hierarchy (i.e., some sub-blocks could be contained in others), and are composed of primitives or other sub-blocks, and contribute to building larger systems. Examples of sub-blocks include operational transconductance amplifiers (OTAs), analog-to-digital converters (ADCs), voltage-controlled oscillators (VCOs), equalizers, and clock/data recovery (CDR) circuits.

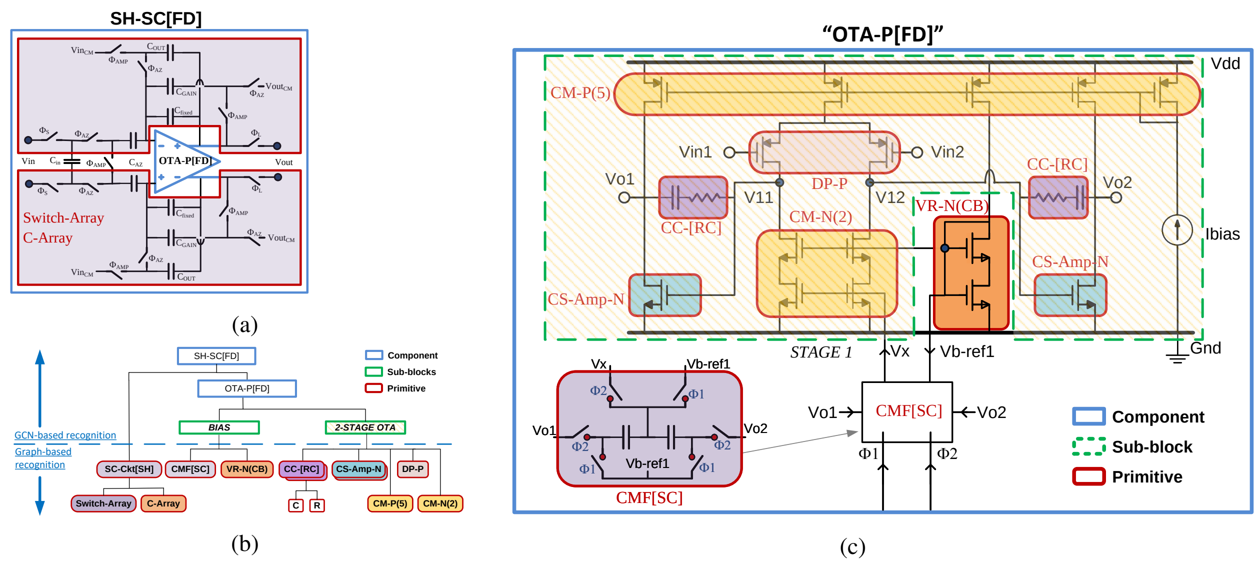

**Figure 1:** Multiple levels of abstraction in a sample-and-hold schematic.

Systems lie at the uppermost level of the hierarchy, and may correspond to structures such as RF transceivers, DC-DC converters, and a high-speed SerDes system. Our initial efforts reported in this paper goes up to the level of sub-blocks.

Fig. 1(a) decomposes a sample-and-hold circuit with a hierarchy tree (Fig. 1(b)) and at the schematic level (Fig. 1(c)):

- The uppermost level of hierarchy is the sample-and-hold switched-capacitor circuit block (SH-SC).
- Below this are the switched capacitor circuit (SC-Ckt) (the shaded blocks in Fig. 1(a)) and the OTA circuit.
- These are further decomposed into smaller structures (Fig. 1(c)).

Note that the decomposition of the OTA sub-block contains a telescopic-ota *STAGE 1* having primitives – the PMOS current mirror (CM-P(5)), the PMOS differential pair (DP-P), and the cascoded current mirror (CM-N(2)), and the bias circuit having primitives – the NMOS common-source amplifier (CS-Amp-N), common-mode feedback (CMF), the current reference (Ibias), and various passives (R, C, CC-RC) along with the bias circuitry.

Thus, our approach builds multiple levels of hierarchy with sub-blocks and primitives in the hierarchy tree. Our recognition algorithm processes primitives and sub-blocks hierarchically to come to the point at which an entire circuit of this type is recognized, and a hierarchy tree similar to Fig. 1(b) can be built.

### B. *Outline of the approach*

Our input is a SPICE circuit netlist, and a netlist for a library of primitive templates. The overall flow of our approach is:

**Netlist flattening:** During preprocessing, we *flatten the input netlist* of the system. We do this to bypass designer-specified hierarchies, which are highly dependent on the choices of individual designers: different design houses, or even different designers within the same house, may create different hierarchy styles. Flattening allows our approach to be universal and independent of designer idiosyncrasies, and allows us to integrate design constraints (e.g., layout constraints) into the recognized blocks in a consistent manner. Beyond the issue of consistency, it should be noted that hierarchies that are logical to the netlist designer may not be the most logical for further optimization steps such as layout automation, e.g., passives have large dimensions as compared to active devices and need to be placed separately for matching constraints, but a schematic designer may create a hierarchies that keeps together the passive and the switches controlling them. Preprocessing also identifies netlist features that help performance but do not affect functionality (and can be disregarded during recognition) – the use of parallel transistors for sizing, series transistors for large transistor lengths; identification of transistor source and drain nodes using voltage levels; the presence of dummy transistors, decaps, etc.

**GNN-based recognition:** We create a graph representation of the flattened netlist and *identify sub-blocks* using a GNN-based approach (Section III) for approximate subgraph isomorphism. The output of this stage annotates nodes of the netlist as being part of specific sub-blocks. Note that it is possible for graph vertices to belong to multiple sub-blocks, e.g., a net that is an output of one sub-block may be an input of another. At this level of abstraction, there is substantial scope for variation in the way designers build circuits. The GNN recognizes features at various levels and identifies subcircuits corresponding to known functionalities, even under design style variations.

**Primitive annotation:** We take each sub-block and *recursively identify* lower-level blocks primitives within it. At the lowest level, primitives are detected using an exact graph isomorphism approach rather than a GNN: the element-level structure of primitives is invariant across circuits and we may work with a library of patterns that can be recognized using exact subgraph isomorphism (Section III-D).

**Postprocessing:** After primitive annotation, we use postprocessing to determine which primitives are an integral part of a specific unit, and which are auxiliary to the unit (e.g., input buffers) and can be separated. This step is necessary because any neural network based approach can never be 100% accurate under a limited training set. Even with a rich training set, we run the risk of overfitting if we attempt to capture every possible detail in every sub-block variant. We show in Section VI how simple postprocessing can enhance recognition accuracy to very high levels (100% for all of our testcases).

Each recognition step help in providing a set of substructures that can be transmitted to a placement/routing algorithm to be independently placed. Moreover, after recognizing a substructure, it is easy to annotate it with any geometric constraints (e.g., symmetry, matching, or common centroid) that must be honored by layout tools.

### C. Graph representation of the netlist

We represent an element-level circuit netlist as an undirected bipartite graph $G(V, E, R)$, where $V$ is the vertex set, $E$ is the edge set, and $R$ is the set of edge relations. The set of vertices $V$ can be partitioned into two subsets, $V_e$, corresponding to the elements (transistors/passives) in the netlist, and $V_n$, the set of nets. For each edge $(v_i, v_j) \in E$, one of $v_i$ of $v_j$ lies in $V_e$ and the other in $V_n$. Each device vertex stores the information related to the type of a device (e.g., NMOS, PMOS, resistor, capacitor) and related device properties, and each net vertex stores information about net type (e.g., ports, signal, power, and ground). A net may be connected to different terminals of a transistor (source, drain, and gate) or a passive device: this information is stored in a relation $r_{ij} \in R$ associated with the edge $(v_i, v_j) \in E$, and is used to infer disparate circuit functionality.

In an analog circuit, a connection between terminals of the same device, e.g., a drain-gate connection of M0 in Fig. 2(a), has an important functionality. Using separate vertices for nets and elements provides us a graph representation without any self-loop; in contrast, using element vertices only would result in a self-loop. The ability to handle a circuit graph without creating self-loops is an important feature of this representation: most GNN methods add a self-loop to all the vertices in a graph [11], which allow the GNN to include features of the vertex while calculating its embeddings. In general, the graph representation used in GNNs allows for only one self-loop, and any preexisting self-loops in the graph are removed. This can cause a loss of a critical information in the circuit representation if only element vertices are used: our model with separate net and element vertices eliminates this problem.

