GANA: Graph Convolutional Network Based Automated Netlist Annotation for Analog Circuits

Kishor Kunal[1], Tonmoy Dhar[1], Meghna Madhusudan[1], Jitesh Poojary[1], Arvind Sharma[1], Wenbin Xu[2], Steven M. Burns[3], Jiang Hu[2], Ramesh Harjani[1], Sachin S. Sapatnekar[1]
[1]University of Minnesota, Minneapolis, MN [2]Texas A&M University, College Station, TX [3]Intel Corporation, Hillsboro, OR

***Abstract*—Automated subcircuit identification and annotation enables the creation of hierarchical representations of analog netlists, and can facilitate a variety of design automation tasks such as circuit layout and optimization. Subcircuit identification must navigate the numerous alternative structures that can implement any analog function, but traditional graph-based methods cannot easily identify the large number of such structural variants. The novel approach in this paper is based on the use of a trained graph convolutional neural network (GCN) that identifies netlist elements for circuit blocks at upper levels of the design hierarchy. Structures at lower levels of hierarchy are identified using graph-based algorithms. The proposed recognition scheme organically detects layout constraints, such as symmetry and matching, whose identification is essential for high-quality hierarchical layout. The subcircuit identification method demonstrates a high degree of accuracy over a wide range of analog designs, successfully identifies larger circuits that contain sub-blocks such as OTAs, LNAs, mixers, oscillators, and band-pass filters, and provides hierarchical decompositions of such circuits.**

I. Introduction

Analog layout automation has been an active research topic for several decades, but prior research is largely applicable only to a narrow class of circuits with specific topologies. Real-world circuits appear in a large number of variants even for a single functionality, e.g., between textbooks [1] and research papers, there are well over 100 widely used operational transconductor amplifier (OTA) topologies of various types (e.g., telescopic, folded cascode, Miller-compensated).

A prerequisite for truly generalizable analog automation is to recognize all variants – including those that have not even been designed to date. This capability is crucial, because analog designers exercise such variants as a "secret sauce" for enhanced performance. Prior methods are *library-based* [2], [3], matching a circuit to prespecified templates, and requiring an enumeration of possible topologies in an exhaustive database, or *knowledge-based* [4], [5], 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.

Both 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. The inability of conventional algorithms to replicate this has limited the industrial application of analog design automation.

We seek to build a foundation for a generalizable analog design methodology *with no human in loop* [6]. 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. However, unlike images, a netlist cannot be abstracted into a planar 2D graph, and convolutional neural networks (CNNs) cannot be used directly.

This work is supported in part by the DARPA IDEA program, as part of the ALIGN project, under SPAWAR Contract N660011824048.

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 convolutional networks (GCNs) [7]–[9] 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 vary significantly across circuits and can be handled using graph-based methods. In contrast, the larger blocks (OTAs, low noise amplifiers (LNAs), mixers, etc.) may appear in a number of variants, and can benefit from using trained GCNs, which can handle variants naturally. A GCN learns the rules during training, thus mimicking the human expert.

Prior work on using ML for subcircuit identification includes a compiler-inspired learning approach in [10] 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 [11] 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.

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); or constraint budgeting to each block while meeting system-level constraints. This paper focuses on algorithms for generating circuit hierarchies and presents a use case that illustrates how these hierarchies can service the above tasks.

II. Hierarchical recognition

A. Representing circuits hierarchically

Our recognition scheme creates a hierarchical representation of the netlist, identifying smaller structures composed into larger structures: *(1) Elements*, i.e., transistors (NMOS/PMOS) and passives (resistors, capacitors, inductors), lie at the lowest level. *(2) Primitives*, composed from 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. *(3) 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. 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. The effort reported in this paper goes up to the level of sub-blocks.

