GeniusRoute: A New Analog Routing Paradigm Using Generative Neural Network Guidance

Keren Zhu, Mingjie Liu, Yibo Lin, Biying Xu, Shaolan Li, Xiyuan Tang, Nan Sun, and David Z. Pan
ECE Department, The University of Texas at Austin, Austin, TX, USA
{keren.zhu, jay_liu, yibolin, biying, slliandy, xitang}@utexas.edu, nansun@mail.utexas.edu, dpan@ece.utexas.edu

***Abstract*—Due to sensitive layout-dependent effects and varied performance metrics, analog routing automation for performance-driven layout synthesis is difficult to generalize. Existing research has proposed a number of heuristic layout constraints targeting specific performance metrics. However, previous frameworks fail to automatically combine routing with human intelligence. This paper proposes a novel, fully automated, analog routing paradigm that leverages machine learning to provide routing guidance, mimicking the sophisticated manual layout approaches. Experiments show that the proposed methodology obtains significant improvements over existing techniques and achieves competitive performance to manual layouts while being capable of generalizing to circuits of different functionality.**

I. Introduction

The endeavor to automate routing for analog and mixed-signal (AMS) integrated circuits (IC) has been continuing for years [1]. However, due to the incapability of following designers' experience and considering various layout-dependent effects, little adoption has been demonstrated in practical analog design flow [2].

Existing efforts on analog routing can in general be classified into three categories: template-based, simulation-based, and heuristic constraint-based approaches. **Template-based** approaches generate routing based on human-designed templates [3], [4]. These techniques can achieve high post-layout performance for design-specific applications such as layout retargeting, which is difficult to scale to general designs due to complexity of input templates. **Simulation-based** approaches rely on simulations to analyze circuit functionality and optimize performance. Through sensitivity analysis, the work of [5], [6] identified critical nets and matching constraints, which were embedded in the layout optimization process. These methodologies can be generalized for various circuits and performance metrics, while the required amount of simulations may not scale with design complexity. **Heuristic Constraint-based** approaches tackle analog routing by identifying human layout techniques and embedding them as constraints. The most widely adopted heuristic is the symmetric net pair constraint [7], [8], [9], [10], [11], [12], [13], [14]. Ou et al. [9] further extended it to different levels of geometrical matching constraints. There are other works that forbid routing over the active regions of transistors [11], [15], optimize power routing [16], [17] and propose shielding critical nets [14]. With complicated real analog designs, these simple heuristics are often not enough to cover various layout dependent effects or follow the experience of design expertise.

In practice, the performance of analog circuits is sensitive to even minor layout changes. We conduct a simple experiment here on a comparator to show how subtle changes in clock routing can affect the offset performance. Figure 1 shows a manual layout of a comparator circuit that is designed by an experienced designer where the clock routing detours around the layout boundary. This particular routing choice is "counter-intuitive" in the sense of minimum wirelength. Better solution could be easily achieved by symmetrically routing in the center of the layout for shorter wirelength. To understand the reason behind such a design choice, we setup an experiment on the clock net routing. In the experiment, we symmetrically route the clock net starting from the exact center, then gradually push it towards the layout boundary, as shown in Fig. 2. Figure 3 plots the post-layout simulation result of input-referred offset, upon which we have following observations. (1) Even with perfect symmetry, clock routing can still significantly affect the offset of this design. This effect might be caused by the clock coupling with sensitive nets, which makes it difficult for automatic tools to predict or optimize for. (2) The manual layout solution might not be optimal for a particular performance metric, but in general, is less sensitive to subtle layout changes. As a result, it can be more robust against layout-dependent effects and process variations.

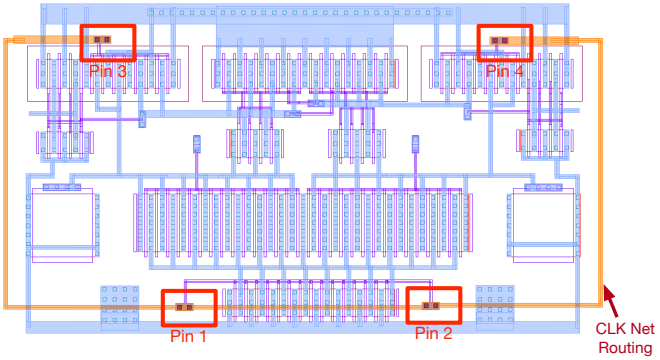


Fig. 1: The routing layers of a manual comparator layout.

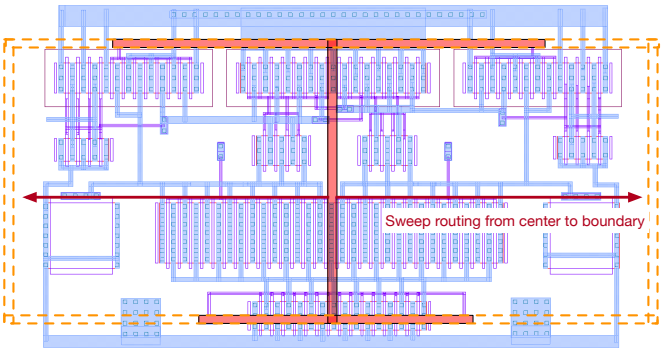


Fig. 2: Experiments on comparator clock routing.

Such design expertise is hidden in well-planned manual layouts, but there is yet lacking an efficient and general way to transfer them into automated routing tools. For instance, from the above experiment, one might summarize a simple heuristic that clock nets should detour around the module to avoid coupling with the nets at the center of the layout. However, this is a design-specific knowledge and is hard to be transferred into a generalized constraint; the heuristic might be

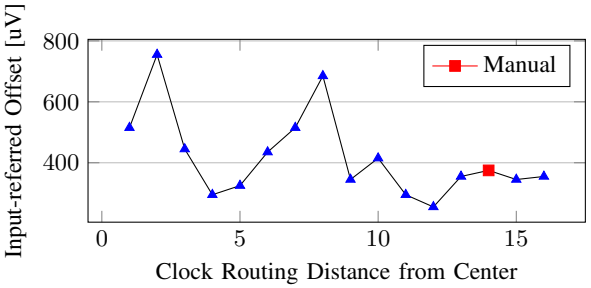


Fig. 3: Comparator input-referred offset experiment results.

valid for some placements but not for all situations. Some routing decisions are based more on experience and multiple performance trade-off considerations rather than simply following explicitly listed rules. Furthermore, arbitrary human knowledge is also often hard to be fully encoded and optimized using traditional programming practices. Thus, a methodology that can implicitly summarize design expertise and automatically extract constraints from good existing layouts is preferred in both the development of analog routing tools and the practice in the real design flow.

