Machine Learning for Analog Layout Automation

ADITYA GAWANDE[*] and DEEXITHA VATTIVELLA[*], Indian Institute of Technology Hyderabad, India

Analog ICs form an integral part of all systems around us. This includes the ICs which convert digital data to analog and vice versa. Compared to digital ICs, where the process node plays an important role in performance, analog ICs are still fabricated at older nodes, meaning circuit architecture is the main driver of performance improvements. Designing them is a highly manual endeavour, usually requiring 5-10 members working 1-2 years. While there have been efforts to automate this, the design space cannot be captured by methods like non-linear programming, due to the design space complexity and high cost of iteration. Research has hence moved to ML-based methods, but getting training data and ground truth values, especially for large circuits, is expensive, with results from each iteration taking days worth of simulation time. Current state of the art automation techniques overcome these limitations by breaking the problem into smaller chunks and using specifically designed frameworks for each of them. The need for better analog IC automation tools has been recognised by DARPA, which funded the Magical EDA framework, with an aim of reducing development time for analog ICs to 24 hours, with minimal human intervention. The Magical framework is highly flexible and research has been done to improve its individual components, like placement engine, constraint generation, placement quality prediction, etc. While this paper is about analog ICs, the methods discussed here have the potential to contribute in solving other problems as well, where design time is high, training data is scarce, and iteration cost is high.

Additional Key Words and Phrases: Analog, Layout, BNN, GNN, NAS, CTL, Automation

ACM Reference Format:
Aditya Gawande and Deexitha Vattivella. 2025. Machine Learning for Analog Layout Automation. 1, 1 (May 2025), 18 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Analog ICs are used extensively in almost all electronics. For any application where a battery is present, voltage conversions are required. This includes all phones, laptops, etc where clean, stable voltage needs to be provided to the processor. These are done by Power Management ICs (PMICs). These ICs also convert the voltage coming from the charger to a suitable level to charge the battery. Here, efficiency of voltage conversion is very important and over the decades, improvements in them have allowed faster charging and more efficient battery-powered devices. Another field where analog ICs are used is Analog to Digital conversion and vice-versa. All the signals in the world are analog. They need to be converted to the digital domain, if they are to be processed by a CPU/GPU, etc. This conversion needs to be accurate and fast, so as to not distort the signal and add latency to the processing. Additionally, digital data needs to be converted into analog, like displaying an image on a screen, or playing music on a speaker. This is done by ICs called ADCs and DACs,

[*]Both authors contributed equally to this research.

Authors' Contact Information: Aditya Gawande, ee22btech11202@iith.ac.in; Deexitha Vattivella, es22btech11036@iith.ac.in, Indian Institute of Technology Hyderabad, Sangareddy, Telangana, India.

respectively. Over the decades, these ICs have been developed to be higher resolution and work at a faster speed, which has allowed us to get higher fidelity video and audio.

Both digital ICs, like CPUs/GPUs and analog ICs use the same CMOS transistor technology, but there is a huge difference in how the transistor is used by them, which changes how products are developed, as well. Building block of digital ICs is the NAND gate. It is a physical structure made up of transistors which mimicks the boolean operation of Not-AND. Here, the transistors work towards discharging a capacitor to 0V or charging it to 2V or so. Transistor is modelled as a simple switch, and the circuits work even if the capacitor gets discharged to 0.2V or charged to 1.8V. In analog ICs, the transistor can work as a number of different things, like amplifier, current mirror, switch, capacitor, etc. Here, depending on its use, it faces different constraints for operating conditions and environment. For example, if a transistor acts like a switch, it must pass the voltage with an error of less than 50uV. Or, if a transistor acts like a current mirror, its operating conditions must be matched pretty well, otherwise it would not mirror current correctly. In both of these cases, if there is a wire carrying clock signal too close to these transistors, we would see some unwanted noise in the output, which may lead to oscillations or instability in the entire circuit. This means the designers needs to very careful when choosing how to place the transistors and connect them on the silicon (here-after referred to as layout).

About the overall development cycles, digital ICs have billions of transistors while analog ICs will have just tens of thousands of them. The team sizes are also much smaller for analog, where 5-10 designers+layouts engineers can make an IC in about a year or two. The development work typically includes designing and simulating transistor-based circuits. Unlike digital ICs, these are designed at a process node of 65nm or 180nm. Smaller transistors do not benefit these designs as much as CPUs/GPUs, and on the contrary, could be worse than larger transistors. There are usually dozens of transistor-level circuits in an ICs, each of which is made specifically for that use-case. These are usually designed in a hierarchical manner, with the lowest level being current mirrors, operational transconductance amplifier, bandgap reference, etc and higher levels being reference voltage generation, clock generation, phase-locked loops, etc. While layout automation has been the norm for digital ICs for a long time, the same is not true for analog ICs, as layout heavily affects performance. This is why development teams spend significant amount of time and effort for layout. For a given design, layout generated by an expert vs. that generated by a digital layout tool can perform wildly differently, with offset voltage (a major parameter for comparators) being 0.3mV and 2.2mV respectively.[4]

This paper have been divided into 7 sections. Section 1 introduces analog ICs and why their development time is high. Section 2 is sub-divided into two parts, manual design flow and Magical design flow, where each step of the layout process is explained briefly. Sections 3-6 focus on the four papers that have been summarised in this term paper. The first three are ML-based tools which improve upon the placement engine, constraint generator and placement quality prediction, which are present in Magical or suggested to be a part of Magical, with Layout-aware sizing focusing on speeding up convergence of problems which need extracted simulations. Finally, section 7 concludes the paper.

2 Design Flow

While both design flows achieve the same outcome, the steps are slightly different. The manual design flow has been established over the course of past 3-4 decades, with manufacturability, error tracability and performance being the key drivers of innovation. Magical framework is a first of its kind tool to go from circuit netlist to GDS without any human intervention.[3] But while it generates GDS (standard format for layout), the performance is still less than expert layout, hence research is going on for replacing Magical modules with better ones.

2.1 Manual

Design of an analog IC starts from a spec sheet, which decides which architecture can be used. Architecture decides the range of performance that can possibly be achieved. Designers decide the circuit parameters based on the process node, spec sheet requirements and power limit. Circuits are designed in a hierarchical manner, where the lowest level may be a comparator or an OTA, while the upper blocks may be a reference generation block or clock generation, etc. These circuits are described in the form of text files called netlists. Once the circuits has been finalized and simulated to be meeting the requirements, layout is started. Before starting any step, the layout engineers usually understand the input and output pins, the noise/matching requirements (how strict they are) and the area constraint, going from top to bottom in the hierarchy. A floorplan is generated with an estimate of how much area each block will take, starting from the bottom to top in the hierarchy. Location and orientation of each transistor (called devices), resistor and capacitor on the silicon needs to be decided. This is called placement. Once all devices are placed, they need to be connected with a metal or polysilicon wire. This is called routing. This is done manually by the layout engineer as well. A few finishing touches like adding pins, metal fills are done using some standard automation tools and we get the final GDS of the design. This is the file which needs to be sent for manufacturing to a fab like TSMC, SCL, or UMC.