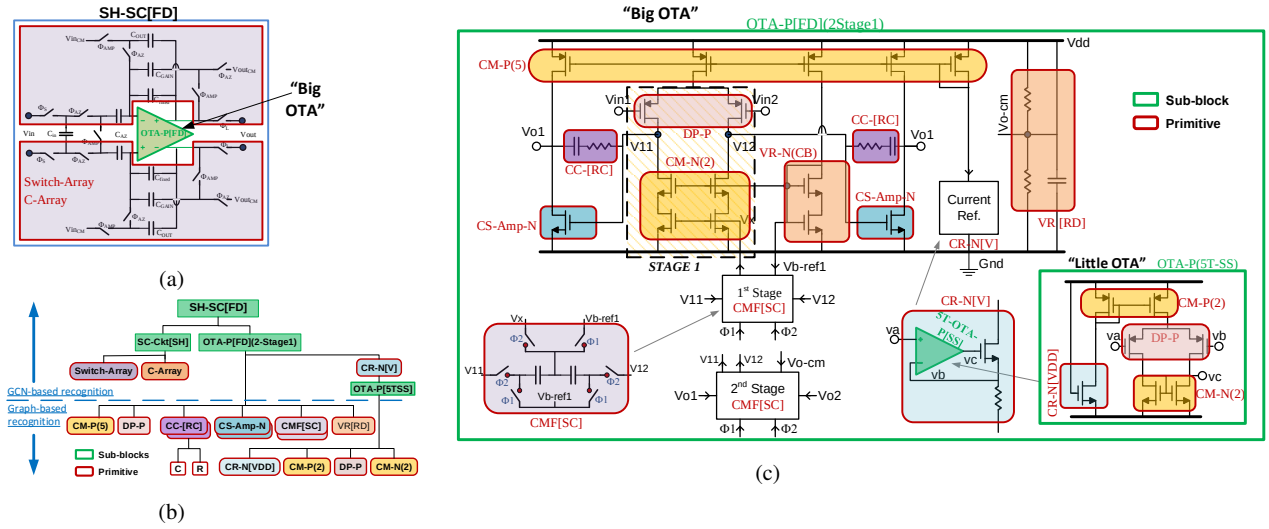


Figure 1: Multiple levels of abstraction in a sample-and-hold schematic.

Fig. 1 decomposes a sample-and-hold circuit at the schematic level and shows the corresponding hierarchy tree. The uppermost level of hierarchy is the sample-and-hold switched-capacitor circuit block (SH-SC). Below this are the switched capacitor circuit and the OTA (shaded in Fig. 1(a)). These are further decomposed into smaller structures (Fig. 1(c)). Note that the decomposition of the high level OTA sub-block (marked as "Big OTA") contains another simpler "Little OTA" sub-block within the current reference, while other sub-blocks are primitives: the PMOS current mirror (CM-P), PMOS differential pair (DP-P), NMOS common-source amplifier (CS-Amp-N), common-mode feedback (CMF), current reference (CR-N), voltage reference (VR), and various passives (R, C, CC-RC).

Our approach builds multiple levels of hierarchy, with potentially multiple level of sub-blocks in the hierarchy tree. Our recognition algorithm processes primitives and sub-blocks hierarchically to recognize an entire circuit of this type, and build its hierarchy tree.

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 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 independent of designer idiosyncrasies, and allows us to integrate design constraints into the recognized blocks in a consistent manner, e.g., it is common to place bias networks in a different hierarchy than an OTA, but this splits current mirror functionality across blocks, making recognition difficult. Beyond consistency, it should be noted that hierarchies that are logical to the netlist designer may not be the most logical for annotation or layout optimization, e.g., passives and switches in a DAC logically lie within the same hierarchy, but the passives should be grouped together in a common-centroid layout, separately from the noisy switches. Preprocessing also identifies netlist features that help performance but do not affect functionality (and can be disregarded during recognition), e.g., parallel transistors for sizing, series transistors for large transistor lengths, dummies, decaps.

GCN-based recognition: We create a graph representation of the flattened netlist and *identify sub-blocks* using a GCN-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 (e.g., a net that is the output of one sub-block and the input of another) to belong to multiple sub-blocks. At this level of abstraction there is substantial scope for variation in the way designers build circuits. The GCN 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 GCN: 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 IV).