In one representation, when nets are connected to multiple terminals of the same device, e.g., net D0 is connected to drain as well as gate of transistor M0 in Fig. 2(a), each connection can be represented as a single edge with a label (e.g., {D, G}). These edge labels can be converted directly to an edge weight [12] and trained using standard GNN approaches (Section III-B). We also use this representation for identifying simple primitives using subgraph isomorphism (Section IV). An alternative representation for a circuit is in the form of a *multigraph* [19], [25]: this is a graph that can have multiple edges of various edge types between pairs of vertices. As we will show, in our context, the different types of edges may correspond to each device terminal connection that is connected to a net. This representation allows a GNN (Section III-C) to learn from each edge type separately.

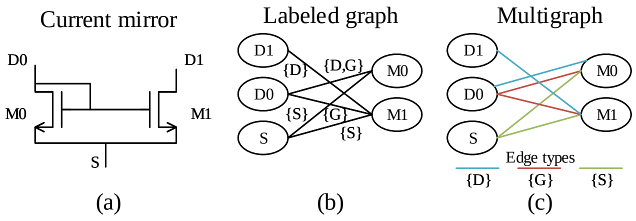

**Figure 2:** (a) An NMOS current mirror primitive, CM-N(2), with two transistors. (b) Its representation as a bipartite graph with labeled edges. (c) Its multigraph representation with color coded edges.

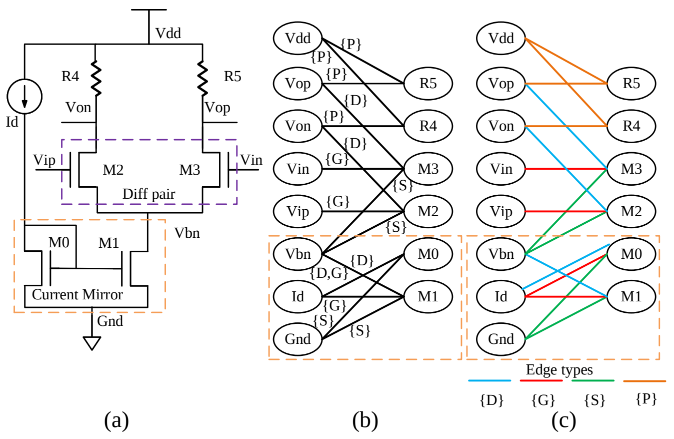

**Figure 3:** (a) A differential OTA (for simplicity, body connections are not shown). (b) Its bipartite graph, showing the subgraph that can be recognized as a current mirror (for clarity, edge labels are not shown).

Fig. 2 illustrates an example of a current mirror primitive with two transistors, M0 and M1, and the edge labeled graph (Fig. 2(b)) as well as the multigraph (Fig. 2(c)) corresponding to this structure. In both graphs, the vertices on the right represent $V_e$ and vertices on the left represent $V_n$. Fig. 2(b) shows the edges connecting the vertices of the graph and the edge labels represent the terminals they are connected. For example, the connection between net vertex D0 and element vertex M0 is represented by an edge between these vertices in the graph, and the edge is labeled as {D, G}, representing the drain and gate terminals of transistor M0 connected to the net D0. Fig. 2(c) shows a multigraph representation of the same graph where, instead of multiple edge labels on one edge, multiple edges of different types, corresponding to (drain (blue), gate (red), and source (green)), are defined. Similar approaches can be used for larger structures: Fig. 3 shows a differential OTA without bias details, its bipartite graph with edge labels, and the multigraph representation. Here, we also show passive devices (resistors); edges connected to passive devices are labeled as {P} (orange).

**Vertex features:** Our implementation associates vertices in the graph with 18 features and four edge types are considered based on their connection to the drain, gate, or source terminals of a transistor or connection to a passive device terminal:

- 6 features that annotate the vertex type, e.g., whether it is an NMOS/PMOS transistor, resistor, inductor, capacitor, and net.
- 9 features that denote the type of net – input, output, bias signal, enable, io port, antenna (for RF Receivers), clock, power, and ground.
- 3 features corresponding to size of the devices. We normalize the size of devices in a circuit and classify them as low, medium, or high (e.g., this can be used to distinguish a DC-DC converter from a filter since the former uses much larger capacitors).

## III. GNN BASED RECOGNITION

### A. Overview of GNNs

GNNs are learning models that are specific to graph data structures. Recent advances in GNNs have shown remarkable results on graph-structured data such as chemical compounds, social networks, and circuits. The input to a GNN is a graph and the vertex features. From this representation, GNNs extract the graph structure information in form of vertex or graph embeddings, i.e., vertices close to each other in the graph must be close to each other in the embedding space. Similar to convolutional neural networks (CNNs) that operate on image data, GNNs learn the embeddings of a vertex in the graph by aggregating information from nearby neighbors and the vertex embeddings. Based on this embedding, a GNN can achieve good separation between the feature representations of vertices in a graph.

Using these embeddings, extracted directly from the graph structure, GNNs can be generalized to perform inference on previously unseen graphs.

ChebNet [10], which was inspired by the graph Fourier transform, is one of the early works in this area. It proposes a spectral graph convolution operation that is similar to the convolutions used in CNNs. In this context, spectral convolutions are defined as the multiplication of a signal (vertex features) by a kernel made of Chebyshev polynomials of the diagonal matrix of Laplacian eigenvalues. These Chebyshev polynomials can be approximated using a truncated expansion. In particular, a $K^{\text{th}}$-order polynomial acts as a spectral filter that extracts the information from a local neighborhood consisting of all vertices that are up to $K$ edges away in the graph.

Like ChebNet, other early GNN models such as the Graph Convolutional Network (GCN) [11], GraphSAGE [27], and Graph Attention Network (GAT) [28], focused on vertex features and considered all edges to be of identical types. Variants such as RGCN [29] and RGAT [30], extended the principles of GNNs, which operate on graphs, to multigraphs. These methods learn embeddings on each relation (edge type) individually and then aggregate these embeddings together. These multigraph models have been used on knowledge graphs and achieved significant performance improvements [29]. In this work, we compare the use of multiple GNN architectures and see a notable improvement in results using multigraphs.

A high-level view of an embedding update method for a GNN layer on multigraphs is shown in Fig.4. Approaches for computing the vertex embedding can be divided into two stages. In the first stage (aggregation), the embedding from the neighbors of the vertex are aggregated using an aggregator (green box), separately for each edge relationship. In the second stage (accumulation) the aggregated embeddings are accumulated using different operators (orange box), such as mean or concatenation. These combined embeddings are then passed through an activation function to obtain the updated embedding for each node.

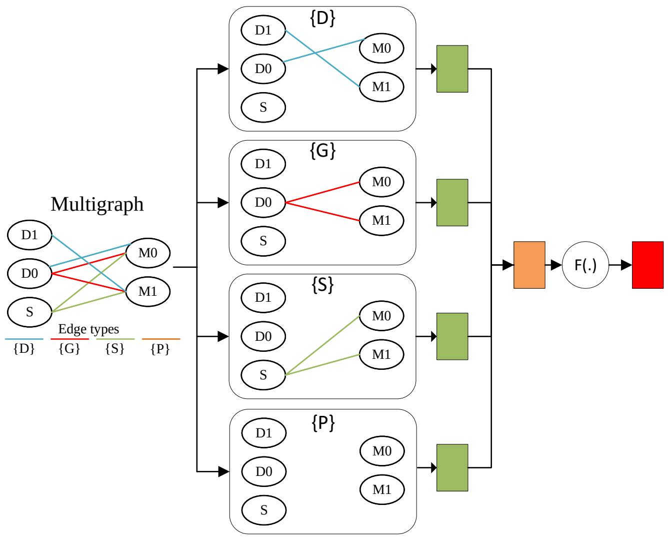

**Figure 4:** Overview of computing the update of a single graph node embeddings. Activations from neighbors (green) are gathered for each relation type individually. These activations are then accumulated (orange) and passed through an activation function which is used to update the vertex embeddings (red). *We also consider edge types corresponding to passives which has not been shown here due to absence of passives in this circuit.

A detailed explanation of various GNN models is provided in the next two subsections.