Recent advancements in machine learning have demonstrated its effectiveness in learning hidden structures from data without explicit instructions [18], [19]. Among the machine learning techniques, generative neural networks have demonstrated successful application in different fields of VLSI design automation [20], [21], [22], [23]. While it is difficult to embed every human layout technique into rules in an automatic routing algorithm, we can rely on machine learning models to extract layout patterns and infer the human behavior on routing. In some sense, it is building a template library automatically via machine learning models. Compared to existing work in rule-based analog layout template mining [24], machine learning approaches provide enhanced generality and flexibility in a fully-automatic manner.

In this paper, we propose ***GeniusRoute***, a new methodology for automatic analog routing with guidance from a generative neural network. Leveraging the machine learning technique based on variational autoencoder (VAE), GeniusRoute extracts latent layout strategies of human engineers and applies the learned knowledge in guiding the routing algorithm. Our main contributions are summarized as follows:

- We propose a fully automated AMS routing that can imitate human intelligence from well-designed manual layouts.
- We develop a new methodology for implicit extraction of routing expertise through machine learning algorithms instead of explicit hard-encoded constraints.
- We propose a detailed analog routing algorithm honoring the generated guidance from the machine learning model as well as the enforcement of additional constraints such as net symmetry.
- Experimental results demonstrate that our proposed framework achieves competitive performance compared with manual layouts from experienced human engineers.

The rest of this paper is organized as follows. Section II formulates the analog routing problem and introduces the background on VAE. Section III presents the GeniusRoute framework. Section IV reports the experimental results. Finally, Section V concludes the paper.

II. Preliminaries

In this section, the problem formulation of analog circuit routing is presented first (Sec. II-A). Then brief overviews of two sub-problems in GeniusRoute are given: routing guide generation (Sec. II-B) and guided analog detailed routing (Sec. II-C). An introduction to the machine learning model, VAE, is also presented (Sec. II-D).

A. Analog Routing Problem Formulation

The analog routing problem can be formulated as follows: given a set of placed devices $M = \{m_i | 1 \leq i \leq |M|\}$, a set of nets $N = \{n_i | 1 \leq i \leq |N|\}$, a set of symmetry net pairs $N^{SP} = \{n_i^{SP} | 1 \leq i \leq |N^{SP}|\}$, a set of self-symmetry nets $N^{SS} = \{n_i^{SS} | 1 \leq i \leq |N^{SS}|\}$, a set of special nets with types $\{N_i^T | 1 \leq i \leq |N_i^T|\}$, and the design rules, route all the nets and optimize the post-layout performance.

GeniusRoute decomposes the analog routing problem into two sub-problems: routing guide generation and guided analog detailed routing.

B. Routing Guidance Generation

Routing guide generation learns the human behavior in manual routing and generates the routing guidance for the downstream router. The problem can be formulated as follows: given the placement M, a set of nets $N^* \in N$ with type N^T, predict the **probability map** $r \in \Re^{n \times n}$, where $r_{i,j}$ describe the inferred probability that N^* is likely to be routed in region i, j of the layout. Intuitively, routing guide is a 2D probability map of the likelihood that the given net will be routed in each region by a human engineer.

C. Guided Analog Detailed Routing

Guided analog detailed routing takes placement M, nets N, design rules and a set of generated routing guidance $R = \{r^i | 1 \leq i \leq |R|\}$ as input, and routes all the nets while honoring symmetric constraints and routing guidance.

D. Variational Autoencoder

We adopt VAE [25] as the base for our machine learning models. VAE is an unsupervised learning algorithm to extract efficient data encoding (latent variables) from the training data. VAE follows a similar idea of autoencoder in constructing an encoder-decoder structure. Figure 4 shows an autoencoder architecture. An autoencoder takes samples $\{x_i\}_{i=1}^n$ from domain X and finds an efficient encoding z of the data. It consists of two neural networks: the encoder E_ϕ and the decoder D_θ. Encoder E_ϕ converts input data x into low-dimensional latent variable vector z, and decoder reconstructs $\hat{X}$ from z. By minimizing the reconstruction loss between original inputs X and reconstructed outputs $\hat{X}$, i.e., $\mathcal{L}(x, \hat{x})$, the autoencoder learns an efficient encoding of X with enough information to reconstruct the inputs.

VAE further uses parametric distribution, usually Gaussian, to model X and Z, i.e., $P(X|z, \theta) \sim \mathcal{N}(\mu(z), \sigma(z))$ and $z \sim \mathcal{N}(0, I)$. The objective is to maximize the probability of each X in the training set under the entire generative process, i.e.,

$$P(X) = \int P(X|z, \theta) P(z) dz.$$

During the training process, $\mu(z)$ and $\sigma(z)$ are trained by an encoder, and the objective is to maximize:

$$logP(X|z) - \mathcal{D}_{KL}[Q(z|X)||P(z)],$$

where $logP(X|z)$ is a reconstruction log-likelihood and $\mathcal{D}_{KL}$ is the Kullback-Leibler (KL) divergence measuring the dissimilarity between the learned distribution Q and training distribution P. In practice, to enable the backpropogation with stochastic gradient decent, the following reparameterization trick is often applied: first

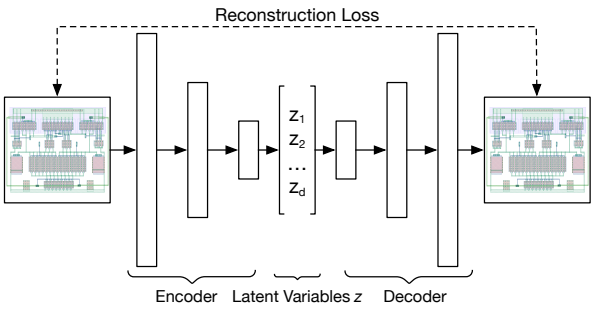


Fig. 4: Architecture of an autoencoder.