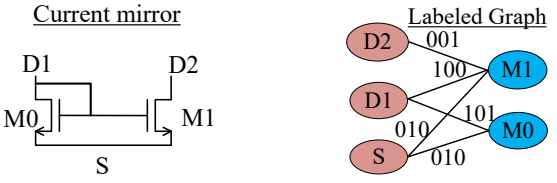
Post-processing: After primitive annotation, we use postprocessing to determine which primitives are an integral part of a specific unit, and which are auxiliary to the unit (e.g., input buffers) and can be separated. This step is necessary because any neural network based approach can never be 100% 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 V how simple post-processing can enhance recognition accuracy to very high levels (100% for all of our testcases).

Each recognition step is helpful 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

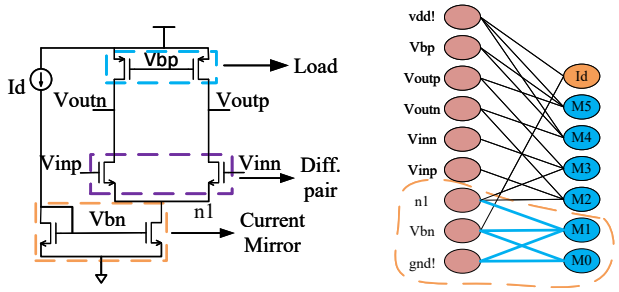
As in [12], we represent an element-level circuit netlist as an undirected bipartite graph $G(V, E)$. The set of vertices V can be partitioned into two subsets, V_e, corresponding to the elements (transistors/passives) in the netlist, and V_n, the set of nets. The edge set E consists of edges between the vertex corresponding to each element to the nets connected to its terminals. There are no edges between elements of V_e or elements of V_n, and therefore the graph is bipartite.

Fig. 2 illustrates an example of a current mirror primitive with two transistors, M0 and M1, and the graph corresponding to this structure, where blue vertices represent V_e and pink vertices represent V_n. Each edge connected to a transistor is assigned a three-bit label, $l_g l_s l_d$, where $l_g = 1$ if the edge from the transistor vertex connects to the net vertex through its gate, and is 0 otherwise; similarly, l_s (l_d) are 1 if the transistor connects to the net through its source (drain), and



(a) (b)

Figure 2: (a) An NMOS current mirror primitive, CM-N(2), with two transistors. (b) Its representation as a bipartite graph.



(a) (b)

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).

is 0 otherwise. Edges connected to passives have no labels. Similar approaches can be used for larger structures: Fig. 3 shows a differential OTA and its bipartite graph (without edge labels, for simplicity).

III. Graph Convolutional Networks

A GCN can achieve good separation between the feature representations of vertices in a graph by using the graph structure. The notion of convolutional neural networks (CNNs) is well established in image recognition. An image is an arrangement of pixels that can be represented by a graph where the vertices correspond to pixels connected to neighboring pixel vertices by an edge. The graph therefore has a very local and structured set of connections. The natural embedding of the graph in the plane is the image itself: a filter can identify features in an image simply by performing a convolution over repeated windows of the image.

General graphs have no unique embedding. A GCN performs convolutions that are independent of the embedding of the graph in the plane. Various types of GCNs have been proposed [7]–[9], [13], but they all share a framework that requires three fundamental steps: (i) the application of localized convolutional filters on graphs, (ii) a graph coarsening procedure that groups together similar vertices and (iii) a graph pooling operation for graph reduction. GCN techniques primarily differ in the nature of the filter that is used for convolution.

A. Spectral GCN filters

A major class of GCNs is based on spectral methods [8], [9], which have been found to be extremely effective and therefore form the basis of the GCN used in our work. Spectral operators are independent of the graph embedding. Spectral graph theory creates a convolution operator using Fourier analysis. A spectral representation in the Fourier space of a graph is enabled through the graph Laplacian representation. The Laplacian $L \in R^{n\times n}$ of an unweighted graph $G(V, E)$ with n vertices is often defined as $D - A$ where $A \in R^{n\times n}$ is the adjacency matrix of the graph and $D \in R^{n\times n}$ is a diagonal matrix whose diagonal entry corresponds to the degrees of all vertices, i.e., the row sums of the adjacency matrix. We will work with the normalized Laplacian representation,

$$L = I - D^{-1/2}AD^{-1/2} \tag{1}$$

The matrix L is symmetric, real, and positive definite (i.e., it has real nonnegative eigenvalues, and these eigenvalues are interpreted as the frequencies of the graph).