Before sending it, the GDS is converted back to a circuit representation called an extracted netlist. This is different from the initial netlist, as in, it contains all the non-idealities that have been introduced due to the layout (called parasitics). These parasitics affect the performance adversely and may sometimes lead to circuits not working at all, due to unintended capacitances and other effects introduced by the layout. Based on the severity of the issues seen and available area, the layout gets revised a few times before it meets the specs, this time with extracted simulations. It is important to note that both simulation time and debug time for extracted simulations is much higher than pre-layout simulations. Simulation time is higher because the parasitic interactions need to be accounted by the simulator as well. Because extracted simulations are done at the highest hierarchy, searching for issues is a tedious and time-taking task, especially, as the issues could be arising due to unexpected interactions between circuits. Due to the heavy effect of layout quality on final performance, and the heavily manual nature of almost all steps, layout takes a significant amount of time during the development cycle.

2.2 Magical

Magical takes the circuit netist as an input and outputs a GDS from it, with no other inputs required from the user. This is done because the Magical framework contains modules for each step of the layout flow, although all of them are rudimentary implementations.

Constraint extraction. Similar to manual layout, magical first analyses the input netlist and tries to find out any symmetry requirements. These symmetry requirements could be between devices or between different modules at a higher level of hierarchy. For device symmetry, this is done through pre-defined pattern libraries and graph isomorphism algorithms. System level symmetry is detected through spectral graph analysis on selective modules, which are used repeatedly. Once the symmetry between different devices and modules is known, they can be used to make a constraint file which is used by the further steps to make informed decisions about placement and routing.

Device generation. All layout flow steps happen in tools called EDA. They generate a layout file, which is proprietry to the tool. This file is like an image representation of the layout, with all of the devices on one layer and wires on rest of the layers. Before we start working on the layout, we need to add all the devices from the circuit netlist to this. This is usually done quite easily with EDA tools. Magical handles this a bit differently, as it needs to generate them in a format usable by its subsequent placement and routing tools. One option provided by Magical is that resistor/capacitor arrays or even standard cells can be considered as an indivisible unit, which wil be used as is by the placement engine. This can provide a higher amount of control to the user to make sure that the critical blocks are placed/routed correctly. Abstracting away standard cells makes the entire process much faster, as they have optimal layouts available for them.

Placement Engine. Due to the manufacturer's capabilities, there is a limit to the minimum distance possible between two devices. There are constaints on the minimum width of wire as well. As these are decided solely by the fab, these do not change for a given process design kit (PDK), and must be taken into account for every placement and routing done in that PDK. These things are checked in the manual layout flow just before generating the GDS. The placement engine in magical uses non-linear objective functions to reduce area and wirelength, while using the symmetry constraints as well. It has a legalization algorithm as well, which makes sure that the design rules and constraints are followed.

Routing Engine. A* path finding algorithm with symmetry constraints is used. This divides the entire 3D space into a grid based on the minimum wire width of the lowest layer. Then the entire problem boils down to finding a path through a 3D space, where the 3rd dimension is the upper metal layers (of which there can be 5-10). It is ensured that the constraints and design rules are followed. This method is very similar to digital routing engines, except for the additional symmetry and matching constraints. The final result is exported as a GDS file. Rest of the steps of extracted simulations are same as manual layout.

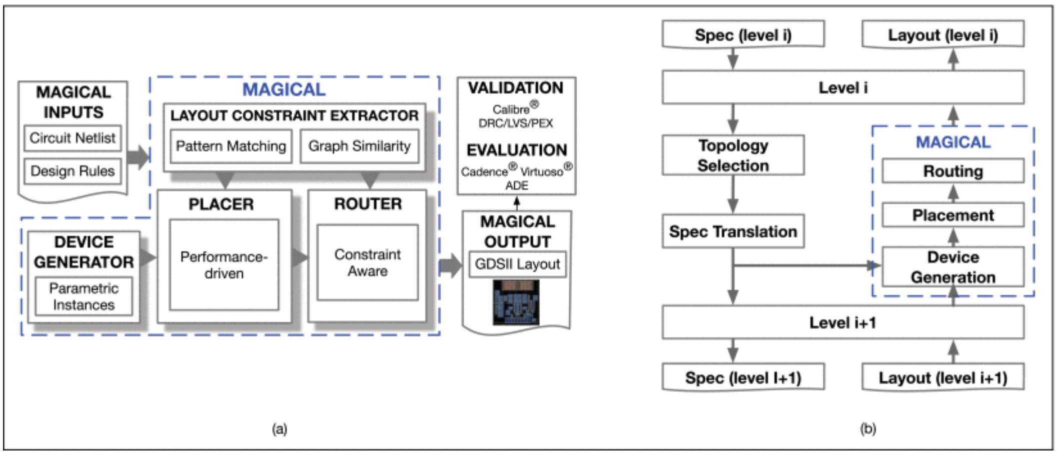


Fig. 1. MAGICAL framework. (a) MAGICAL submodules. (b) MAGICAL hierarchical flow, taken from [3]

While the Magical flow does work and it does generate layout for Analog ICs within minutes, the issue is with performance, which is a crucial factor depending heavily on layout (in state of the art designs). And due to the rudimentary nature of almost all of the blocks, the layouts generated through magical flow do not compete with those made by experts.

ML can be used to bridge the gap here, but some obvious paths are not feasible in this specific case - 1. RL with extracted simulation results - This is not feasible, as extracted full chip simulations take tens to hundreds of CPU-hours. Also, even for smaller designs, it is not guaranteed that the layout will converge within a reasonable amount of time, to a reasonable proximity of performance target, or at all. 2. Using other layouts as inputs train a CNN model - There are not a lot of layouts available for any given topology/circuit architecture. Additionally, these layouts are not accessible publicly and are one of the most protected IPs in analog IC design. On top of this, topologies are changed constantly to get better performance, hence transferred models from old circuits may not have good-enough accuracy.