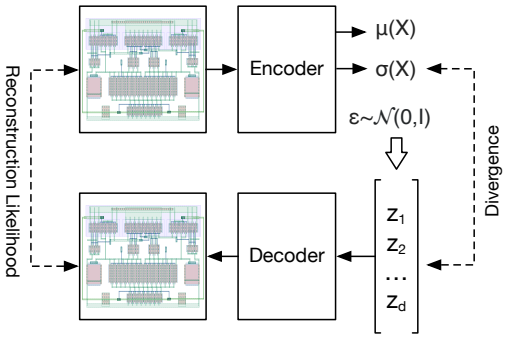


Fig. 5: Architecture of a VAE.

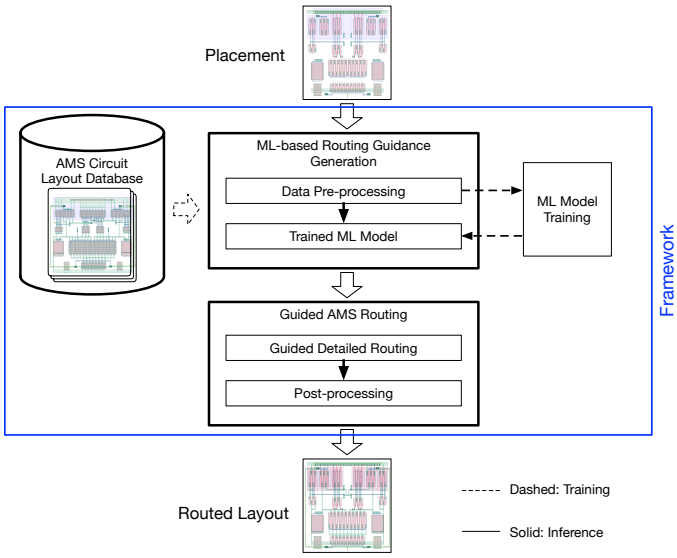


Fig. 6: The overall flow of GeniusRoute.

sample $\varepsilon \sim \mathcal{N}(0, I)$ and then compute $z = \mu(X) + \sigma^{1/2}(X) * \varepsilon$ [26]. In summary, Fig. 5 shows the architecture of a VAE. In a VAE structure, $\mu(z)$ and $\sigma(z)$ are trained by neural networks, and ε is sampled from a simple Gaussian distribution.

Autoencoder and VAE models often embed convolution layers in the neural networks to leverage the effectiveness of convolutional neural network (CNN) in computer vision applications [27].

III. The GeniusRoute Algorithm

In this section, we present the proposed GeniusRoute framework flow and detail the algorithms. Figure 6 shows the overall flow of GeniusRoute. The framework consists of two phases: training and

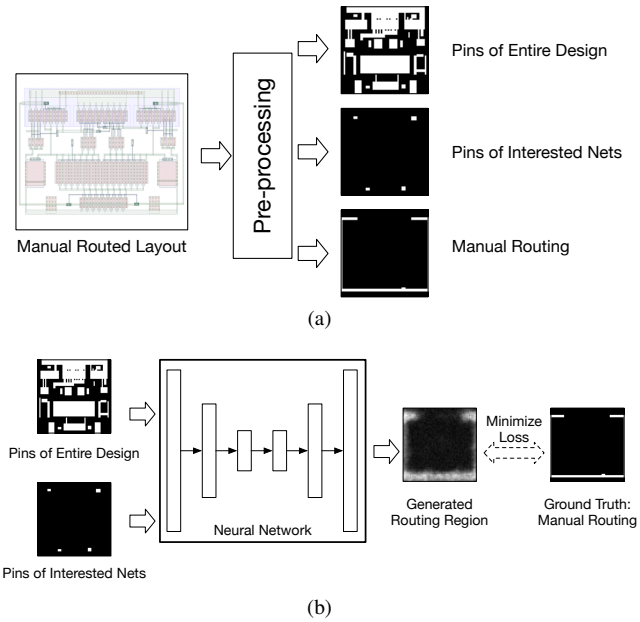


Fig. 7: Training phase. (a) Data pre-processing. (b) Model training.

inference. In the training phase, neural networks are trained to extract design expertise from manual layouts. The training phase consists of data pre-processing and model training. Due to the efficiency in image-based generative learning algorithms[27], GeniusRoute adopts images for representing placements and routing. In the data pre-processing stage, routing-relevant information are extracted from placement layouts into 2D images. Then the model training stage captures the human behaviors into machine learning models. Figure 7 shows the flow of the training phase. In the inference phase, the framework conducts machine-guided analog routing leveraging the trained machine learning models. The inference phase firstly pre-process the placement, then generates the routing probability map via trained models. The downstream AMS router routes the design following the probability map as guidance in the end. Figure 8 shows the flow of the inference phase.

The inference phase uses the model trained in the training model and takes the inputs defined in Sec. II-A to perform the analog routing. Instead of relying on the detailed guidelines or constraints, GeniusRoute attempts to generate human-like routing with minimum extra information from the inputs. To be specific, the symmetric nets and the types of nets are assumed to be more "obvious" to the designers than the guidelines on the physical layouts and can be potentially identified by existing constraint generation algorithms such as [5].

GeniusRoute consists of three tasks: (1) layout pre-processing to extract the data for neural network model (Sec. III-A), (2) model training to learn human layout approaches and generate routing guidance (Sec. III-B) and (3) performance-driven analog routing framework by mimicking manual layouts (Sec. III-C). The details of the three tasks will be discussed in the rest of this section.

A. Data Representation and Pre-processing

Data pre-processing abstracts the routing and placement and extracts the routing-relevant information for machine learning models. To effectively learn the relation between routing and placement, a good strategy of layout-image conversion is needed.