The normalized graph Laplacian can be eigendecomposed as $L = U\Lambda U^T$, with an eigenvalue matrix $\Lambda \in R^{n\times n}$ and a Fourier basis U corresponding to the eigenvector matrix. The graph Fourier transform of a signal $x \in R^n$ is given by $\tilde{x} = U^T x \in R^n$, and its inverse as $x = U\tilde{x} \in R^n$. This transform enables operations such as graph filtering, which are vital to the definition of a GCN.

In this work, we use the approach proposed by [8], which creates spectral filters around a vertex that function within a region of radius K of (i.e., up to K edges away from) the vertex. A convolution operator on the graph is defined as

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = Ug_\theta(\Lambda)U^T x \tag{2}$$

where g_θ is a filtering operator that acts on an input signal x to produce an output signal y. In this case, the signal corresponds to a region of the graph around a specific vertex.

However, in general, a spectral domain filter may not be localized and could be computationally expensive to compute, with $O(n^2)$ operations due to the multiplication with U, even if the eigendecomposition of L is known. Defferrard [8] proposed a solution to this problem that parameterizes $g_\theta(L)$ as a polynomial Chebyshev expansion, which truncates the filter expansion to order of $K - 1$ as:

$$g_\theta(\Lambda) = \textstyle\sum_{k=0}^{K-1} \theta_k T_k(\hat{\Lambda}) \tag{3}$$

Here $\hat{\Lambda} = 2\Lambda/\lambda_{max} - I_n$ scales and translates the eigenvalues by the largest eigenvalue, λ_{max}, so that $|\hat{\Lambda}_{ii}| \leq 1$; $\theta \in R^K$ is a vector of coefficients, and $T_k(\hat{\Lambda}) \in R^{n\times n}$ is a Chebyshev polynomial of order k. The Chebyshev polynomials are defined as $T_0(x) = 1$, $T_1(x) = x$,

$$T_k(x) = 2\ x\ T_{k-1}(x) - T_{k-2}(x), \tag{4}$$

Given the K top eigenvalues of L (computed inexpensively using the Lanczos algorithm), for a graph of bounded degree where the number of edges is $O(n)$, this polynomial can be evaluated using K multiplications by a sparse L with a cost of $O(Kn) \ll O(n^2)$.

As a result, Equation (2), which defines the convolution operator at a graph, can be written as:

$$y = g_\theta(L)x = \textstyle\sum_{k=0}^{K-1} \theta_k T_k(\hat{L})x \tag{5}$$

where $\hat{L}$ is defined similarly as $\hat{\Lambda}$. The truncation to K terms restricts the filter to operate at most K hops away from each vertex, thus creating an inexpensive and localized filter.

B. GCN topology for circuit recognition

For the filter is defined in Equation (5), we construct a GCN that uses multiple convolutional stages consisting of filtering and pool layers. The GCN topology used in this work is shown in Fig. 4: it has two convolutional and two pooling layers, which then feed a fully connected (fc) layer, whose outputs provide the classification results.

The pooling layer combines similar vertices in a graph. Pooling at multiple levels of the GCN can be interpreted as multilevel clustering. Since optimal multilevel clustering is NP-hard, it is typical to use a heuristic method. The GCN used in this work uses the greedy Graclus heuristic, built on top of the Metis algorithm [14] for multilevel clustering [15]. The pooling operator is based on a balanced binary tree that represents each cluster: pooling operations can be performed very efficiently by traversing the tree. The final layer is a fully connected layer of size 512 along with `softmax` function for classification.