3 Constraint Generation

3.1 Motivation

Although Magical contains a layout constraint extractor, it is a very basic tool and uses pattern libraries to check for symmetry. This is not good enough for larger circuits, hence work has been done on improving structure recognition and constraint generation for analog circuits. The default placement and routing engines use these constraints to a large extent to work correctly, hence this is an important step of the process. Constraints also include factors like grouping, where some devices are kept close to each other, due to their nature of operation in the circuit. As discussed in Section 2, this is done manually by experts as well, but as circuits get larger and hierarchies get more complicated, this task becomes difficult and its possible that some aspects may be missed by an expert as well. Hence, training these models with expert written constraints is not ideal.

Previous state of the art constraint generators could not detect all kinds of structures and did not handle hierarchical circuits well. [4] handles this by using Graph Neural networks, selective topological search, distance thresholding, newer methods for checking symmetry, excluding power nets, etc. Checking symmetry is especially improved in [4], because it starts checking from the pins, instead of devices. This is similar to how experts check circuit netlists. Power nets are also excluded early on by checking the body connections of the devices (which are usually tied to GND or VDD). Along with this, the issue of training the model is solved by using an expert generated layout, which is analysed by its constraint extractor. So, instead of learning constraints associated with a netlist, it learns how the devices and nets placement symmetry is associated with the circuit netlist. Additionally, only one expert layout was used to train the model in [4].

3.2 Implementation

The goal is to design a generalized framework to comprehend and transfer various constraint types across differing AMS topologies and technology nodes.

3.2.1 Represenation: A multi-graph with node-level heterogeneity is constructed based on the AMS-circuit netlist. It consists of two distinct types of vertex sets: devices and nets. Each device in the graph can either represent an individual component or an embedded multi-graph, functioning as a module. A feature vector is defined for each node type. Device nodes are encoded with a 13-element vector(like device type, the total connection count). Nets are encoded with a 7-element vector(like type of nets, the degree of node).

3.2.2 Constraint learning. Initially only the inter-symmetry constraint for net nodes is learned. The following is the overall flow:

(1) The AMS circuit netlist topology is converted into a multi-graph as mentioned earlier.
(2) This graph is the input to the Nest-GNN to learn optimal embeddings for the inter-symmetry constraint on the net nodes(this is done because there are fewer net nodes compared to those of device and other related constraint types near the pins can be transferred easily from these).
(3) Feature vectors, also known as embeddings, for each node type are generated.

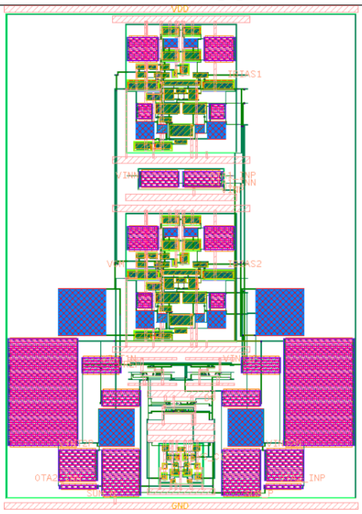
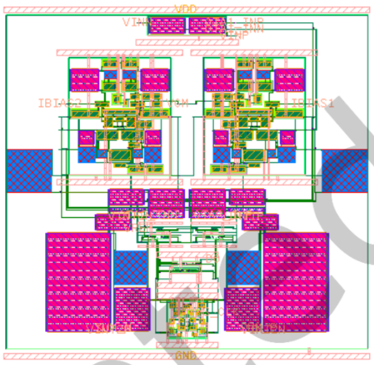
CLT method	DATE'24 [8]	This work
Layout image		
Sym. wirelength (μm)	996800	896000
Total wirelength (μm)	2443600	2575600
Area (μm^2)	14122	11760

Fig. 2. Layout comparison with previous work, taken from [4]

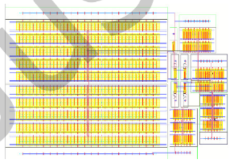
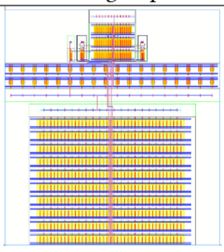
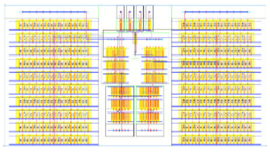
	w/o annotation	with functional group annotation	with transferred constraints
Layout			
offset(mV)	-2.212	8.068	0.3384
delay(ns)	3.077	3.014	2.708
power(μ W)	1.317	1.413	1.27
wirelength(#R)	7004	7184	7080
area(μm^2)	1644.07	2377.728	1768.534

Fig. 3. Post-layout performance summary of a strongArm comparator circuit with layouts generated with diferent constraint methodologies, taken from [4]

(4) The node embeddings are passed to a multi-model Nest GNNs(each model with a different configuration, like the number of layers) for training. Now, we get the updated node embeddings.
(5) Euclidean distances of every possible pin-net combination are calculated. These distances are used as similarity scores to tell if two pins should be classified as a net inter-symmetry pair.If the distance is below a predetermined threshold, the pin-net pair is added to the constraint

group. A lower value of the threshold makes it less strict and helps in identifying some partial-symmetry pairs.
(6) Ground truth values are extracted from the expert layout corresponding to the input netlist.
(7) A binary cross-entropy loss function is used to update the Nest-GNN models.
(8) This process is continued until the loss values of all models converge to about the same level.

3.2.3 Constraint Transfer Using SelecTS Algorithm . After learning the symmetry constraint on net pairs, other constraints are inferred using the Selective Topological search(SelecTS) Algorithm. The following is the overall flow:

(1) A hierarchical tree is built to capture the structural organization of the circuit using the netlist and the Process Design Kit (PDK).
(2) Digital blocks are excluded, as symmetry constraints are not applied to them.
(3) Power wires are identified by traversing the hierarchical tree from the bottom up.
(4) A graph is constructed where each node represents either a device or a net, as previously defined.
(5) The trained **Nest-GNN** is used to generate embeddings for each node, capturing both topological and functional context.
(6) All pairs of net embeddings are compared using Euclidean distance, and candidate inter-symmetry pairs are marked based on a threshold criterion.
(7) Connected candidate pairs are grouped into initial inter-symmetry sets.
(8) **Symmetry Constraint Completion:** The **SelectTS Algorithm 1** is applied to complete full, partial, and intra-symmetry groups. This is achieved by performing a dual-sided hierarchical breadth-first search from the detected symmetrical pin-net pairs.
(9) **Grouping and Matching Constraint Completion:** The **SelectTS Algorithm 2** is used to complete device grouping and net impedance matching constraints. This step relies on known symmetrical net groups and examines connection patterns and device similarities.