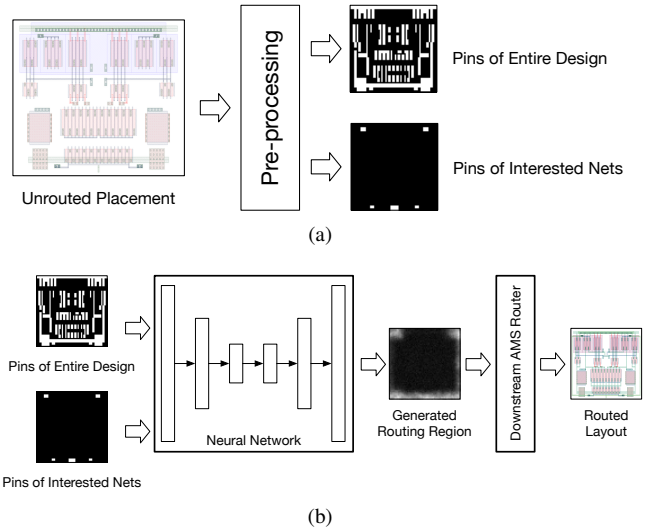


Fig. 8: Inference phase. (a) Data pre-processing. (b) Model inference.

Extracting routing from layouts into images is relatively straightforward; the regions that metal interconnections of the given nets lay in can be easily converted into 2D images. On the other hand, the information from placements that affects the routing decisions is latent and needs additional definitions.

1) Placement Data Representation: In order to infer the routing probability map, placement data representation needs to capture the concise "pins" for the input nets and also a high-level global view of the whole placement. In GeniusRoute, the global view is captured by the "pins" of all the nets in the entire design. This allows the trained model to consider the routing of other nets, which might largely impact the routing decisions. In summary, for each data point, we extract the pins for the entire design and the pins for the given nets and map them into two separate channels of an image. In all experiments, the image size for each channel is selected to be 64×64.

However, unlike the standard cell-based digital routing, the concept of "pins" is ambiguous in customized analog circuits. For example, layout designer may choose to have a common "pin" for multiple fingers of one device (Fig. 9(a)) or connect every finger separately (Fig. 9(b)). To generalize the methodology in extracting the starting points of routing, we propose the following strategy:

1) The first metal layer (M1) shapes overlapping with contact window (CO) shapes are pins. CO layer is used as contacts between interconnection metals and oxide diffusion (OD) or poly-silicon (PO). The pins identified with this strategy in practice are the terminal points of the metal interconnection for typical transistor and resistor devices.
2) Metal oxide metal capacitor uses metal layers as the terminals for routing and are labelled by hands in our experiments.
3) Nets sometimes have additional ports connecting the external system and are labelled by hands in our experiments.

2) Data Pre-processing: Data processing step takes the layout, list of nets of interest, and additional labeled pins as inputs, and outputs placement and routing information as images for the downstream machine learning models.

The layouts in both training and inference phases are in GDSII format. GDSII represents the layout as shapes in different layers, e.g., metal and via layers. To identify the nets of interest in the layout, we label a text of net name on the layout for each net of interest. Since

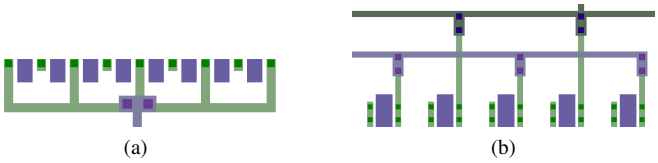


Fig. 9: Examples of pins of transistors. (a) Combined pin for fingers. (b) Separated pin for fingers.

the standard layout flow requires all IO nets being labelled to pass Layout Versus Schematic, the number of nets that needs additional manual labeling is reasonably small.

Algorithm 1 describes the main steps of data pre-processing. Firstly, the layouts are read, and a disjoint set is constructed with each shape as an individual set. An R-tree is also built for fast geometrical querying of the shapes and text labels (line 1-7). Then the connectivity of all shapes is explored by querying the overlapping relations between shapes. The overlapping shapes will be unioned in the disjoint set (line 8-11). During the process, the text labels are also be investigated, and the net names are assigned to the disjoint sets (line 12-14). After the shape clustering and labeling, the pins and routing segments are aggregated in channels and exported into desired images (line 15-18).

Algorithm 1 Layout Data Pre-Processing

Input:
Layout L
List of explicitly labeled pins P
List of interested nets N
Output:
Pins of entire design CH_1
Pins of interested nets CH_2
Routing of interested nets CH_3
1: $S \leftarrow$read_layout_shapes(L)
2: $R \leftarrow$ R-tree
3: $D \leftarrow$ Disjoint_set
4: **for all** s in S **do**
5: **if** s is metal, via, CO, PO or pin label **then**
6: R.insert(s)
7: D.make_set(s)
8: **for all** s in S **do**
9: **if** s is metal, via, CO or PO **then**
10: $S^* \leftarrow$ Query shapes overlapping with s in R
11: Union the sets for s and S^*
12: **if** s is net label **then**
13: $S^* \leftarrow$ Query shapes overlapping with s in R
14: Label the sets of S^* to be net $s.text$
15: Add all $p \in P$ to CH_2
16: Save all pins to CH_1
17: Save all pins in $\{d \in D : d.net \in N\}$ to CH_2
18: Save all metals in $\{d \in D : d.net \in N\}$ to CH_3

When exporting the images, we apply Gaussian blurring on the images (Fig. 10) to remove unwanted details for two reasons: encouraging the models to focus on the routing regions rather that the exact metal shapes and improving the model accuracy. We choose Gaussian kernel sizes to be 17×17 for routing and 5×5 for placement in all experiments.

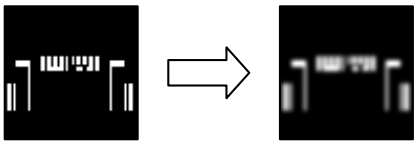


Fig. 10: Gaussian blurring.

B. Machine Learning-based Routing Guide Generation

Predicting the routing has two major challenges: the routing problem is difficult, and labeled data are limited and costly to obtain. To efficiently learn a general routing strategy and overcome the shortage of labeled data, we refine the objective of our learning task and use semi-supervised training methods.

1) Learning Tasks: We propose to learn the probability map of routing with multiple models.

Probability Map: While generative learning model is for learning probabilistic distributions, routing is a discrete and highly-constrained problem. Directly trying to reproduce routing through machine learning is difficult. Furthermore, good routing solutions may not be unique. It is preferred to make routing predictions based on a general strategy rather than following a particular example the model has learned. In other words, hoping the machine learning model to produce an unique routing solution is not fitting to the practice. Hence, we propose to predict the probability map that models the routing likelihoods in each region.