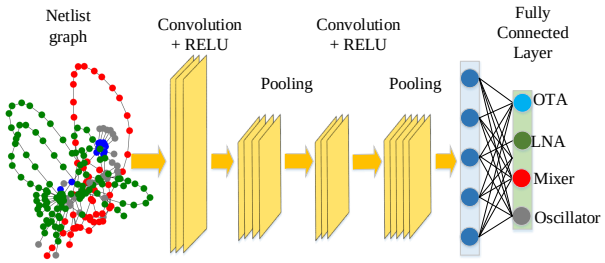


Figure 4: A GCN topology used for circuit recognition.

C. Sub-block constraint annotation

The sub-blocks identified by the GCN 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. 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.

IV. Annotating primitives

We populate a library of 21 basic primitives that are building blocks for larger sub-blocks. 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(a) 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(a) [recall that to reduce clutter, edge labels are omitted from Fig. 3].

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 [16]. We use VF2, an established graph matching algorithm [17]. This method has a worst-case complexity of $\Theta(n!n)$ for the general subgraph isomorphism problem, where n is the number of vertices, but 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. 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. Experimental results

Our flow is implemented in Python 3.7.0 including `TensorFlow` for GCN training/testing, and `scikit` for sparse matrix computations.

A. GCN training

Training set for the GCN: The training set for the GCN was taken from several sources, including standard textbooks [1], [18] and papers in the literature (e.g., [19]–[22]). 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 I. 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 and oscillators (OSC).

Datasets	# Circuits	# Nodes	# Labels	# Features
OTA_bias	624	32152	2	18
RF_data	608	21886	3	18

Table I: A description of our training dataset.

The input to the GCN is the circuit graph, $G(V, E)$, in the form of an adjacency matrix and an $n \times d$ matrix of features, where d is the number of features. Our implementation associates vertices in the graph with 18 features:

- 12 features that annotate the element type, e.g., whether it is an NMOS/PMOS transistor, resistor, inductor, capacitor, voltage reference, current reference, or a hierarchical block (and if so, the level of hierarchy); whether its value is 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).
- 5 features that denote the type of net – input, output, bias signal, supply net, ground net – associated with an edge in the graph.
- 1 feature that describes the edges incident on a transistor vertex (Section II-C).

The task of the GCN 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.

Architecture and hyperparameter optimization: The training step uses standard regularization techniques: batch normalization, which ensures that all input quantities are in the same numerical range so that no one input dominates the others, and dropout, which randomly ignores a set of neurons during training to avoid overfitting. The input data is split into an 80% : 20% ratio for the training set and validation set, and a random search method is used to optimize hyperparameters such as the learning rate, regularization, decay rate, and filter size.

Choosing the number of layers: Several versions of the GCN architecture in Fig. 4 were explored to determine a good topology. For the activation function, the use of the ReLU and tanh functions was examined across all layers, and we empirically found that ReLU provides consistently better results.

We observed that in going from one layer to two, there is a noticeable improvement in accuracy, but moving to three layers reduces the accuracy. This is due to the fact that each graph convolution layer acts as an approximate aggregation of characteristics of neighboring nodes, and adding too many layers over-smoothens the output features. Although some recent methods [13] can use deeper models they require larger training data – but large training sets of analog circuit are not easily found in the public domain. Therefore, we choose a two-layer GCN for our implementation, whose training accuracy is 88.89%, with a variance of 1.71%, for the OTA_bias dataset, and 83.86%, with a variance of 1.98%, on the RF_data dataset. This choice corresponds to the best accuracy as well as the smallest variance. While this accuracy is not quite 100%, as stated in Section II-B, it is supplemented by the effective use of post-processing, which we will be detailed shortly. The training time is under 2 hours for each dataset.

Impact of filter size: Since the value of K in Eq. (5) indicates the number of hops away from a vertex that a GCN filter can "see," an appropriate choice of filter depends on the radius of the graph structure that corresponds to each class. Multiple configurations of filter sizes were tested for the two-layer GCN. Fig. 5 shows that larger filters provide improved accuracy but this is achieved at a cost of increased runtimes. The accuracy flattens out beyond a filter size of about 30. A five-fold cross validation is used to reduce the sensitivity to data partitioning, and a filter size of 32 was chosen.