Figure 2 shows a comparison between [4] and previous state of the art. Figure 3 shows the comparison between no-constraints, constraints generated by [4] and functional group annotation, which groups similar devices, with no consideration for overall symmetry or matching.

This work introduces a robust framework for understanding and transferring various constraint types across different AMS topologies and technology nodes. By leveraging the Nest-GNN model for learning inter-symmetry constraints and the SelecTS algorithm for constraint transfer, the approach efficiently captures both topological and functional dependencies within AMS circuits. The resulting model can be utilized in automated circuit design workflows to optimize placement and symmetry constraints, significantly reducing manual effort and improving design accuracy. Once trained, the model can be applied to diverse circuit designs, enabling fast adaptation to new topologies and technology nodes. Results are in Figures 2 and 3.

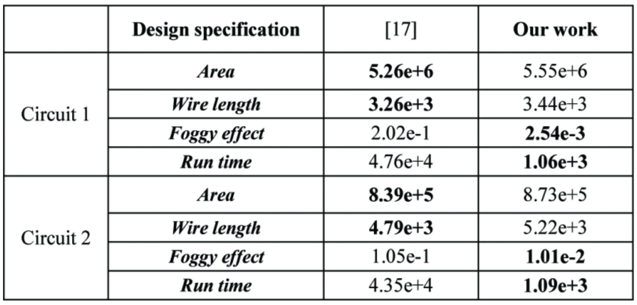
4 Placement Generation

4.1 Motivation

Placement is a module arranging or a floorplanning problem, where we need to fit some modules into a limited area, while minimizing distances between some modules and maintaining symmetry among others. Two major metrics for placement are area and wirelength. Area is directly related to cost and wirelength is related to the resistance between devices, which should be minimized, but not at the cost of more important metrics, like offset voltage and CMRR. Calculation of these parameters is more complicated, as we need to run extracted simulations to get these values, but the rule of thumb is higher symmetry means lower offset voltage (lower is better) and higher CMRR (higher is better). In an ideal world, the entire layout can be made completely symmetric to get the best performance, but due to area constraints, it is usually not possible. Additionally, we need to prioritise some nets and devices higher than others, as there is a hierarchy for how symmetry of various types would affect these metrics. So, it is obvious that better constraint generation would lead to better performance, which has been dealt with in Section 3.

[5] specifically deals with placement where we have additional metrics to minimize, i.e., Foggy effect in this case. This is an issue in newer nodes where the sizes of the devices can vary a bit due to the lithographic process non-idealities. [5] has modelled this through a gaussian distribution, which means that the values for foggy effect can be calculated quite easily. It is also important to note that foggy effect can be replaced with any other parameter, as long as it does not slow down the algorithm too much. This fact can be used with the framework from Section 5 to get a good placement generator, but this has not been tested anywhere. [5] also improves upon the convergence rate of previous algorithms by a lot, through a simpler representation of the modules. This follows the idea of working at a given hierarchy by considering modules as discrete items. The paper improves on its convergence speed further by including a legalization step, which punishes actions which would lead to overlapping modules. These requirements for minimum distance are decided by the manufacturer, and are part of the design rules. It is essential that these design rules are always followed.

Runtime is reduced from about 12 hours to 15 minutes.



	Design specification	[17]	**Our work**
Circuit 1	***Area***	**5.26e+6**	5.55e+6
	Wire length	**3.26e+3**	3.44e+3
	Foggy effect	2.02e-1	**2.54e-3**
	Run time	4.76e+4	**1.06e+3**
Circuit 2	***Area***	**8.39e+5**	8.73e+5
	Wire length	**4.79e+3**	5.22e+3
	Foggy effect	1.05e-1	**1.01e-2**
	Run time	4.35e+4	**1.09e+3**

Fig. 4. Simulation results for two circuits, taken from [5]

4.2 Implementation

The objective is to use the Advantage Actor-Critic (A2C) reinforcement learning method to achieve optimal placement, taking into account factors such as foggy effect, area, wire length, and runtime.

4.2.1 Method. Modules are represented as nodes using the B*-tree structure, which helps reduce the dimensionality of the placement problem. This representation provides only relative positioning information—where the root module is placed at the bottom-left corner, and child modules are positioned either to the right or above their parent. In this work, the foggy effect is modeled using the Gaussian Point Spread Function (PSF). The input to the system includes a netlist and initial coordinates of the modules, which are then mapped to the B*-tree structure. The cost of a placement is calculated using the following function:

$$\text{Cost} = \alpha \cdot \text{Area} + \beta \cdot \text{Wire Length} + \phi \cdot \text{Foggy Effect}$$

where α, β, and ϕ are user-defined coefficients for area, wire length, and foggy effect variation, respectively.

If the computed cost is below a user-specified threshold, the design is considered feasible. Otherwise, the A2C agent proposes an action—such as relocating a module—to improve the design. This action is applied to the environment, resulting in a new state. A corresponding reward is then returned based on the effectiveness of the action. Using this reward, the agent updates its policy. This interaction between the agent and the environment continues iteratively until an optimal cost is achieved or a predefined limit of 500 episodes is reached.

Figure 4 shows the comparison between [5] and previous state of the art. There is a significant difference in the runtime, while not compromising on the area and wirelength.

This approach successfully utilizes the Advantage Actor-Critic (A2C) reinforcement learning method to optimize module placement by considering key factors such as area, wire length, and the foggy effect. By representing modules using the B*-tree structure, the method reduces the dimensionality of the placement problem and efficiently explores the design space. The model's ability to iteratively improve placement decisions based on feedback from the environment enables the generation of highly optimized designs. Once trained, the resulting model can be deployed to automatically optimize module placements in circuit design workflows, providing faster and more accurate solutions compared to traditional placement methods

5 Placement Quality Prediction

5.1 Motivation

When dealing with a block like Comparator or an OTA, we are concerned with one or two performance parameters in extracted simulations, like offset voltage or CMRR, which are highly dependent on the layout. But to check these values, routing and extracted simulations need to be completed, which is highly time-consuming. Placement quality prediction can be done by a CNN which takes the placement image as an input and outputs a score for how good or bad the offset voltage or CMRR will be. Placement quality prediction was suggested in [3], but not implemented. Having this block means the iterations for placement can be performed much faster. Later works did try it, but there were issues with accuracy. This is due to the fixed nature of the CNN model being used. Additionally, generating a model manually is a time-taking task in itself. As discussed earlier, topologies keep changing in designs in search of higher performance, which means generating a CNN model manually for each design may not be feasible, as it can take weeks or months worth of time. Data scarcity is another problem here, as we do not have thousands to tens of thousands of placement results to train the model on. Thus [2] uses a dataset of about 16K placements and their results which are generated on the circuit of interest, and uses NAS to generate CNN models specific to it. They have tried training the models generated for one design on other, similar circuits and the accuracy is above 90%.