Special Net Types: Compared to training a general model for different applications, using different models for reduced tasks simplifies the problems and is more efficient in learning. Knowing different net functionality has different routing strategy a priori, GeniusRoute trains different models for different net types. Within the scope of our training and testing data, three major special net types are chosen for learning: (1) **differential nets**, (2) **clocks** and (3) **power and ground (PG)**.

During learning, different models adopt different strategies based on the practical problem of the net type. Differential nets are varied in scope and the combined "sensitive" region is more important for the downstream router. Therefore, in GeniusRoute, all nets belonging to the type are combined in learning, and the model predicts the routing region for the whole class of nets. On the other hand, we use single power or ground for the PG model to learn human behaviors in planning power and ground lines.

2) Semi-supervised Machine Learning Algorithm: We propose a semi-supervised machine learning algorithm to predict the routing probability. The algorithm consists of two parts, unsupervised and supervised learning. This subsection first presents the neural network architecture. Then we introduce the data augmentation technique being used to improve the data efficiency. Finally, the semi-supervised training procedure is explained in details.

Neural Network Architecture To avoid overfitting to limited training data, we choose small and simple encoder and decoder networks. The dimension of the latent variable vector defined in Sec. II-D is kept small and set to 32. Table I shows the detailed architecture of the neural network. The ReLu activation layers after each convolution (conv), deconvolution (deconv), and fully connected (FC) layers are omitted for simplicity. Since the learning objectives are different in the unsupervised training phase (Stage 1) and supervised training (Stage 2 & 3) phase, there is a minor difference in the structure of the decoder networks. This difference will be explained in details below.

Data Augmentation: GeniusRoute applied data augmentation to improve the data efficiency [27]. Data are augmented by 1) flipping horizontally, 2) flipping vertically, and 3) rotating 180°.

TABLE I: Network configurations.

Stage 1 Configuration		Stage 2 & 3 Configuration
Input ($2 \times 64 \times 64$ image)		Input ($2 \times 64 \times 64$ image)
conv/$5 \times 5 \times 64$		conv/$5 \times 5 \times 64$
conv/$5 \times 5 \times 128$		conv/$5 \times 5 \times 128$
FC/64		FC/64
Latent Variables (32)		Latent Variables (32)
FC/$16 \times 16 \times 64$	FC/$16 \times 16 \times 64$	FC/$16 \times 16 \times 64$
deconv/$4 \times 4 \times 32$	deconv/$4 \times 4 \times 32$	deconv/$4 \times 4 \times 32$
deconv/$4 \times 4 \times 1$	deconv/$4 \times 4 \times 1$	deconv/$4 \times 4 \times 1$
Output ($2 \times 64 \times 64$ image)		Output ($1 \times 64 \times 64$ image)

Training data are labeled in the three aforementioned special net types. After the data augmentation, the numbers of labeled data points for clock, differential nets, and power/ground nets are 168, 128, 256 respectively. On the other hand, the number of unlabeled data points is 6360 after data augmentation.

Semi-supervised Training Algorithm: Inspired by [28], we use the semi-supervised training strategy as shown in Fig. 11 to leverage the unlabeled data. The training procedure consists of three stages:

1) Unsupervised initial feature extraction. The encoder and decoder networks are initialized with Gaussian random weights.
2) Supervised decoder training. We initialize encoder network with the pre-trained encoder network and the decoder network with Gaussian random initialization. The encoder weights are fixed, and only the decoder is trained for this stage.
3) Model fine-tuning. We fine-tune both the encoder and decoder networks together with a reduced learning rate.

Stage 1 (Fig. 11(a)) learns an encoder network to effectively extract the latent feature variable in an unsupervised fashion. This allows us to fully exploit unlabeled data. The encoding network extracts features from both pins of the target nets and its corresponding layout placement. The decoding network outputs the two corresponding reconstructions of the input images. The objective of this standard VAE is to maximize

$$logP(X|z) - \mathcal{D}_{KL}[Q(z|X)||P(z)].$$

With such objective, the model is trained such that at the bottleneck level between encoder and decoder, the latent variable vector contains enough compact information to reproduce the image. Hence it can be used as a feature vector representing the inputs. The goal of this stage is to extract important features of pre-processed placements using unlabeled data and provides a generalized initial points for the following supervised learning.

Stage 2 (Fig. 11(b)) trains a generative network to predict routing for different net types. Distinct model is trained for each net type on the labeled data. In this stage we keep the encoding network fixed to the pre-trained network in the unsupervised feature extraction stage. The generative decoder network outputs a single image as the routing probability prediction. The objective is to minimize the $\mathcal{L}^2$ norm of the distance between ground truth Y and inferred output $\hat{Y}$, i.e.,

$$||Y - \hat{Y}||_2.$$

Stage 3 (Fig. 11(c)) fine-tunes the entire model to achieve better accuracy. Since the network is already close to a nearly optimal point, we set our learning rate much lower than that in Stage 1 and 2. The objective is to maximize

$$logP(Y|z) - \mathcal{D}_{KL}[Q(z|X)||P(z)].$$

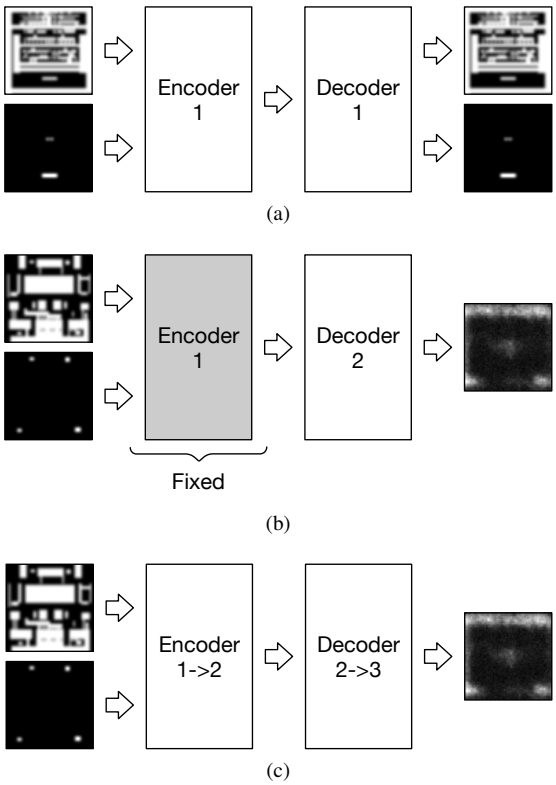


Fig. 11: Tree-stage training algorithm. (a) Stage 1: unsupervised initial feature extraction. (b) Stage 2: supervised decoder training. (c) Stage 3: model fine-tune with reduced learning rate.