### B. GNNs on simple graphs

GNN models on simple graphs consider all edges to be of same type, and thus have a common aggregation stage, without a separate accumulation stage. For each model, Table II provides the mathematical computations in the aggregation step. In the table, $AGG_i$ is the function that aggregates embeddings from the nearest neighboring vertex set $N(i)$ of a vertex $v_i$; $h_j \in \mathbb{R}^{\mathbb{F}}$ refers to the embeddings of dimension $\mathbb{F}$ of a vertex $v_j \in N(i)$; $W \in \mathbb{R}^{\mathbb{F}\times\mathbb{F}}$ is a trainable parameter corresponding to every node; $C_{i,j}$ is a normalization coefficient; and $b$ corresponds a bias term. The result of accumulation, $ACC_i = AGG_i$, for this class of GNN models.

**GCN.** In a GCN [11], the $K-$localized Chebyshev spectral filters in ChebNet are extended to a first-order filter with $K = 1$, thus limiting the feature extraction to the nearest neighbors. This makes GCNs much more computationally efficient than ChebNet. Multiple GCN layers can be stacked to extract information from a larger neighborhood, where the

**Table II:** A summary of operations for various GNNs that operate on graphs and multigraphs.

| | GNN method | Aggregation method | Accumulation method |
|---|---|---|---|
| **For graphs** | GCN [11] | $AGG_i = b + \sum_{v_j \in N(i)} \frac{1}{c_{ij}} W h_j$ | $ACC_i = AGG_i$ |
| | GraphSAGE-pool [27] | $AGG_i = W' \cdot \text{concat}\,(h_i,\ \max(h_j, \forall v_j \in N(i)) + b)$ | $ACC_i = AGG_i$ |
| | GAT [28] | $e_{i,j} = \text{LeakyReLu}(\overrightarrow{a}^T \text{concat}(W h_i, W h_j))$<br>$\alpha_{i,j} = \text{softmax}_j(e_{i,j})$<br>$AGG_i = \sum_{v_j \in N(i)} \alpha_{i,j} W h_j$ | $ACC_i = AGG_i$ |
| **For multigraphs** | RGCN [29] | $AGG_{i,r} = \sum_{v_j \in N(i)^r} \frac{1}{\lvert N^r(i) \rvert} W_r h_j$ | $ACC_i = W_0 h_i + \sum_{r \in R} AGG_{i,r}$ |
| | GraphSAGE-pool+ [13] | $AGG_{i,r} = W'_r \cdot \text{concat}\,(h_i,\ \max(h_j, \forall v_j \in N^r(i)) + b)$ | $ACC_i = W' \cdot \text{concat}(\sum_{r \in R} AGG_{i,r}, h_i + b)$ |
| | RGAT [30] | $e_{i,r,j} = \overrightarrow{a}^T \text{concat}(W_r h_i, W_r h_j)$<br>$\alpha_{i,r,j} = \text{softmax}_j(e_{i,r,j})$ [WIRGAT and ParaGraph]<br>**or,** $\alpha_{i,r,j} = \text{softmax}_{r,j}(e_{i,r,j})$ [ARGAT]<br>$AGG_{i,r} = \sum_{v_j \in N^r(i)} \alpha_{i,r,j} W_r h_j$ | $ACC_i = \sum_{r \in R} AGG_{i,r}$ [WIRGAT, ARGAT]<br>**or** $ACC_i = W' \cdot \text{concat}(\sum_{r \in R} AGG_{i,r},\ h_i + b)$<br>[ParaGraph] |

addition of each layer increases the radius of the region from which information is extracted by 1.

The aggregation function is shown in Table II. The normalization coefficient $C_{i,j}$ is defined as $\sqrt{|N(i)||N(j)|}$ where $|N(i)|$ and $|N(j)|$ are the degree of vertices $v_i$ and $v_j$, respectively. Passing $AGG_i$ through a ReLU activation function provides the updated embedding of a vertex.

**GraphSAGE.** GraphSAGE [27] leverages a convolutional aggregator in the spatial domain to approximate the localized filters. It proposes multiple aggregation filters such as mean, LSTM, and pool. We use the max-pool filter in our evaluation as it is shown to have better accuracy. The vertex embeddings are updated by concatenating (using the "concat" operator) the embeddings with the aggregated neighbor embeddings. This is similar to a skip connection between different layers of the GraphSAGE algorithm. These concatenated embeddings are then fed through a fully connected layer $W' \in \mathbb{R}^{\mathbb{F} \times 2\mathbb{F}}$, which transforms the representations to be used at the next step of the algorithm. GraphSAGE also adds a neighborhood sampling method which improves its scalability for graphs with a large number of neighbors.

**GAT.** Both GCN and GraphSAGE consider all edges to be of identical importance, and therefore the importance of the neighborhood is dependent on the graph structure. Specifically, higher degree nodes are generally observed to achieve higher accuracy in mean-based aggregators than lower degree nodes. This issue is resolved using attention-based weights in GAT [28] networks, which compute the importance of each neighbor of a vertex implicitly using attention weights. A relative attention score, $\alpha_{i,j}$, is computed based on vertex embeddings, $e_{i,j}$, using a softmax function $P_i(\overrightarrow{a}) = \frac{e^{a_i}}{\sum_k e^{a_k}}$. A updated embedding is computed using a weighted mean of the neighborhood embeddings, where $\alpha_{i,j}$ operates as weight of the neighbor.

## C. GNNs on multigraphs

The models built for simple graphs considered all edges of a graph to be same type, but cannot be directly used for multigraphs. Recall that for our problem, the GNN must differentiate, for example, between the connections of a net to the drain, source, and gate of a transistor, and in order to consider such relationships, we employ a multigraph (Section II-C) to represent a circuit. The RGCN [29] and RGAT [30] models extend the principles from GCN and GAT to multigraphs whose edge relationships are based on which port of a device a net is incident on. An extension of GraphSAGE that provides the flexibility to be applied to multigraphs is provided in [13]. These algorithms transform each neighbor of a graph vertex in terms of the relation on the edge, and use multiple adjacency matrices corresponding to each edge relationship.

In the table, $AGG_{i,r}$ is the aggregated embedding from the nearest neighboring vertex set $N(i)$ of a vertex $v_i$ corresponding to each relation type; $h_j \in \mathbb{R}^{\mathbb{F}}$ refers to the embeddings of dimension $\mathbb{F}$ of a vertex $v_j \in N(i)$; $N^r(i)$ is the set of neighbors of vertex $i$ with relationship $r \in R$; $W_r \in \mathbb{R}^{\mathbb{F} \times \mathbb{F}}$ is a relation-specific trainable parameter corresponding to every node; $W'_r \in \mathbb{R}^{\mathbb{F} \times 2\mathbb{F}}$ is a trainable parameter corresponding to concatenated features; and $b$ is a bias term.

**RGCN.** For multigraphs, RGCN applies different weight matrices to different relational edge groups and aggregates each group independently. In Table II, $|N^r(i)|$ is used as for normalization of weights, and $W_0$ is a trainable parameter corresponding to self-loops introduced in the graph. RGCN also introduces a basis-decomposition method (not shown in table) for regularization to reduce overfitting in cases of large number of relationships between nodes. As the number of edge types for our graphs is not large, we do not observe this overfitting problem even without the use of basis-decomposition.

**GraphSAGE+. (HinSAGE)** is an extension of the GraphSAGE algorithm for relational data. We use the GraphSAGE-pool aggregator for each relation individually and then use a mean for accumulating embeddings corresponding to each relation. These accumulated embeddings are again concatenated with the vertex embeddings, and a fully-connected layer $W'$ is applied to convert the $2\mathbb{F}$-dimensional concatenated embeddings to $\mathbb{F}$-dimensional vertex embeddings. HinSAGE also provides the option of using other concatenation variations such as full, partial, and none. In this work, we choose the partial method for our comparison.

**RGAT.** Relation Graph Attention networks (RGAT) introduces two attention methods, Within-Relation Graph Attention (WIRGAT) and Across-Relation Graph Attention (ARGAT), extending attention mechanisms to the relational graph domain. In WIRGAT, once the neighborhood embeddings, $e_{i,r,j}$, are computed for each neighbor with relation $r$, a softmax is taken over each $e_{i,r,j}$ matrix for each relation type separately to form the attention coefficients $\alpha_{i,r,j}$; in ARGAT, the softmax is taken across all edges independent of relation types. Next, the relative normalization attention coefficients, $\alpha_{i,r,j}$, compute a linear combination of all embedding from the neighbors to get the intermediate embeddings for each relation type. During the accumulation stage, the embeddings from different relations are aggregated and passed through a ReLU activation function to obtain the updated embeddings.