5.2 Implementation

The objective is to find a neural network architecture using NAS that provides the most accurate prediction of layout performance from the placement data. The authors chose CNN as the main architecture of interest. Typically, NAS consists of three key parts: search space, evaluation strategy, and search strategy. A graph-based search space and a graph propagation strategy are utilized. Detailed explanations for these are given in the upcoming parts.

5.2.1 Search Space. It is divided into two parts, searchable and fixed.The fixed parts are responsible for performing standard transformations, e.g, downsampling. The searchable parts are capable of extracting high quality learned representation to extract critical information from the input features. It is composed of six Directed Acyclic Graphs(DAGs)), named guide-DAGs. Each vertex represents a set of operations, more specifically, convolution, mixed convolution, and atrous convolution operations. Each edge $e(u, v) \in E_i$ represents the propagation of the output tensor of vertex u to the input of v. Each DAG is constructed with the maximum possible edges to provide all possible connections in the sampled DAG

5.2.2 Search Strategy - Graph Propagation Strategy. It involves edge sampling and operation sampling. Weights are assigned for edges as well as operations in each vertex. These are learned/updated throughout sampling iterations based on the accuracy obtained from including the specific sampled-DAG in the overall architecture.

$$\text{Edge weight updation:} \quad w_e = w_e \cdot \exp\left(\alpha(\eta - \beta)\right)$$

Sampling of edges and corresponding vertices are done based on the edge weights. An edge is probabilistically sampled and the one with higher weight has a higher probability of getting sampled. Now, for a sampled vertex, operation sampling is done based on the operation weights in that vertex. An operation with higher weight has more probability of getting selected. During the weight updation process, the weights of those edges and operations that are included in the sampled-DAG increase. Thus, in the subsequent iterations, the probability of these getting chosen increases. This sampling and weight updation continues until the model performance converges. This graph propagation allows each sampled-DAG to have a varied number of vertices, edges, and operation combinations. Despite this great flexibility, the search cost is just 0.5 days, indicating its remarkable efficiency.

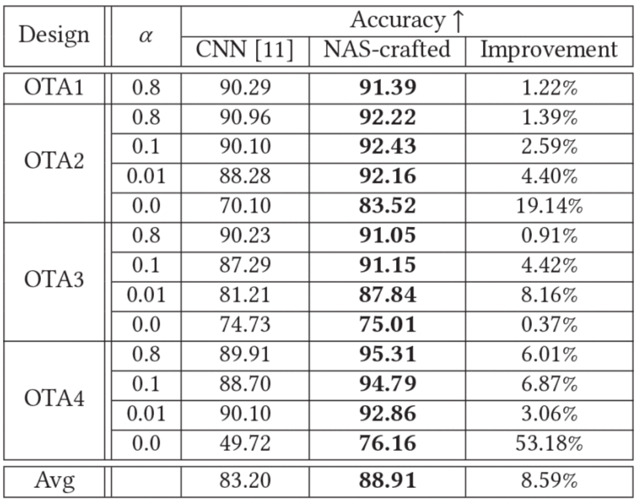
Design (#transistor)	Accuracy ↑		
	CNN [11]	NAS-crafted	Improvement
OTA1 (31)	90.29	**91.39**	1.22%
OTA2 (31)	92.61	**92.74**	0.14%
OTA3 (22)	91.33	**93.57**	2.45%
OTA4 (23)	92.05	**95.37**	3.61%
Avg	91.57	**93.27**	1.85%

Fig. 5. Placement quality prediction result, taken from [2]

5.2.3 Evaluation Strategy. The model is trained on the training split of the dataset, and the accuracy for the validation split is calculated. This accuracy is directly employed as our search objective.

Figure 5 contains the accuracy values when the NAS is trained specifically for the dataset. Figure 6 contains the accuracy values when the model is generated for OTA1 using NAS, but finetuned to a different dataset, with some finetuning ratio. Max training dataset is 0.8, hence $\alpha = 0.8$ means training completely on another dataset.

This NAS-based approach efficiently identifies an optimal CNN architecture tailored for layout performance prediction by leveraging a graph-based search space and a graph propagation strategy. Once trained, the resulting model can be used to predict post-layout performance directly from placement data, enabling faster design iterations and early feedback without requiring full layout



Design	α	Accuracy ↑		
		CNN [11]	NAS-crafted	Improvement
OTA1	0.8	90.29	**91.39**	1.22%
OTA2	0.8	90.96	**92.22**	1.39%
	0.1	90.10	**92.43**	2.59%
	0.01	88.28	**92.16**	4.40%
	0.0	70.10	**83.52**	19.14%
OTA3	0.8	90.23	**91.05**	0.91%
	0.1	87.29	**91.15**	4.42%
	0.01	81.21	**87.84**	8.16%
	0.0	74.73	**75.01**	0.37%
OTA4	0.8	89.91	**95.31**	6.01%
	0.1	88.70	**94.79**	6.87%
	0.01	90.10	**92.86**	3.06%
	0.0	49.72	**76.16**	53.18%
Avg		83.20	**88.91**	8.59%

Fig. 6. Transfer learning result; α is the finetuning factor, taken from [2]

generation and simulation. This significantly accelerates the analog design flow by providing accurate layout-aware insights at early stages.

6 Layout-aware sizing

6.1 Motivation

All previous works are purely for layout generation, but there are some things which are interlinked with the design aspects of the circuits, such as sizing of the transistors. For a given circuit topology, it is the designer's responsibility to find out the sizes of the transistors for which we maximize the performance. Usually these performance parameters, like bandwidth and gain are linear functions of the sizes, so it is easy to compute them for any given performance requirement. But, for a design where layout area is a constraint, it may not be easy to find out the optimal sizes, as placement and routing take up a lot of time. This is a typical problem which is solved through RL. But due to the requirement of placement, routing and post-layout simulations, convergence time may be huge. Thus, some careful shortcuts need to be taken.

Relation between schematic and post layout sims. Although not equal, for a given method of layout, the schematic and post-layout sims results are highly correlated. This can be used to skip expensive layout and post layout simulations, when the model can assume a certain correlation. There are no works which use this strategy to speed up convergence in these sizing problems. [1] uses this, along with multi-fidelity BNNs as an optimization algorithm to decide sizing for a given circuit topology. BNNs are employed here because they are effective at handling scarce datasets and avoid overfitting.