Intuitively, the unsupervised training stage extracts placement and net pin features from the entire unlabeled data set. The feature extraction generalizes the extracted features and avoids overfitting to the small labeled data set. The supervised decoder training allows the model to specialize in the prediction for different net types. The fine tuning stage with reduced learning rate allows small perturbations to the encoder and decoder for higher prediction accuracy. Through this approach, we can pertain the knowledge learned from unsupervised learning while achieving higher accuracy in predicting the routing guidance for different net types.

C. Guided Analog Detailed Routing

In the inference phase, GeniusRoute adopts the A^* search algorithm for detailed routing. It leverages the routing guidance R generated by machines models to make routing decisions. Our analog routing flow consists of two steps: detailed routing and post-processing.

1) Detailed Routing: Detailed routing routes all nets via A^* search, honoring symmetric constraints and input routing guidance. Our approach is summarized as follows:

1) Large pins are split into sets of searching points at the intersections of routing tracks with pins.
2) Each multi-pin net is decomposed into a set of 2-pin nets by Minimum Spanning Tree.
3) A^* search is applied to connect the 2-pin nets in a sequential manner. During each search, routing guidance R is honored via penalties in the cost function. Symmetric constraints are enforced by mirroring the nets. In addition, when searching paths for nets with routing guidance, it is non-trivial to estimate the heuristic costs in the A^* search routine. Therefore the heuristic costs are set to zero in such cases to avoid giving preference in the search direction. In other words, we de facto route the nets with guidance in a Maze routing fashion.
4) A negotiation-based rip-up and reroute scheme is implemented to ensure feasibility.

We embed the routing guidance as cost functions in A^* search, together with other common routing objective such as wire length and penalty of vias. The cost from the routing guidance is composed of two parts: the penalty of violating the guidance (violating cost), and the cost of routing in the region of other nets demand (competition cost). Violating cost is a monotonic decreasing function of probability map, i.e., $Cost_{violate} = f(r^n_{i,j})$. In this paper we choose f to be $f(x) = a + \frac{b}{2^{(x/c)}}$. Competition cost is the penalty of routing the net in a region demanded by other nets in routing guidance and is determined by the difference of $r^n_{i,j}$ and the average routing probability $\bar{r}_{i,j}$, i.e. $Cost_{compete} = g(\bar{r}_{i,j} - r^n_{i,j})$. We choose $g(x) = max(d \cdot x, 0)$. In summary, the proposed cost function is $Cost = Cost_{wire} + Cost_{VIA} + Cost_{history} + Cost_{violate} + Cost_{compete}$.

2) Post-Processing for Power Delivery Network: PG routing is further polished in a post-processing step. Additional connection between pins for V_{DD} and V_{SS} nets are routed based on the routing guidance.

The post-processing consists of following steps:

1) Potential pin connections are identified via breadth-first search on the routing probability map R. Pairs of pins are considered as candidates if there is a confident path between them on R.
2) Candidates are pruned by removing the pairs that were routed previously.
3) Pins are routed via A^*-based detailed routing routine.

After the detailed routing and post-processing, the routed layout is exported into GDSII format.

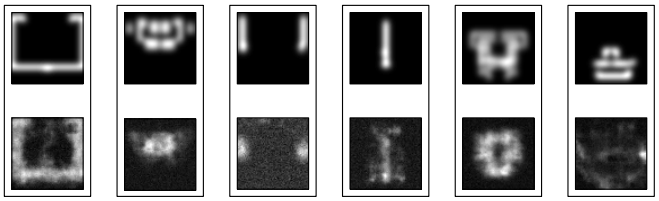
IV. Experimental Results

We implemented the proposed routing guide generation in Python based on Tensorflow [29], and the data pre-processing and detailed routing algorithm were programmed in C++. All experiments were performed on a Linux workstation with Intel 3.4GHz i7-3770 CPU and Nvidia GTX1080 GPU with 32GB memory.

As discussed in Sec. III-B, we collect both unlabeled and labeled human layouts for training data. The labeled data are categorized into differential nets, clocks and PG nets from component-level analog circuits. To be specific, the labeled data are all collected from comparator and operational amplifier (OpAmp) designs while the unlabeled data include a variety of component-level circuits from multiple mixed-signal system designs. Comparators and OpAmps are among the most representative analog components in mixed-signal systems such as data converters.

A. Experimental Results on Learning Models

We first conduct experiments on the trained neural network models. Figure 12 shows output examples of the model inference on testing sets. Among the results, first three (Fig. 12 (a)-(c) are manual placements, while the last three (Fig. 12 (d)-(f)) are automatic placements from [30]. Figure 12 (a) and (c) are outputs of the clock model, figure 12 (b) and (e) are from differential nets model, and figure 12 (c) and (f) are from PG model. The machine learning models not only learn well in manual placements, but are also capable of generating reasonable outputs for machine-generated placement for clock and differential nets. However, the PG model fails to predict



(a) (b) (c) (d) (e) (f)

Fig. 12: Example of inferences of testing set. The upper row shows the ground truth, and the lower row shows the inference.

the routing for automatic placement (Fig. 12(f)). The PG pins in automatic placement have different patterns from the manual layouts of the training data; hence the model had not seen an example of routing with similar placement and failed to make reasonable inference.

A major challenge of training a general models for routing probability is the lack of training data. As discussed in Sec. III-B, GeniusRoute attempts to learn a relatively rough routing probability instead of detailed routing implementation using small network models. In the scale of experiments, routing PG nets incorporates more detailed considerations such as IR drop than routing clocks and differential nets. Thus with limited number of training data, learning a general routing strategy for PG nets is intuively and empircally a more difficult task.