ParaGraph [19] uses the key ideas of RGCN, GraphSAGE, and GAT to predict net parasitics. During the accumulation stage, it uses a concatenation of the sum of neighborhood embeddings with the vertex embeddings which are passed through a linear transformation using a fully connected layer ($W' \in \mathbb{R}^{\mathbb{F} \times 2\mathbb{F}}$) and ReLU activation function to get the final embeddings.

### D. Annotating the identified sub-blocks

As described in Section II-C, the circuit to be annotated can be represented in the form of a graph or multigraph. The task of annotation requires the vertices and edges of this graph to be classified as belonging to specific analog sub-blocks. In our work, we use the GNN algorithms described previously in this section to classify vertices in the graph as members of a specific sub-block. The sub-blocks identified by the GNN have known functionalities, and are typically associated with a set of specific constraints. For example, an OTA layout should be symmetric about a differential pair axis; it is vital for an LNA to be placed close to the antenna; devices in RF blocks such as LNAs and mixers need guard rings for isolation; oscillators and BPFs must be symmetric about a cross-coupled transistor pair. Moreover, recognizing the class of circuit brings forward other important constraints, e.g., if a sub-block is recognized as part of a wireless circuit, minimization of wire lengths is important due to the sensitivity to parasitics. Additionally, recognizing a sub-block enables the ability to provide performance constraints on that block: from the system-level constraints, we may infer block-level constraints, e.g., we may place constraints on parameters such as the gain, bandwidth, and offset on an OTA sub-block. During layout, this places constraints on the allowable wire parasitics on sensitive nodes of this sub-block [31]. Therefore, for every known category of blocks, it is possible to associate the recognized block with a set of layout constraints based on its functionality.

For the sample and hold circuit example shown in Fig. 1(c), the GNN classifies the nodes of the graph into two classes: bias nodes and OTA nodes. The results after GNN classification are shown in Fig. 5. Most of the nodes are identified correctly, except the resistor and capacitor connected from Vo1 and Vo2, and these misclassifications are rectified during the postprocessing step explained in Section V. After identifying the nodes in these sub-blocks, we identify the primitives using a library-based approach, as explained in the next section.

## IV. Annotating primitives

As described in Section III-D, once a sub-block is identified, the next step in the hierarchy is to determine the primitives that compose the sub-block. In our approach, we perform this recognition by matching the nodes of the circuit graph to those of predefined primitives. The major difference between primitive identification and sub-block identification is that primitives are fundamentally much simpler than sub-blocks: they contain fewer elements and have fewer variants. As a result, algorithmic graph-based matching techniques are adequate for identifying primitives, and GNN-based methods are not necessary at this level of hierarchy.

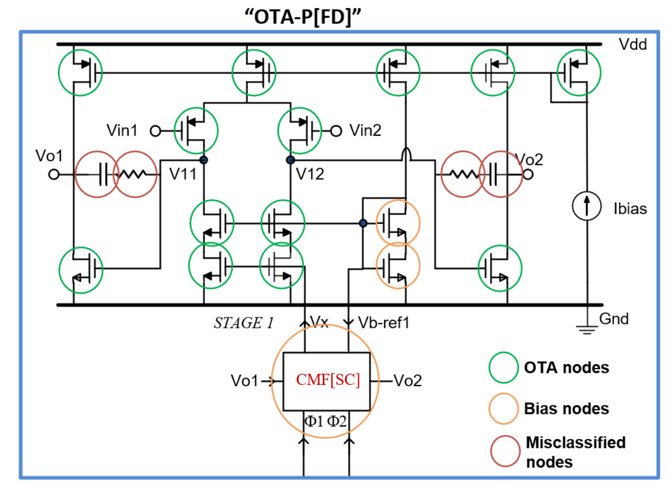

**Figure 5:** Graph nodes classified by the GNN as belonging to an OTA or a bias circuit. Green circles highlight the correct OTA classification, orange circles highlight the correct bias classification, and red circles show the OTA nodes that are misclassified as bias nodes.

We use a library of 21 basic primitives from ALIGN [20], [21] that are building blocks for larger sub-blocks. This list of primitives is derived from previous works in this field [32], [33] and some new structures encountered in testcases. The primitives are specified as SPICE netlists, enabling a user to easily add new primitives to the library. For each primitive, we perform a one-time translation to a graph (Section II-C).

### A. Identifying primitives

The problem of identifying primitives within a sub-block corresponds to performing subgraph isomorphism checks between the sub-block graph $G$ and pattern graph $\mathcal{G}_i$ for every element $i$ of a library of primitives. For example, the current mirror primitive of Fig. 2 is a subcircuit of the OTA, and correspondingly, the graph of the CM is a subgraph of the OTA. This is indicated by the blue edges in the Fig. 3, which match Fig. 2.

For the sample and hold circuit example as shown in Fig. 1(c), we identify the following primitives – one CM-P(5), one DP-P, two CC-RC, one CM-N(2), two CS-Amp-N, one VR-N(CB), and one CMF[SC] – as highlighted in Fig. 6.

The subgraph isomorphism problem is not known to be NP-complete, but no polynomial time solution is known for the general isomorphism problem. This problem is speculated to be in the NPI class of intermediate difficulty [34]. We use VF2, an established graph matching algorithm [35]. This method has a worst-case complexity of $\Theta(n!n)$ but for the general subgraph isomorphism problem, where $n$ is the number of

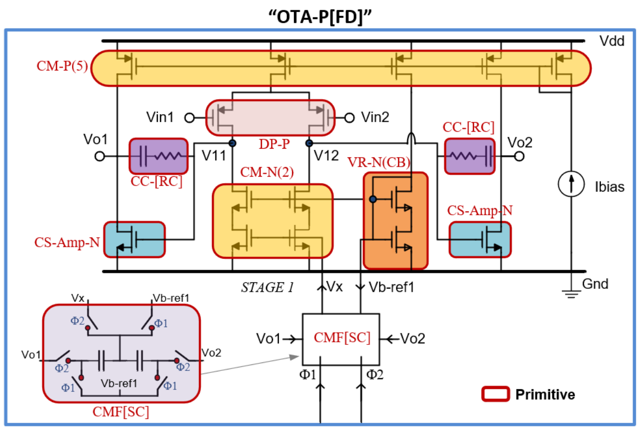

**Figure 6:** Primitives annotated in each sub-block identified by the GNN classifier.

vertices, for our problem where the library subgraph to be matched has $O(1)$ diameter and $O(1)$ degree, the complexity is $O(n)$. Specifically, the complexity of VF2 can be estimated by calculating the computation required for calculating the next candidate pair $P(s)$ of nodes $(p, q)$ from two graphs and determining the semantic feasibility of the new pair. Computing the next pair takes $O(N_1 + N_2)$ time where $N_1$ and $N_2$ are the sizes of the original graph and the subgraph. Since $N_2 = O(1)$, this becomes $O(N_1)$. The computation of semantic feasibility depends on the number of edges incident on nodes $p$ and $q$, which is $O(1)$ for a bounded degree graph. Thus, for our problem, VF2 has $O(n)$ complexity.

### *B. Layout constraint annotation*

The primitive library allows designer annotation of basic constraints. For instance, a symmetry and matching constraint can be set at the primitive level for a differential pair (DP) primitive. These geometric constraints can be transmitted to a layout generator and used to identify further higher-level symmetries involving groups of primitives or sub-blocks, e.g., in Fig. 1, the primitives for CM-N and DP in Stage 1 can be annotated with matching/common centroid constraints. When propagated to the next level, these two may be combined to ensure a common symmetry axis for both structures.

## V. Postprocessing

In general, GNNs cannot guarantee 100% accuracy in their classification results. Rather than placing the full burden of recognition on the GNNs, we use an engineering approach that allows the GNN to perform a large fraction of the recognition task and then uses a set of simple postprocessing heuristics to complete the annotation. As we will see, a simple set of postprocessing steps brings the recognition accuracy to close to 100% for the evaluated circuits. Our postprocessing operation uses simple graph-based heuristics after GCN classification in which we associate the nodes that belong to the same channel-connected component (CCC)[1] with a sub-block. This idea of using CCC is driven by the observation that the considered hierarchies are separated by voltage signals connected to gate of transistors as current signals across hierarchies can create mismatch which are a major concern for analog designs. Capacitor connections are also considered as endpoint of CCC.