Because of skipping post-layout simulations and a better optimization algorithm, [1] reaches a better figure of merit than previous works, while being faster than previous works.

6.2 Implementation

The goal is to achieve layout aware sizing using ML-based simulation in-the-loop automation method. No previous existing dataset is used in this paper, and all training data is generated during the optimization process.

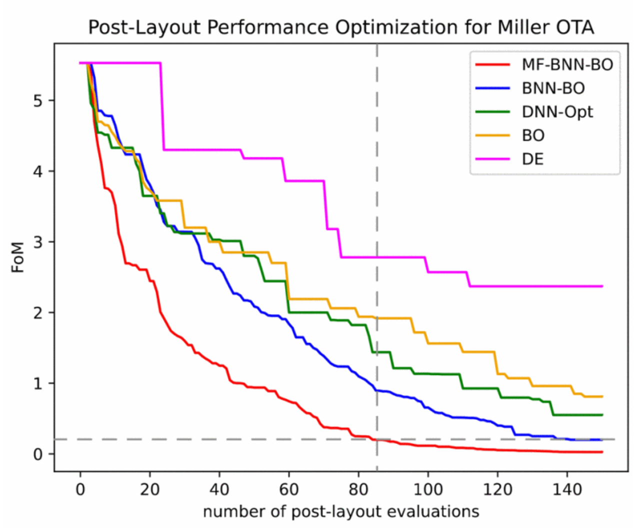


Fig. 7. Miller OTA Post-Layout Performance Optimization, taken from [1]

6.2.1 *Problem Formulation.* :

The AMS schematic-level sizing and layout aware sizing task is formulated as a constrained optimization probem as below.

$$\begin{aligned} &\text{minimize} \quad f_0(\mathbf{x}) \\ &\text{subject to} \quad f_i(\mathbf{x}) \leq 0 \quad \text{for } i = 1, \ldots, m \end{aligned}$$

where,

$\mathbf{x} \in \mathbb{R}^d$ (design space)
$f_0(\mathbf{x})$ is the objective function
$f_i(\mathbf{x})$ is the i^{th} constraint.

Throughout the paper ,the quality of the design is evaluated using Figure of Merit (FoM) defined in the following form:

$$\text{FoM}(\mathbf{x}) = w_0 \times f_0(\mathbf{x}) + \sum_{i=1}^{m} \min\left(1, \max\left(0, w_i \times f_i(\mathbf{x})\right)\right) \quad (1)$$

The FoM is designed to penalize constraint violation and to give equivalent value for all feasible designs.

6.2.2 *Schematic-Level Sizing automation.* The initial dataset for training is obtained by sampling random points in the design space and simulating them via a SPICE simulator to get the ground truth performance metrics. Now, a trust region is initialized(hypercube surrounding the 'best' point).This trust region determines the bounds of the exploration space. A separate BNN model for each performance metric is constructed. These BNN models are trained using the dataset. After training, we approximate the function BNN of each constraint/performance metric fit, using Hamiltonian Monte Carlo(HMC). A set of r points are sampled from the trust region. Now, the function approximations computed using HMC are used to calculate FoM for each of the r pointsand the top q points with the least FoM values are selected. SPICE simulations for these q points are done and are added to the dataset. Now, the trust region is updated(hypercube surrounding the point with the least FoM). Then, the BNN models are retrained with the dataset containing the newly added points.

6.2.3 Multi-Fidelity BNN Model. Instead of training separate models for schematic-level and post-layout simulation, a co-learning BNN that learns from them is proposed.
The BNN outputs two values for each input design x:
$\phi(x)[1] \rightarrow$ Schematic-level prediction (low fidelity)
$\phi(x)[2] \rightarrow$ Post-layout prediction (high fidelity).
They are not separate networks, but two output heads of the same shared network. This allows the model to learn correlations between the fidelities (schematic-level and layout level) The trust region is updated only using post-layout results for reliable exploration of the design space. For choosing points to add to the dataset, FoM is computed for both fidelities and chosen accordingly. For selected points, it is randomly chosen whether to do schematic or layout level simulations(to get ground truths). This balances the cost incurred, aka time/computation spent in simulations, to reach a good accuracy score.
Convergence speed comparison against other algorithms is present in Figure 7.

In summary, the proposed approach effectively combines Bayesian modeling, trust region–based exploration, and multi-fidelity simulation strategies to automate layout-aware sizing. By generating data on the fly and leveraging schematic–layout correlations, the method significantly reduces simulation costs while maintaining high accuracy, paving the way for scalable and efficient AMS design automation.

7 Conclusion

This summary paper has covered four areas where work has been done to improve analog layout automation. All of these contribute to the problem statement independently, as well as, help other areas perform better. In the overall magical flow, replacing the original placement engine or the constraint generator is quite easy. Hence, it is important to note that while results from Magical alone are not groundbreaking, it provided a strong framework for others to build upon, which has been working great, as seen through the state of the art and their improvements.

Overall, this problem statement is quite different from the usual use case of ML, as in we do not have a lot of training data, and we cannot iterate quickly, but we have a lot of compute resources and are willing to sacrifice time to get better results, as manual layout takes months worth of time anyways. We also want to emphasize that aim of this paper is to provide the reader with a new perspective on how ML could be used in constrained environments.

References

[1] Ahmet F. Budak, Keren Zhu, and David Z. Pan. 2023. Practical Layout-Aware Analog/Mixed-Signal Design Automation with Bayesian Neural Networks. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD).* 1–8. doi:10.1109/ICCAD57390.2023.10323923

[2] Chen-Chia Chang, Jingyu Pan, Zhiyao Xie, Yaguang Li, Yishuang Lin, Jiang Hu, and Yiran Chen. 2023. Fully Automated Machine Learning Model Development for Analog Placement Quality Prediction. In *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC).* 58–63.

[3] Hao Chen, Mingjie Liu, Biying Xu, Keren Zhu, Xiyuan Tang, Shaolan Li, Yibo Lin, Nan Sun, and David Z. Pan. 2021. MAGICAL: An Open- Source Fully Automated Analog IC Layout System from Netlist to GDSII. *IEEE Design Test* 38, 2 (2021), 19–26. doi:10.1109/MDAT.2020.3024153

[4] Kaichang Chen and Geroges Gielen. 2025. A Generalized Constraint Learning and Transfer Methodology with Net-First Graph Neural Network and Selective Topological Search for Hierarchical Analog / Mixed-Signal Circuit Layout Synthesis. *ACM Trans. Des. Autom. Electron. Syst.* (March 2025). doi:10.1145/3722556 Just Accepted.