B. Experimental Results on GeniusRoute Framework

To evaluate our proposed framework, we conducted experiments on two analog circuits placements by experienced designers, a comparator (COMP1) and two-stage miller-compensated operational transconductance amplifier (OTA). To further validate the generality of our machine learning models, we also tested our framework on a machine-generated placement [30] (COMP2) with the same schematic of COMP1. We conducted our experiments in TSMC 40nm process. After routing the circuits, we used Calibre PEX to extract parasitic RC and coupling capacitance and used Cadence ADE to perform post-layout simulations.

TABLE II: Runtime. (seconds)

	W/o guide	This work
COMP1	11.7	26.2
COMP2	2.3	21.3
OTA	49.6	55.4

TABLE III: Runtime breakdown. (seconds)

	PP	MI	DR
COMP1	<1	17.6	8.6
COMP2	<1	17.6	3.7
OTA	3.5	13.2	38.7

Table II shows the run time of our proposed framework. "W/o guide" denotes the runtime of our detailed analog router alone without routing guidance. "This work" refers to our proposed framework. The runtime for the proposed framework includes all steps in the inference phase, and Table III shows the runtime breakdown. "PP", "MI" and "DR" denotes data pre-processing, model inference and detailed routing correspondingly. To avoid outliers, we did ten experiments on model inference and took the average time as the runtime for each execution. The reported model inference runtime in each experiment is calculated by multiplying the model inference execution time by the number of model inference operations needed. Model inference execution times are treated as the same for the three models since they employ similar network structures. Note that the majority time of the model inference is on loading the model parameters. While data pre-preprocessing and detailed routing runtime scales with circuit size, the model inference time is irrelevant to the layout size and only depends on the number of inferences needed.

For comparison, we implement the algorithm in [11] with the following modification: (1) Some of our pins are inside active regions and forbidding routing over active regions is infeasible for some nets. Thus, we assign a substantial penalty for routing over active regions instead of setting a hard constraint. (2) As the authors [11] do not explicitly state their strategy in choosing the maximum allowed parallel run length, we choose to avoid routing two nets with spacing within 1μm in parallel for more than 2μm; (3) Instead of using only M1 and M2, we used M1-M3 for routing because some of the pins in experiments are on the M3 layer.

TABLE IV: Comparison of post-layout simulation results for COMP1.

	Schematic	Manual	[11]	W/o guide	This work
Offset (μV)	/	480	1230	2530	830
Delay (ps)	102	170	180	164	163
Noise (μV_{rms})	439.8	406.6	437.7	439.7	420.7
Power (μW)	13.45	16.98	17.19	16.82	16.80

TABLE V: Comparison of post-layout simulation results for COMP2.

	Schematic	Manual	[11]	W/o guide	This work
Offset (μV)	/	550	350	1180	280
Delay (ps)	103	196	259	235	241
Noise (μV_{rms})	439.8	380.0	383.6	369.6	367.8
Power (μW)	13.45	20.28	20.17	20.23	20.15

Table IV and V show the post-layout simulation for the two comparator layouts, COMP1 and COMP2. "Offset" denotes the input-referred offset. "Delay" is the output delay measured with 500 μV differential input. "Noise" is the input-referred noise referred measured with 550 mV common mode input. "Power" is measured with 500μV differential input voltage, 200 MHz clock and 1.1 V supply voltage. In both two experiments, we achieve comparable performance to manual routing. Our work consistently outperform [11] in all performance metrics. Compared to the results without routing guidance, we achieve 67% and 76% reduction in input-referred offset, with comparable or better results in other metrics. We observed that [11] results in lower input-referred offset and higher output delay compared to our baseline "W/o guide". It might be caused by the avoidance of routing over active regions, which reduce critical nets coupling to sensitive devices but introduce extra parasitics.

TABLE VI: Comparison of post-layout simulation results for OTA.

	Schematic	Manual	[11]*	W/o guide	This work
Gain (dB)	38.20	37.47	43.60	36.61	37.36
PM (°)	64.66	72.46	29.97	94.68	76.40
Noise (μV_{rms})	222.0	223.7	278.8	292.7	224.8
Offset (mV)	/	0.88	2.49	3.21	0.39
CMRR (dB)	/	59.61	29.97	58.52	59.15
BW (MHz)	110.5	102.5	92.4	232.1	107.3
Power (μW)	776.93	757.35	528.11	715.11	787.82
* Without active region avoidance					

Table VI shows another experiment of OTA. "Gain" refers to DC open loop gain. "PM" denotes phase margin. "Noise" and "Offset"

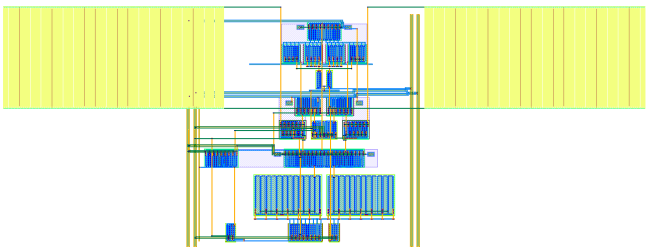


Fig. 13: GeniusRoute layout of OTA.

are input-referred noise and input-referred offset. "BW" abbreviates for unity-gain bandwidth. "Power" is the DC power measured with 1.1 V supply voltage. In the table, we show the simulation results of [11] without routing over active regions constraint. Routing over active regions brings extra parasitic capacitance to sensitive nodes and causes failure in common-mode feedback. As a result, the OTA becomes dysfunctional. Compared to manual layout, our routing achieves similar performance, with minor degradation in phase margin and a slight improvement in input-referred offset. On the other hand, "W/o guide" result has significant shifts from the schematic in phase margin and unity-gain bandwidth and notable drop in performance of DC gain and input-referred offset. Figure 13 shows the resulting layout of OTA.

V. Conclusion

In this paper, we present a new methodology in analog IC routing by automatically learning human behaviors in manually routed layouts. GeniusRoute, a performance-driven analog routing framework, is proposed with routing guidance generation and automatic guided analog detailed routing. GeniusRoute proposes a new methodology of automatically extracting routing regions for different nets from human layouts and apply the learned knowledge into analog router. Experimental results show that our proposed framework produces performance close to manual design and outperforms a previous work of analog routing in component-level circuits.