After postprocessing, we cluster nodes with similar labels into sub-blocks and identify all primitives within a using graph-based approaches as described in Section IV for extracting primitives within a sub-block. All primitives that are an integral part of a sub-block (e.g., a differential pair in an OTA) are added to the hierarchy tree at the same level; a primitive that can be considered a stand-alone unit (e.g., an input buffer for an oscillator, encountered in our phased array example in Section VI) is separated from the sub-block and listed as a stand-alone primitive in the hierarchy tree. Together, these heuristics solve most of the misclassifications in the training and test set for the given class of circuits.

For the sample and hold circuit example in Fig. 1(c), the results after GNN classification (Fig. 5) consist of four misclassified nodes in the graph, corresponding to the encircled resistor and capacitor elements. These nodes are connected to Vo1 and Vo2 pins at the drain terminal of the differential pair primitive. Based on CCC rule i.e., the nodes connected to drain/source terminal of the transistor are part of same cluster these nodes and are considered as part of OTA sub-block. The final hierarchy for layout generation is shown in Fig. 1(b) and Fig. 1(c).

**Generating layout:** To illustrate a use case for circuit annotation, we transmit the results of circuit recognition to the ALIGN [37] custom layout generator under the ASAP7 [38] PDK. The hierarchies identified by our algorithm are used by the layout tool to construct layouts for primitives, which are assembled into layouts for larger blocks. For example, by clustering the OTA circuit (highlighted in dotted green box) vertices together and annotating the differential pair and current mirror primitives, we build a hierarchical representation that generates layouts for these primitives and assembles them together. The generated layout is shown in Fig. 7. The annotation helps in identifying and clustering the central annotated OTA sub-block shown in the figure. Based on identification of the OTA sub-block, we generate symmetry constraints for the OTA sub-block with a common axis of symmetry (dotted red line). The symmetry and proximity constraints are first detected at the primitive level and propagated to other levels of hierarchy (Section IV(B)), creating a common axis of symmetry for the entire layout. The symmetric capacitors inside the OTA are laid out in a common centroid fashion. The bias circuit does not require matching and is optimized for area and wirelength.

## VI. Experimental results

Our flow is implemented in Python 3.8.0 including a TensorFlow library for training and testing of the GNN, and the scikit-learn library for sparse matrix computation.

### *A. Datasets*

The dataset for the GNN was taken from several sources, including standard textbooks [3], [39] and papers in the

[1]A channel-connected component is a cluster of transistors connected at the sources and drains (not counting connections to supply and ground nodes). It can be identified using simple linear-time graph traversal schemes [36].

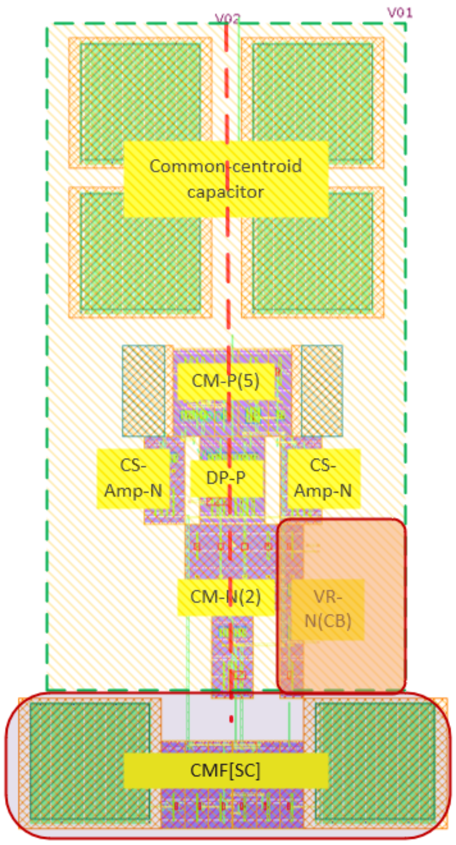

**Figure 7:** Layout of the sample and hold circuit in Fig. 1(c) using the ALIGN layout generator [37], guided by the hierarchy extracted using our approach.

literature (e.g., [40]–[43]). We chose the SPICE format because it is the most natural and universal mode in which an analog designer (who typically does not have experience with graph abstractions) may use the software can expand the training set. Using these sources, two labeled datasets for OTA and radio frequency (RF) sub-blocks have been created with characteristics as listed in Table III. These benchmarks are publicly available at [44]. The OTA-bias dataset consists of multiple OTA configuration with appropriate signal and bias subcircuit labels, while the RF-data dataset consists of different RF circuits, with labels attached to elements that compose low noise amplifiers (LNAs), mixers, oscillators (OSC), and baseband amplifiers. The dataset is split into an 80% : 20% ratio for the training set and validation set.

**Table III:** Statistics of our dataset.

| Datasets | # Circuits | # Nodes | # Labels | # Features |
|---|---|---|---|---|
| OTA-bias | 1026 | 52346 | 2 | 18 |
| RF-data | 2597 | 77918 | 2 or 3 or 4 | 18 |

The input to the GNN is the circuit graph, $G(V, E, R)$, in the form of sparse matrix and an $n \times d$ matrix of features, where $d$ is the number of features. All these models work on directed graphs but can be adapted to the undirected graph of circuits by providing edges in both directions.

The task of the GNN is to (a) identify and extract such features, (b) compose a combination of [possibly approximate versions of] these features from the training set to infer circuit functionality as one of several trained classes of circuits, and (c) identify the vertices (primitives) and edges (nets) that belong to the recognized class.

## B. Architecture and hyperparameter optimization

All of the GNN architectures listed in Table II are evaluated to determine their performance on our annotation problem. The implementation of the algorithms is similar to their descriptions in the source references, with minor changes and specific choices of settings. This inludes the type of layers, activation functions, regularization, and the choice of dropout. For GraphSAGE, as suggested in [27], we use max-pooling for aggregation. GraphSAGE also proposes a sampling strategy (randomly selecting 20 neighbors): this can be helpful for larger graphs but does not provide any improvements for the analog circuits, where the degree of a vertex is typically small (less than three), and therefore we do not use sampling. For a balanced comparison, we use same learning rate (0.001), optimizer (Adam), error function (log softmax), initialization weights (from [45]), learning rate decay (none), maximum number of training epochs (200), pooling layers (none), and batch size (1).

*Comparison metric:* As the size of different classes are not balanced, we use the F1-score as a metric for comparison. This metric is defined as

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1}$$

where Precision and Recall are defined as:

$$\text{Precision (PPV)} = \frac{TP}{TP + FP} \tag{2}$$

$$\text{Recall (TPR)} = \frac{TP}{TP + FN} \tag{3}$$

where $TP$ is the number of true positives, $FP$ is the number of of false positives, and $FN$ is the number of false negatives. We also calculate F1-scores after postprocessing ("F1-p") and analyze them in the next section.

*Baseline:* As a baseline, we use GANA from the preliminary conference version of this work [12]. This approach includes pooling layers with a ChebNet convolution, with a $K$-localized spectral Chebyshev filter for convolution in the spectral domain, derived from [10], setting $K = 6$. In this work, we do not use the pooling layers as they do not have any trainable parameters, and thus do not provide any significant improvements.

*Number of layers optimization:* The models consist of multiple stacked layers of GNN methods with ReLU activation function. We calculate the most efficient number of GNN layers as evaluated based on grid search. Fig. 8, shows a comparison of the F1-score for various GNN approaches for different numbers of layers. For enhanced readability of the figure, we only show WIRGAT from the three RGAT methods (WIRGAT, ARGAT, and ParaGraph).