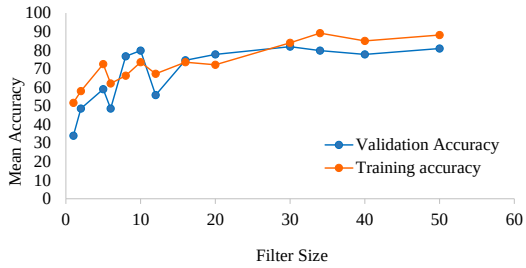


Figure 5: Two-layer GCN accuracy as a function of filter size.

Postprocessing: Rather than placing the full burden of recognition on the GCN, we use an engineering approach that allows the GCN to perform a large fraction of the recognition task and then uses a set of simple postprocessing heuristics to complete the annotation. A simple set of postprocessing steps is shown to bring the recognition accuracy to 100% for all evaluated circuits. Our postprocessing operation uses two classes of heuristics after GCN classification:

Postprocessing I involves graph-based heuristics in which we associate the nodes that belong to the same channel-connected component (CCC)[1] with a sub-block. Next, we identify all primitives within a CCC using graph-based approaches as described in Section IV for extracting primitives within a sub-block. All primitives in a CCC 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) is separated and listed as a stand-alone primitive in the hierarchy tree.

Postprocessing II is related to knowledge that is specific to circuit classes, and is based on information about connections to input/output ports. For example, LNA and mixers may have structurally similar topologies, but can be differentiated because an LNA has an antenna input, while a mixer has an oscillating input. Such information can be provided by the designer as a separate label on the port, or can be inferred from the test bench in the input SPICE netlist.[2]

Together, these heuristics solve all of the misclassifications in the training and test set for the given class of circuits. Note that the Postprocessing I heuristic is the same across all designs, while Postprocessing II requires domain-specific annotation, and may require new rules as new classes of subcircuits are added.

B. Experimental evaluation

To evaluate the annotation procedure, we apply our procedure on four different types of test sets, as summarized in Table II. Note that the switched capacitor filter and phased array testcases are more complex systems that include OTA and RF sub-blocks, respectively, and correspond to hand-crafted circuits provided to us by designers.

Test set	# Circuits	# Nodes	GCN accuracy
OTA_bias	168	9296	90.5%
Switched capacitor filter	1	57	98.2%
RF_data	105	17640	83.64%
Phased array system	1	902	79.8%

Table II: Results of classification on test data

The first test set consists of 168 OTA circuits with different signal and bias configurations, disjoint from the training data. We utilize our GCN-based model to extract the subcircuit components of signal and the bias components of the circuit. The training set for the OTA annotates the nodes as OTA nodes or bias nodes. Though the GCN in itself does not to correctly classify all nodes of the subcircuit, misclassifications are rectified using the postprocessing steps. Specifically, on this dataset, the GCN achieves an accuracy of 90.5% which is raised to 100% after the "Postprocessing I" step.

The second testcase consist of a composite circuit, a switched capacitor filter, with an OTA. This is similar to the sample and hold circuit in Fig. 1(a) and contains 32 devices and 25 nets, including an OTA sub-block and switched capacitors. The telescopic OTA subcircuit is 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. During preprocessing in the first step in Section II-B, dummy devices and stacking of resistors, capacitors, and transistors are ignored (for recognition purposes only). Using the GCN alone we achieve an accuracy of 56/57 in identifying OTA and bias circuit nodes. The misclassified vertices belong to the OTA interconnect ports, and all nodes (100%) are correctly classified after "Postprocessing I."

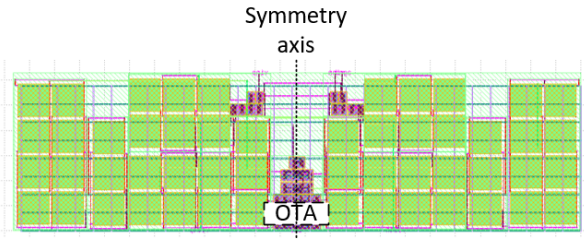


Figure 6: Layout of the filter based on the extracted hierarchy.