Acknowledgement

This work is supported in part by the NSF under Grant No. 1704758, and the DARPA ERI IDEA program. The authors would like to thank Mohamed Baker Alawieh, Jiaqi Gu and Wuxi Li from The University of Texas at Austin for helpful comments and discussions.

References

[1] M. P. Lin, Y. Chang, and C. Hung, "Recent research development and new challenges in analog layout synthesis," in *ASPDAC*, Jan 2016, pp. 617–622.

[2] R. A. Rutenbar, "Analog circuit and layout synthesis revisited," in *ISPD*, 2015, pp. 83–83.

[3] J. Crossley, A. Puggelli, H. . Le, B. Yang, R. Nancollas, K. Jung, L. Kong, N. Narevsky, Y. Lu, N. Sutardja, E. J. An, A. L. Sangiovanni-Vincentelli, and E. Alon, "Bag: A designer-oriented integrated framework for the development of ams circuit generators," in *ICCAD*, Nov 2013, pp. 74–81.

[4] E. Chang, J. Han, W. Bae, Z. Wang, N. Narevsky, B. NikoliC, and E. Alon, "Bag2: A process-portable framework for generator-based ams circuit design," in *IEEE Custom Integrated Circuits Conference (CICC)*, April 2018, pp. 1–8.

[5] U. Choudhury and A. Sangiovanni-Vincentelli, "Constraint generation for routing analog circuits," in *DAC*, June 1990, pp. 561–566.

[6] E. Charbon, E. Malavasi, U. Choudhury, A. Casotto, and A. Sangiovanni-Vincentelli, "A constraint-driven placement methodology for analog integrated circuits," in *IEEE Custom Integrated Circuits Conference (CICC)*, vol. 28, 1992, pp. 1–4.

[7] J. M. Cohn, D. J. Garrod, R. A. Rutenbar, and L. R. Carley, "Koan/anagram ii: new tools for device-level analog placement and routing," *JSSC*, vol. 26, no. 3, pp. 330–342, March 1991.

[8] P. Lin, H. Yu, T. Tsai, and S. Lin, "A matching-based placement and routing system for analog design," in *International Symposium on VLSI Design, Automation, and Test (VLSI-DAT)*, April 2007, pp. 1–4.

[9] H. Ou, H. C. Chien, and Y. Chang, "Non-uniform multilevel analog routing with matching constraints," in *DAC*, June 2012, pp. 549–554.

[10] P. Pan, H. Chen, Y. Cheng, J. Liu, and W. Hu, "Configurable analog routing methodology via technology and design constraint unification," in *ICCAD*, Nov 2012, pp. 620–626.

[11] L. Xiao, E. F. Y. Young, X. He, and K. P. Pun, "Practical placement and routing techniques for analog circuit designs," in *ICCAD*, Nov 2010, pp. 675–679.

[12] C. Wu, H. Graeb, and J. Hu, "A pre-search assisted ilp approach to analog integrated circuit routing," in *ICCD*, Oct 2015, pp. 244–250.

[13] H. Chi, H. Tseng, C. J. Liu, and H. Chen, "Performance-preserved analog routing methodology via wire load reduction," in *ASPDAC*, Jan 2018, pp. 482–487.

[14] Q. Gao, Y. Shen, Y. Cai, and H. Yao, "Analog circuit shielding routing algorithm based on net classification," in *ISLPED*, Aug 2010, pp. 123–128.

[15] H. Ou, H. Chang Chien, and Y. Chang, "Simultaneous analog placement and routing with current flow and current density considerations," in *DAC*, May 2013, pp. 1–6.

[16] J.-W. Lin, T.-Y. Ho, and I. H.-R. Jiang, "Reliability-driven power/ground routing for analog ics," in *ACM TODAES*, vol. 17, no. 1. New York, NY, USA: ACM, 2012, pp. 6:1–6:26.

[17] R. Martins, N. Lourenço, A. Canelas, and N. Horta, "Electromigration-aware and ir-drop avoidance routing in analog multiport terminal structures," in *DATE*, March 2014, pp. 1–6.

[18] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *DAC*, 2019, pp. 106:1–106:6.

[19] Y. Cao, A. B. Kahng, J. Li, A. Roy, V. Srinivas, and B. Xu, "Learning-based prediction of package power delivery network quality," in *ASP-DAC*, 2019, pp. 160–166.

[20] M. B. Alawieh, Y. Lin, Z. Zhang, M. Li, Q. Huang, and D. Z. Pan, "Gan-sraf: Sub-resolution assist feature generation using conditional generative adversarial networks," in *DAC*, 2019, pp. 149:1–149:6.

[21] W. Ye, M. B. Alawieh, Y. Lin, and D. Z. Pan, "Lithogan: End-to-end lithography modeling with generative adversarial networks," in *DAC*, 2019, pp. 107:1–107:6.

[22] B. Xu, Y. Lin, X. Tang, S. Li, L. Shen, N. Sun, and D. Z. Pan, "Wellgan: Generative-adversarial-network-guided well generation for analog/mixed-signal circuit layout," in *DAC*, 2019, pp. 66:1–66:6.

[23] H. Yang, P. Pathak, F. Gennari, Y.-C. Lai, and B. Yu, "Deepattern: Layout pattern generation with transforming convolutional auto-encoder," in *DAC*, 2019, pp. 148:1–148:6.

[24] P. Wu, M. P. Lin, and T. Ho, "Analog layout synthesis with knowledge mining," in *European Conference on Circuit Theory and Design (ECCTD)*, Aug 2015, pp. 1–4.

[25] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[26] C. Doersch, "Tutorial on variational autoencoders," in *arXiv preprint arXiv:1606.05908*, 2016.

[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[28] Y. Zhang, K. Lee, and H. Lee, "Augmenting supervised neural networks with unsupervised objectives for large-scale image classification," in *International Conference on Machine Learning (ICML)*, 2016, pp. 612–621.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[30] B. Xu, S. Li, C.-W. Pui, D. Liu, L. Shen, Y. Lin, N. Sun, and D. Z. Pan, "Device layer-aware analytical placement for analog circuits," in *ISPD*, 2019, pp. 19–26.