As shown in Fig. 8, we see an improvement in results going from a single layer to four layers; beyond this, increasing the number of layers provides no significant improvement in accuracy. The performance in node representation learning remains the same or degrades as the graph convolutional models become deeper. This matches the normal understanding of GNNs, where the nearer neighbors have the most impact on a node, and increasing the number of convolution layers results in less distinguishable representations for nodes. Each graph convolution layer acts as an aggregator of characteristics of neighboring nodes, and adding too many layers over-smoothens the output features [46]. Although some recent methods [47] can use deeper models, they require larger training data – but large training sets of analog circuits are not easily found in the public domain.

of number of stacked GNN layers.png

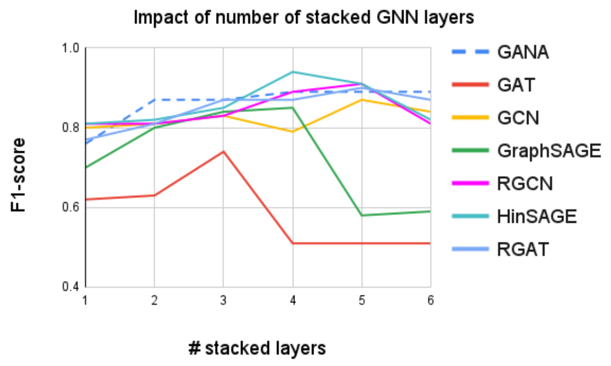

**Figure 8:** Change in the validation accuracy for various GNN architectures, on the OTA-bias dataset, as the number of stacked layers is increased.

**Table IV:** Comparison of validation set F1-score and training runtime.

| GNN methods | OTA-bias | | RF-data | |
|---|---|---|---|---|
| | F1-score | train-time | F1-score | train-time |
| GANA(K=6) | **88.20± 0.31** | 203 min | **87.61± 0.34** | 242 min |
| GCN | 87.5± 0.29 | 81 min | 85.36± 0.31 | 89 min |
| GraphSAGE | 86.86± 0.36 | 62 min | 86.51± 0.41 | 67 min |
| GAT | 74.40± 0.23 | 105 min | 76.46± 0.32 | 125 min |
| RGCN | 89.18 ± 0.21 | 182 min | 88.18± 0.27 | 198 min |
| GraphSAGE+ | **92.31 ± 0.23** | 222 min | **90.11± 0.28** | 246 min |
| WIRGAT | 90.26 ± 0.25 | 224 min | 88.16± 0.24 | 256 min |
| ARGAT | 89.36 ± 0.24 | 298 min | 87.26± 0.29 | 324 min |
| ParaGraph | 89.24 ± 0.23 | 202 min | 87.15± 0.28 | 227 min |

The F1-scores corresponding to the best accuracy are listed in Table IV. The upper box in the table corresponds to graph-based architectures and the lower box to multigraph-based architectures. A five-fold cross validation is used to reduce the sensitivity to data partitioning. The mean and variance of the scores are listed in the Table IV. The best method in each box is highlighted in bold characters. For graph-based architectures, we observe that the other candidate approaches do not provide any noticeable improvement in the F1-score over GANA. The GNN methods with edge weights (GANA, GCN, GraphSAGE) performed better than native GAT, which we found is due to extremely low scores for weight initialization in GAT. For multigraph-based architectures, a significant improvement is achieved over GANA, particularly for GraphSAGE+. Comparing individual methods between simple and multigraph approaches, we observe an improvement from GCN to RGCN for the OTA-bias dataset. Between two possible RGAT [30] accumulation mechanisms, WIRGAT and ARGAT, we observed that WIRGAT performs better than ARGAT, indicating that the edge relations are independent in nature. The best results are obtained for GraphSAGE+, which uses a concatenation of the embeddings from different relations: therefore, we select it for final experimental evaluation in the next section.

The improvements in accuracy can primarily be attributed to a change from a bipartite graph representation to a bipartite multigraph representation, which represents all connections to a multipin element such as a transistor without weighting any connection more highly than the others, as in GANA.

### C. Experimental evaluation

To evaluate the complete annotation procedure (GraphSAGE+ with postprocessing), we apply it on multiple different analog designs, as summarized in Table V. The five OTA test cases (OTA1–5) and five receiver test cases (RF1–5) are different variants of OTA and receiver circuits and are not included in the training data. We also test our annotation scheme on, a switched capacitor filter, a sample-and-hold circuit, a mixer-first receiver with oscillator, and a phased array test case. These examples are more complex systems that include OTA and RF sub-blocks, respectively, and correspond to hand-crafted circuits provided to us by designers. The GNN output is compared against annotated solutions provided by human designers.

**Table V:** Statistics of the test dataset. For the set of five circuits (min–max) is shown.

| Datasets | # Circuits | # Nodes | # Labels |
|---|---|---|---|
| OTA | 5 | 22–38 | 2 |
| Switched capacitor filter | 1 | 58 | 2 |
| Sample-and-hold | 1 | 92 | 2 |
| Receiver | 5 | 64–92 | 3 |
| Phased array | 1 | 928 | 4 |
| Mixer-first Rx with oscillator | 1 | 64 | 3 |

The results on the designs from Table V are summarized in Table VI. In the table, we compare the accuracy of our new approach (GraphSAGE+) with GANA on specific circuits. We also show F1-scores after postprocessing (Sec. V) in column "F1-p" achieving close to 100% classification accuracy for most of the designs. The OTA and receiver class designs are separated by a double horizontal line. An analysis of our results for annotating these complex circuits is provided in the following paragraphs.

*Switched capacitor filter (SCF)*: This circuit is taken from our previous work [12]. It is similar to the sample-and-hold circuit in Fig. 1(a) and contains 58 vertices (32 devices and 26 nets), including an OTA sub-block and switched capacitors. The OTA sub-block used in this circuit is not seen by the training set. The design netlist is initially flattened, and various features such as the device type, connectivity information, and sizing are extracted. As described in Step 1 in Section II-B, a preprocessing step that involves ignoring (for recognition purposes only) dummy devices and stacking of resistors,

**Table VI:** Comparison of GANA vs. our approach for subcircuit identification, mean recall (TPR), mean precision (PPV), F1-score, and processed F1-score (F1-p) for various benchmark circuits.

| **Benchmark** | | | **GANA** | | | | | **This work (GraphSAGE-pool+)** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Design | # Devices | # Nets | TPR | PPV | F1-score | F1-p | Runtime | TPR | PPV | F1-score | F1-p | Runtime |
| OTA1 | 12 | 14 | 0.91 | 0.86 | 0.88 | 1.00 | 3.5ms | 0.87 | 0.90 | 0.88 | 1.00 | 5.7ms |
| OTA2 | 15 | 19 | 0.94 | 0.87 | 0.90 | 0.96 | 2.9ms | 0.94 | 0.90 | 0.92 | 1.00 | 5.2ms |
| OTA3 | 17 | 22 | 0.93 | 0.78 | 0.84 | 0.94 | 3.0ms | 0.93 | 0.90 | 0.91 | 1.00 | 5.3ms |
| OTA4 | 18 | 17 | 0.83 | 0.95 | 0.87 | 1.00 | 2.8ms | 0.91 | 0.92 | 0.91 | 0.98 | 5.2ms |
| OTA5 | 24 | 19 | 0.95 | 0.87 | 0.91 | 0.98 | 3.2ms | 0.97 | 0.88 | 0.92 | 1.00 | 5.4ms |
| SCF | 32 | 26 | 0.94 | 0.92 | 0.93 | 1.00 | 4.1ms | 0.92 | 0.98 | 0.94 | 1.00 | 6.1ms |
| Sample-and-hold | 58 | 34 | 0.87 | 0.86 | 0.86 | 0.92 | 4.2ms | 0.86 | 0.91 | 0.88 | 1.00 | 6.3ms |
| RF1 | 46 | 27 | 0.81 | 0.86 | 0.84 | 0.96 | 3.8ms | 0.93 | 0.92 | 0.92 | 1.00 | 6.1ms |
| RF2 | 31 | 33 | 0.88 | 0.89 | 0.88 | 0.92 | 3.7ms | 0.88 | 0.87 | 0.87 | 1.00 | 5.4ms |
| RF3 | 57 | 54 | 0.91 | 0.76 | 0.84 | 0.94 | 4.2ms | 0.90 | 0.92 | 0.91 | 0.98 | 6.7ms |
| RF4 | 24 | 24 | 0.84 | 0.88 | 0.86 | 0.98 | 3.2ms | 0.92 | 0.91 | 0.91 | 1.00 | 5.4ms |
| RF5 | 45 | 49 | 0.84 | 0.84 | 0.84 | 1.00 | 4.1ms | 0.84 | 0.86 | 0.85 | 1.00 | 5.2ms |
| Phased array | 546 | 382 | 0.78 | 0.76 | **0.77** | 1.00 | 6.8ms | 0.88 | 0.86 | **0.87** | 0.98 | 12.1ms |
| Mixer-first Rx | 64 | 32 | 0.88 | 0.92 | 0.90 | 0.98 | 4.2ms | 0.98 | 0.89 | 0.94 | 1.00 | 4.3ms |

capacitors, and transistors is performed. For the accuracy calculation, we only consider device labels as nets can be connected between different hierarchies, and thus can be a part of multiple hierarchies.