[5] Mirvala Sadrafshari, Octavia Dobre, and Lihong Zhang. 2024. Reinforcement-Learning-Based Foggy-Aware Optimal Placement Method for Analog and MixedSignal Circuits. In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. doi:10.1109/ISCAS58744.2024.10558520

A Details about Implementation

A.1 Constraint Generation

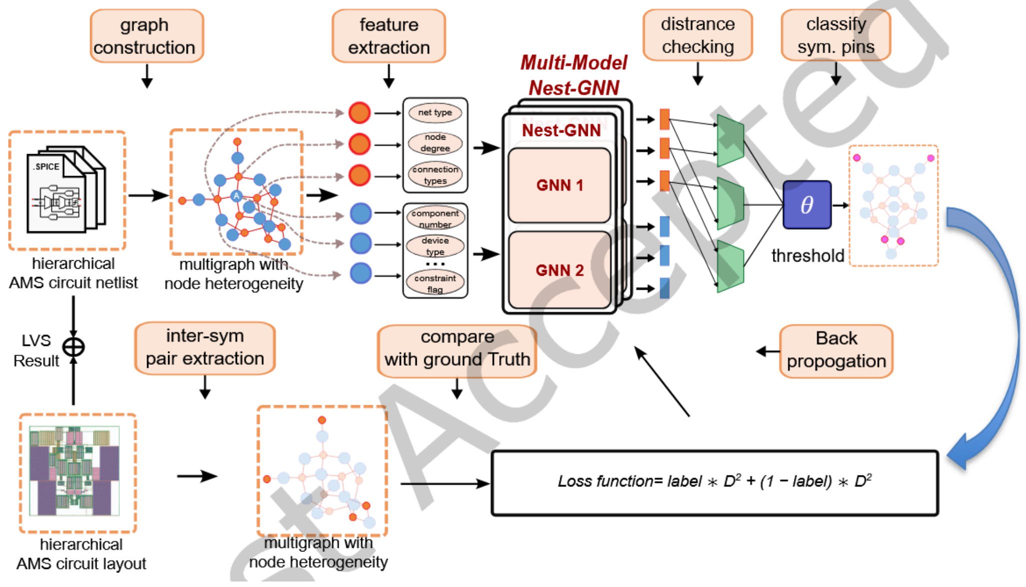


Fig. 8. Overview of the proposed constraint learning framework with multi-model Nest-GNN.

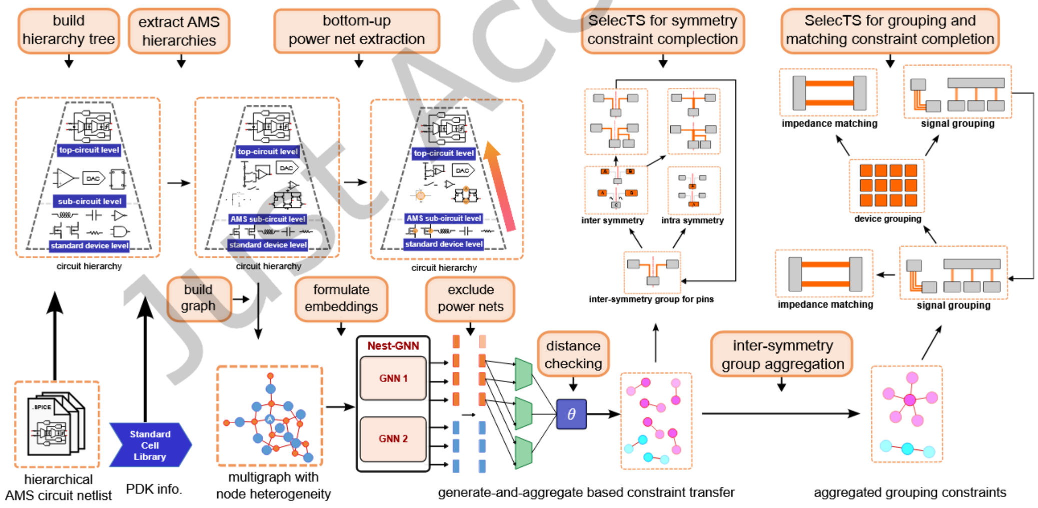

build hierarchy tree
extract AMS hierarchies
bottom-up power net extraction
SelecTS for symmetry constraint complection
SelecTS for grouping and matching constraint completion
top-circuit level
sub-circuit level
standard device level
circuit hierarchy
AMS sub-circuit level
impedance matching
signal grouping
device grouping
inter symmetry
intra symmetry
inter-symmetry group for pins
build graph
formulate embeddings
exclude power nets
distance checking
inter-symmetry group aggregation
Nest-GNN
GNN 1
GNN 2
hierarchical AMS circuit netlist
Standard Cell Library
PDK info.
multigraph with node heterogeneity
generate-and-aggregate based constraint transfer
aggregated grouping constraints

Fig. 9. Overview of the proposed constraint transfer framework.

A.2 Placement Generation

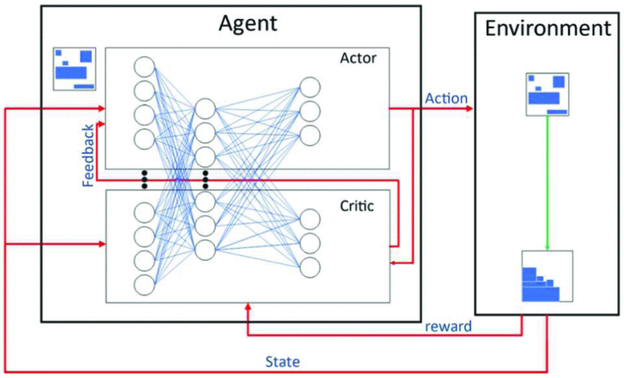


Fig. 10. Advantage actor-critic (A2C) model

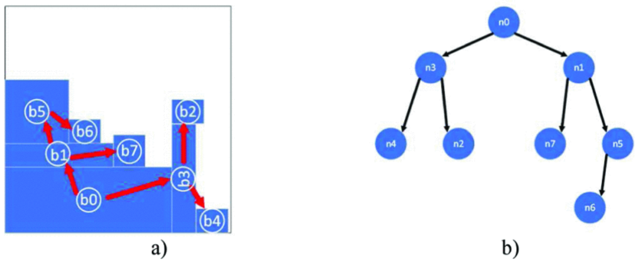


Fig. 11. B*-tree representation a) Correspondence placement of B*-tree representation b) B*-tree representation