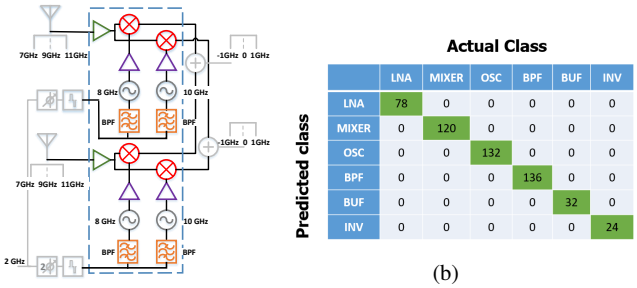
To illustrate a use case for circuit annotation, we take the results of circuit recognition to pass the design through a custom layout generator for the ASAP7 [24] PDK. The hierarchies identified by our algorithm are used by the layout tool to construct layouts for primitives, which are assembled into layouts for larger blocks. For example, by clustering

[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 [23].

[2]Since these are deterministic features (without variants), there is no value to building them into a training set: this would require a larger training set, where *all* correspondingly labeled items would have this feature.

the OTA circuit 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 symmetry and proximity constraints detected at the primitive level are propagated to other levels of hierarchy (Section III-C), creating a common axis of symmetry for the entire layout. The generated layout is shown in Fig. 6. The annotation helps in identifying and clustering the central annotated OTA sub-block shown in the figure. The other elements of the layout correspond to the switched capacitors, and are shown as the large capacitor arrays (green rectangles) and small switches (shown in blue) next to the capacitor array.

The third test set is associated with RF circuits and consists of 105 different datasets that combine various LNAs, mixers, and oscillators in a receiver [19]. The architectures that combine LNAs, mixers, and oscillators used in this set are similar to some architectures in the RF_data training set, but the precise circuits are distinct, with no repetitions from the training set. We use our trained model to identify nodes associated with LNA, mixer, and oscillator subcircuits. The GCN achieves an accuracy of 83.64%, which goes to 89.24% after "Postprocessing I." After "Postprocessing II," this rises to 100% and all nodes are correctly identified: here, the antenna at the LNA port and the oscillating signal at the oscillator port are used to correct LNA/oscillator misclassifications.



(a)

Actual Class

Predicted class	LNA	MIXER	OSC	BPF	BUF	INV
LNA	78	0	0	0	0	0
MIXER	0	120	0	0	0	0
OSC	0	0	132	0	0	0
BPF	0	0	0	136	0	0
BUF	0	0	0	0	32	0
INV	0	0	0	0	0	24

(b)

Figure 7: (a) Phased array system [25] and (b) results of GCN after postprocessing, showing the correctness of vertex classification.

The fourth and largest testcase consists of a phased array system [25], illustrated in Fig. 7(a), containing a mixer (red), LNA (green), BPF (orange), oscillator (gray), VCO buffer (BUF) and inverter-based amplifier (INV) (violet) sub-blocks. The graph for the input netlist has 902 vertices (522 devices + 380 nets). The GCN based classification identifies nodes belonging to LNA, mixer, and oscillator and passes these results through postprocessing. After "Postprocessing I," the BPF is identified as a combination of an oscillator with two input transistors. INV and BUF primitives are identified and a separate hierarchy is created for them which boosts the accuracy to 87.3%. During "Postprocessing II," which uses an antenna label at LNA input and oscillating input for mixer, all nodes are identified correctly. At this point, the classification result after post-processing is shown in Fig. 7(b): all 522 devices (100%) are classified correctly.

Our annotation scheme is fast, and is dominated by the runtime of the GCN. We report numbers on the more complex circuits on an Ubuntu host with an Intel Core i7 processor @2.6GHz with 8 cores and 32GB RAM: the procedure takes 135s for the switched capacitor filter circuit, and 514s for the phased array system. The postprocessing step requires less than 30s.