Using our procedure, within the switched capacitor filter, we identify the OTA sub-block and bias circuit sub-block. We further identify primitives within the OTA and the bias circuit. After GraphSAGE+ classification, one of the bias nodes is misclassified as an OTA node but is rectified after the postprocessing step, and 100% of all nodes are correctly classified.

*Sample and hold circuit*: This circuit, shown in in Fig. 1(a), contains 92 vertices (58 devices and 34 nets). The two-stage of the OTA and the bias sub-block is identified using the GraphSAGE+ and primitives such as CS-Amp-N, DP-P, CM-P, CM-N, CC-[RC], CMF[SC] are identified during primitive identification. The common-mode feedback circuit was unique to this with no such data in the training set resulting in lower accuracy after GNN-based recognition. After postprocessing, based on the identification of CCCs, the CS-Amp-N transistors, and CC-[RC] (compensation capacitors) are identified as part of OTA signal path, while the CMF[SC] gets identified as part of the bias-circuit thus providing us 100% classification accuracy.

*Phased array circuit*: This test case consists of a phased array system [48], with four beams is illustrated in Fig. 9, containing a mixer (red), LNA (blue), oscillator (gray), and clock buffer primitive (BUF) (violet) sub-blocks. The output corresponding to each of the four beams feeds into baseband amplifiers (OTA). The graph for the input netlist has 928 vertices (546 devices (transistors and passives) + 382 nets), and the number of devices in each sub-block is mentioned in the figure (two LNA devices of size 39 each, two oscillators of size 32, two oscillators of size 34, four clock buffers of size 14, four mixers of size 30, and eight baseband amplifiers of size 20 (not shown)).

The GraphSAGE+-based classification identifies nodes belonging to LNA, mixer, baseband amplifier (OTA), and oscillator and passes these results through postprocessing and primitive identification.

After postprocessing, BUF primitives are identified and a separate hierarchy is created for them which boosts the F1-score to 0.98. The remaining devices are the bias resistors, which are connected to the output of mixer and are being identified as part of the baseband amplifier. However, this misclassification does not cause any impact on performance as these large resistors are only used for biasing. All sub-blocks (LNA, oscillator, baseband amplifiers, and mixer) inside the dotted blue box of phased array system are highlighted in the diagram. We create the next levels of hierarchy by identifying primitives inside these sub-blocks.

In this circuit we see significant benefit of our new GNN flow (F1-score: 0.87) compared to GANA (F1-score: 0.77) where we were observing a misclassification of LNA and the oscillator circuits because of similar topologies. In GANA the nodes corresponding to these two sub-blocks were filtered during postprocessing-II based on identifying the antenna port and the oscillating signal at the oscillator nodes. In this approach, we do not require the postprocessing II and these sub-blocks are identified correctly during GNN.

*Mixer-first receiver circuit*: This circuit as illustrated in Fig. 10, consists of a passive mixer [49] (dotted red), a coupled quadrature local oscillator [39] (gray), and a baseband amplifier [50] (green). The RF input directly feeds into the mixer. This circuit consists of 64 devices (transistors and passives) and 32 nets the number of devices in each sub-block is listed in the figure). The GraphSAGE+-based classification identifies nodes belonging to mixer, oscillator, and baseband amplifier (OTA). Our GNN framework achieves a F1-score of 0.94. The resistor connected to the input and output of the OTA is misclassifed as part of mixer after GraphSAGE+-based classification but gets corrected during post-processing generating correct labels for all the devices.

Our annotation scheme is fast, and is dominated by the runtime of the GNN (GraphSAGE+): in comparison, post-processing time is negligible (less than $1ms$ for graphs with less than a thousand nodes). We report numbers on the more

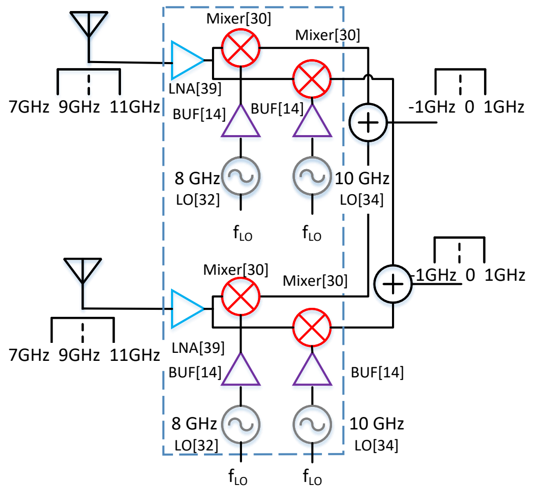

**Figure 9:** Block diagram of a phased array system [48]. The numbers in square brackets next to each sub-block indicate the number of transistors+passives in it (e.g., the LNA has 39 devices/passives).

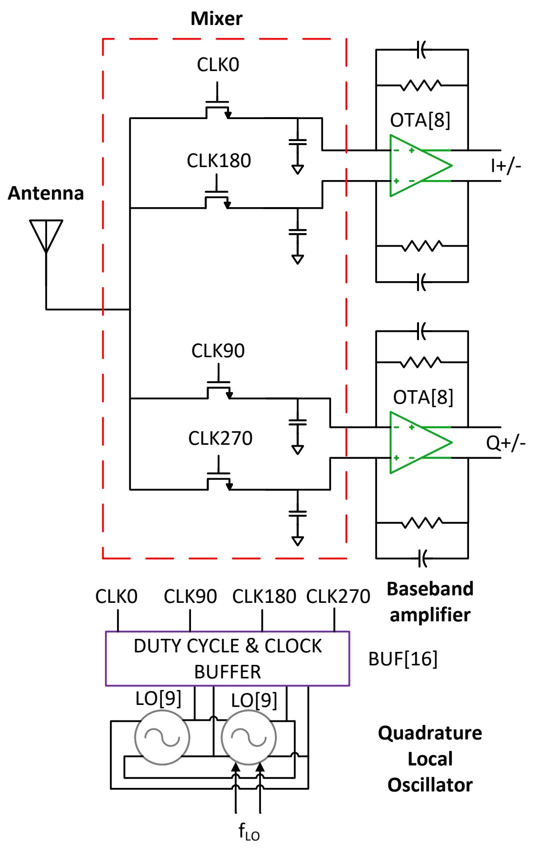

**Figure 10:** Block diagram of a mixer-first receiver with oscillator [49]. As in Fig. 9, the numbers in square brackets refer to the number of transistors+passives in each sub-block.

complex circuits on an Ubuntu host with a 2.6GHz Intel Core i7 processor with 8 cores and 32GB RAM the procedure takes 6.1 ms for the switched capacitor filter circuit, 6.2 ms for the sample-and-hold circuit, 12.1 ms for the phased array system, and 4.3 ms for the mixer-first receiver.

## VII. Conclusion

In this paper, a comparison of multiple GNN topologies have been presented for the classification of analog circuits into a multilevel hierarchy using a library-based primitive matching and a GNN-based machine learning method. The GNN-based approach can handle different design topologies of the same sub-blocks and is demonstrated on a variety of testcases. The method is more scalable than prior approaches and shows success in classifying circuits into sub-blocks and creating circuit hierarchy trees. This approach can be used to guide optimization steps such as circuit layout, as demonstrated in the case of a switched-capacitor filter. This approach can be extended to a wide variety of circuit blocks, and the primary effort required for this extension is the laborious task of curating the training set for these new circuit blocks. This can be a topic for future work.