A.3 Placement Quality Prediction

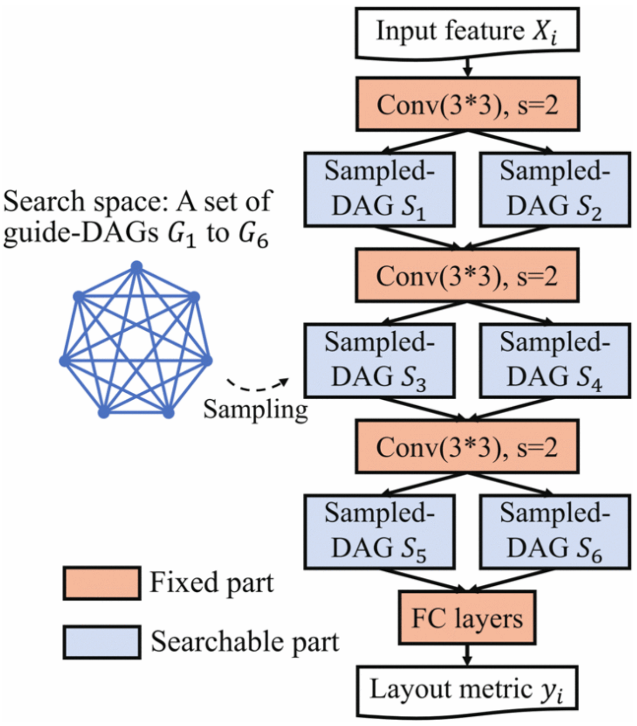


Fig. 12. The search space and the sampled architecture of our NAS method.Our model architecture is separated into the searchable and fixed part. The searchable part composes of six sampled DAGs, whose structures are sampled from the guide-DAGs. The fixed part has three convolution layers with kernel size 3 and stride 2 to perform downsampling and two fully-connected layers to produce the layout metric

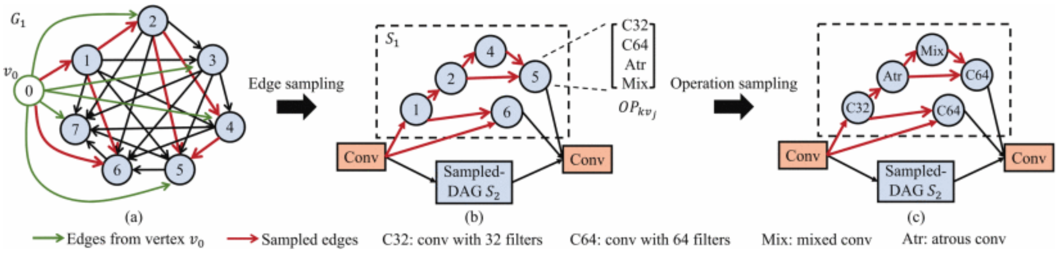


Fig. 13. An example of graph propagation. For edge sampling, we select edges from the guide-DAG G1 (a) to form the sampled DAG S1 (b). Then, we choose the operations of the selected vertices in operation sampling to form the sampled architecture (c)

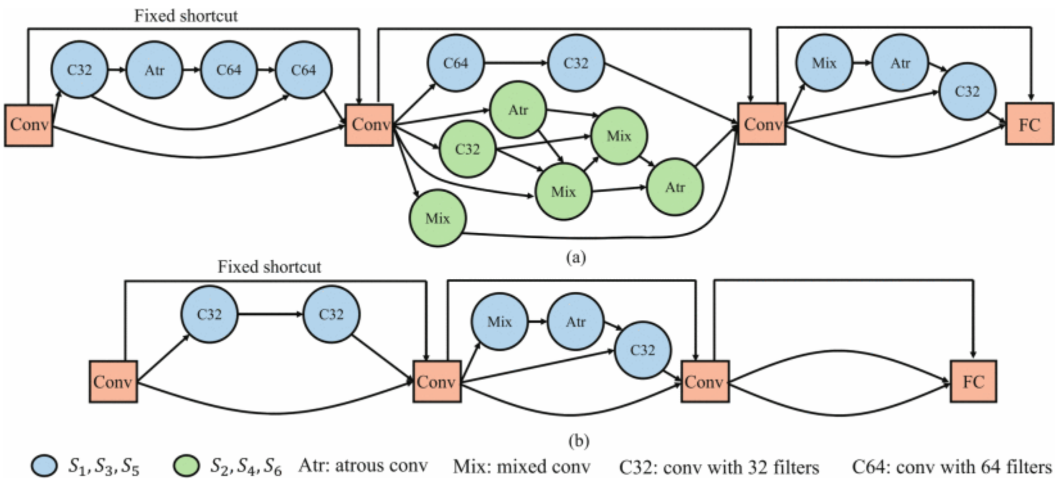


Fig. 14. NAS-crafted models (a) for OTA1 and (b) for OTA3

A.4 Layout-aware sizing

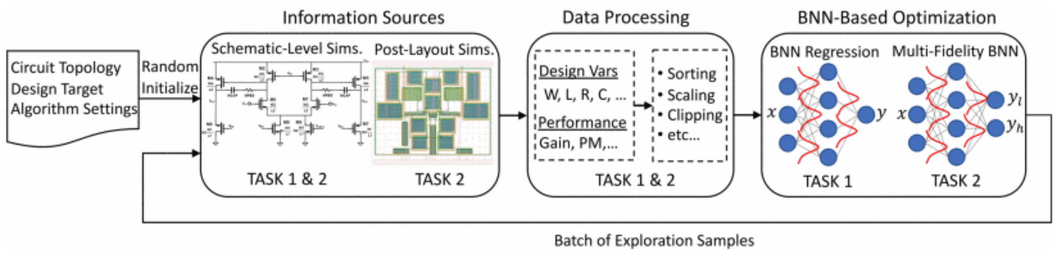


Fig. 15. Proposed AMS Automation Framework

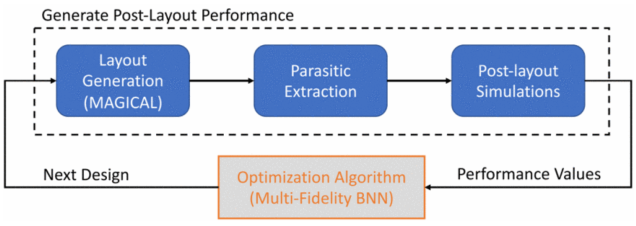

Generate Post-Layout Performance
Layout Generation (MAGICAL)
Parasitic Extraction
Post-layout Simulations
Next Design
Optimization Algorithm (Multi-Fidelity BNN)
Performance Values

Fig. 16. Post-Layout Performance Based Optimization

Received 2 May 2025