VI. Conclusion

In this paper, a novel approach for the classification of analog circuits into a multilevel hierarchy using a library-based primitive matching and a GCN-based machine learning method, is presented. The GCN-based approach can handle different design topologies of the same sub-blocks and is demonstrated on a variety of testcases, including two hand-crafted circuits. The method is more scalable than prior approches and shows success in classifying circuits into sub-blocks and creating circuit hiearchy trees. This can be used to guide optimization steps such as circuit layout, as demonstrated in the case of a switched-capacitor filter. Although the GCN-based approach does not provide complete accuracy, it can be improved using a more diverse training set. Even for a limited training set, postprocessing provides 100% accuracy in all 275 test cases evaluated here.

References

[1] B. Razavi, *Design of Analog CMOS Integrated Circuits*. New York, NY, USA: McGraw-Hill, Inc., 1st ed., 2001.

[2] T. Massier, *et al.*, "The sizing rules method for CMOS and bipolar analog integrated circuit synthesis," *IEEE T. Comput. Aid D.*, vol. 27, pp. 2209–2222, Dec. 2008.

[3] M. Meissner and L. Hedric, "FEATS: Framework for explorative analog topology synthesis," *IEEE T. Comput. Aid D.*, vol. 34, no. 2, pp. 213–226, 2015.

[4] R. Harjani, *et al.*, "A prototype framework for knowledge-based analog circuit synthesis," in *Proc. DAC*, pp. 42–49, 1987.

[5] P.-H. Wu, *et al.*, "A novel analog physical synthesis methodology integrating existent design expertise," *IEEE T. Comput. Aid D.*, vol. 34, no. 2, pp. 199–212, 2015.

[6] K. Kunal, *et al.*, "ALIGN: Open-source analog layout automation from the ground up," in *Proc. DAC*, pp. 77.1–77.4, 2019.

[7] W. L. Hamilton, *et al.*, "Representation learning on graphs: Methods and applications," in *IEEE Data Eng. Bull.*, 2017.

[8] M. Defferrard, *et al.*, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. NeurIPS*, pp. 3844–3852, 2016.

[9] T. Kipf, "Semi-supervised classification with graph convolutional networks," in *Proc. ICLR*, 2017.

[10] H. Li, *et al.*, "Analog circuit topological feature extraction with unsupervised learning of new sub-structures," *Proc. DATE*, pp. 1509–1512, 2016.

[11] G.-H. Liou, *et al.*, "Classifying analog and digital circuits with machine learning techniques toward mixed-signal design automation," *Proc. SMACD*, vol. 15, pp. 173–176, 2018.

[12] M. Ohlrich, *et al.*, "SubGemini: Identifying subcircuits using a fast subgraph algorithm," in *Proc. DAC*, pp. 31–37, 1993.

[13] R. Ying, *et al.*, "Graph convolutional neural networks for web-scale recommender systems," in *Proc. KDD*, pp. 974–983, 2018.

[14] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[15] I. Dhillon, *et al.*, "Weighted graph cuts without eigenvectors: A multilevel approach," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, pp. 1944–1957, Nov. 2007.

[16] 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.

[17] L. P. Cordella, *et al.*, "A (sub)graph isomorphism algorithm for matching large graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, pp. 1367–1372, Oct. 2004.

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

[19] 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.

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

[21] M. P. Garde, *et al.*, "Power-efficient class-AB telescopic cascode opamp," *Electronics Letters*, vol. 54, no. 10, pp. 620–622, 2018.

[22] J. Haspeslagh and W. Sansen, "Design techniques for fully differential amplifiers," in *Proc. CICC*, pp. 12.2/1–12.2/4, 1988.

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

[24] L. T. Clark, *et al.*, "ASAP7: A 7-nm FinFET predictive process design kit," *Microelectron. Reliab.*, vol. 53, pp. 105–115, 2016.

[25] Q. Meng and R. Harjani, "A 4GHz instantaneous bandwidth low squint phased array using sub-harmonic ILO based channelization," in *Proc. ESSCIRC*, pp. 110–113, 2018.