## References

[1] J. Scheible and J. Lienig, "Automation of analog IC layout: Challenges and solutions," in *Proceedings of the International Symposium on Physical Design*, 2015, pp. 33–40.
[2] R. A. Rutenbar, "Design automation for analog: The next generation of tool challenges," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 458–460.
[3] B. Razavi, *Design of Analog CMOS Integrated Circuits*, 1st ed. New York, NY, USA: McGraw-Hill, Inc., 2001.
[4] T. Massier, H. Graeb, and U. Schlichtmann, "The sizing rules method for CMOS and bipolar analog integrated circuit synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 12, pp. 2209–2222, 2008.
[5] M. Meissner and L. Hedric, "FEATS: Framework for explorative analog topology synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 213–226, 2015.
[6] R. Harjani, R. A. Rutenbar, and L. R. Carley, "A prototype framework for knowledge-based analog circuit synthesis," in *Proceedings of the ACM/IEEE Design Automation Conference*, 1987, pp. 42–49.
[7] P.-H. Wu, M. P.-H. Lin, T.-C. Chen, C.-F. Yeh, X. Li, and T.-Y. Ho, "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 199–212, 2015.
[8] K. Settaluri and E. Fallon, "Fully automated analog sub-circuit clustering with graph convolutional neural networks," in *Proceedings of the Design, Automation & Test in Europe*, 2020, pp. 1714–1715.
[9] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," in *IEEE Data Engineering Bulletin*, 2017.
[10] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.
[11] T. Kipf, "Semi-supervised classification with graph convolutional networks," *arXiv:1609.02907*, 2017.
[12] 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 *Proceedings of the Design, Automation & Test in Europe*, 2020, pp. 55–60.
[13] H. Wilson, "Heterogeneous GraphSAGE (HinSAGE)," https://github.com/stellargraph/stellargraph, 2020.
[14] A. K. Sharma, M. Madhusudan, S. M. Burns, P. Mukherjee, S. Yaldiz, R. Harjani, and S. S. Sapatnekar, "Common-centroid layouts for analog circuits: Advantages and limitations," in *Proceedings of the Design, Automation & Test in Europe*, 2021, pp. 1224–1229.
[15] N. Karmokar, M. Madhusudan, A. K. Sharma, R. Harjani, M. P.-H. Lin, and S. S. Sapatnekar, "Common-Centroid Layout for Active and Passive Devices: A Review and the Road Ahead," in *Proceedings of the Asia-South Pacific Design Automation Conference*, 2022.
[16] Q. Zhang, https://github.com/USCPOSH/AMS_KGD, 2019.
[17] H. Li, F. Jiao, and A. Doboli, "Analog circuit topological feature extraction with unsupervised learning of new sub-structures," *Proceedings of the Design, Automation & Test in Europe*, pp. 1509–1512, 2016.

[18] G.-H. Liou, S.-h. Wang, Y.-y. Su, and M. P.-h. Lin, "Classifying analog and digital circuits with machine learning techniques toward mixed-signal design automation," *Proceedings of the International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design*, vol. 15, pp. 173–176, 2018.

[19] H. Ren, G. F. Kokai, W. J. Turner, and T.-S. Ku, "ParaGraph: Layout parasitics and device parameter prediction using graph neural networks," in *Proceedings of the ACM/IEEE Design Automation Conference*, 2020, pp. 1–6.

[20] K. Kunal, M. Madhusudan, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, and S. S. Sapatnekar, "ALIGN: Open-source analog layout automation from the ground up," in *Proceedings of the ACM/IEEE Design Automation Conference*, 2019, pp. 77.1–77.4.

[21] T. Dhar, K. Kunal, Y. Li, M. Madhusudan, J. Poojary, A. K. Sharma, W. Xu, S. M. Burns, R. Harjani, J. Hu, D. A. Kirkpatrick, P. Mukherjee, S. S. Sapatnekar, and S. Yaldiz, "ALIGN: A system for automating analog layout," *IEEE Design & Test*, pp. 8–18, 2021.

[22] B. Xu, K. Zhu, M. Liu, Y. Lin, S. Li, X. Tang, N. Sun, and D. Z. Pan, "MAGICAL: Toward fully automated analog IC layout leveraging human and machine intelligence," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2019, pp. 1–8.

[23] K. Kunal, J. Poojary, T. Dhar, M. Madhusudan, R. Harjani, and S. S. Sapatnekar, "A general approach for identifying hierarchical symmetry constraints for analog circuit layout," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2020, pp. 1–8.

[24] 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 *Proceedings of the Asia-South Pacific Design Automation Conference*, 2020, pp. 193–198.

[25] 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 *Proceedings of the Asia-South Pacific Design Automation Conference*, 2021, pp. 152–157.

[26] 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 *Proceedings of the ACM/IEEE Design Automation Conference*, 2021, pp. 1243–1248.

[27] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1025–1035.

[28] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Li, and Y. Bengio, "Graph attention networks," *arXiv:1710.10903*, 2018.

[29] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. vanden Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *Lecture Notes in Computer Science*, 2018, pp. 593–607.

[30] D. Busbridge, D. Sherburn, P. Cavallo, and N. Y. Hammerla, "Relational graph attention networks," *arXiv:1904.05811*, 2019.

[31] T. Dhar, J. Poojary, Y. Li, K. Kunal, M. Madhusudan, A. K. Sharma, S. D. Manasi, J. Hu, R. Harjani, and S. S. Sapatnekar, "Fast and efficient constraint evaluation of analog layout using machine learning models," in *Proceedings of the Asia-South Pacific Design Automation Conference*, 2021, pp. 158–163.

[32] M. Eick, M. Strasser, H. Graeb, and U. Schlichtmann, "Automatic generation of hierarchical placement rules for analog integrated circuits," in *Proceedings of the International Symposium on Physical Design*, 2010, pp. 47–54.

[33] M. P.-H. Lin, H.-Y. Chi, A. Patyal, Z.-Y. Liu, J.-J. Zhao, C.-N. J. Liu, and H.-M. Chen, "Achieving analog layout integrity through learning and migration invited talk," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, 2020, pp. 1–8.

[34] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman and Company, 1979.

[35] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[36] S. S. Sapatnekar, *Timing*. Boston, MA, USA: Springer, 2004.

[37] "ALIGN: Analog layout, intelligently generated from netlists," Software repository, accessed October 29, 2022., https://github.com/ALIGN-analoglayout/ALIGN-public.

[38] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectronics Reliability*, vol. 53, pp. 105–115, 2016.

[39] B. Razavi, *RF Microelectronics*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2011.

[40] A. Bevilacqua and A. M. Niknejad, "An ultrawideband CMOS low-noise amplifier for 3.1–10.6-GHz wireless receivers," *IEEE Journal of Solid-State Circuits*, vol. 39, no. 12, pp. 2259–2268, 2004.

[41] A. A. Abidi, "Direct-conversion radio transceivers for digital communications," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 12, pp. 1399–1410, 1995.

[42] M. P. Garde, A. J. Lopez-Martin, and J. Ramirez-Angulo, "Power-efficient class-AB telescopic cascode opamp," *Electronics Letters*, vol. 54, no. 10, pp. 620–622, 2018.

[43] J. Haspeslagh and W. Sansen, "Design techniques for fully differential amplifiers," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, 1988, pp. 12.2/1–12.2/4.

[44] K. Kunal, https://github.com/kkunal1408/GANA_circuit_data, 2022.

[45] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256.

[46] S. Zhang, H. Tong, J. Xu, and R. Maciejewski, "Graph convolutional networks: A comprehensive review," *Computational Social Networks*, vol. 6, no. 11, 2019.

[47] J. Godwin, M. Schaarschmidt, A. Gaunt, A. Sanchez-Gonzalez, Y. Rubanova, P. Velikovi, J. Kirkpatrick, and P. Battaglia, "Simple GNN regularisation for 3D molecular property prediction and beyond," in *Proceedings of the International Conference on Learning Representations*, 2022.

[48] Q. Meng and R. Harjani, "A 4GHz instantaneous bandwidth low squint phased array using sub-harmonic ILO based channelization," in *Proceedings of the European Solid State Circuits Conference*, 2018, pp. 110–113.

[49] J. Poojary and R. Harjani, "A 1-to-3GHz co-channel blocker resistant, spatially and spectrally passive MIMO receiver in 65nm CMOS with +6dBm in-band/in-notch B1dB," in *Proceedings of the IEEE International Solid-State Circuits Conference*, vol. 64, 2021, pp. 96–98.

[50] C. Andrews and A. C. Molnar, "A passive mixer-first receiver with digitally controlled and widely tunable RF interface," *IEEE Journal of Solid-State Circuits*, vol. 45, no. 12, pp. 2696–2708, 2010.