Reinforcement Learning Guided Detailed Routing for Custom Circuits

Hao Chen
The University of Texas at Austin,
Austin, TX, USA
haoc@utexas.edu

Kai-Chieh Hsu
Princeton University,
Princeton, NJ, USA
kaichieh@princeton.edu

Walker J. Turner
NVIDIA Corporation,
Jacksonville, FL, USA
wturner@nvidia.com

Po-Hsuan Wei
NVIDIA Corporation,
Santa Clara, CA, USA
pohsuanw@nvidia.com

Keren Zhu
The University of Texas at Austin,
Austin, TX, USA
keren.zhu@austin.utexas.edu

David Z. Pan
The University of Texas at Austin,
Austin, TX, USA
dpan@ece.utexas.edu

Haoxing Ren
NVIDIA Corporation,
Austin, TX, USA
haoxingr@nvidia.com

ABSTRACT

Detailed routing is the most tedious and complex procedure in design automation and has become a determining factor in layout automation in advanced manufacturing nodes. Despite continuing advances in custom integrated circuit (IC) routing research, industrial custom layout flows remain heavily manual due to the high complexity of the custom IC design problem. Besides conventional design objectives such as wirelength minimization, custom detailed routing must also accommodate additional constraints (e.g., path-matching) across the analog/mixed-signal (AMS) and digital domains, making an already challenging procedure even more so. This paper presents a novel detailed routing framework for custom circuits that leverages deep reinforcement learning to optimize routing patterns while considering custom routing constraints and industrial design rules. Comprehensive post-layout analyses based on industrial designs demonstrate the effectiveness of our framework in dealing with the specified constraints and producing sign-off-quality routing solutions.

CCS CONCEPTS

• **Hardware → Wire routing**.

KEYWORDS

Physical design, full-custom layout, detailed routing, reinforcement learning, graph neural networks.

ISPD '23, March 26–29, 2023, Virtual Event, USA

ACM ISBN 978-1-4503-9978-4/23/03…$15.00
https://doi.org/10.1145/3569052.3571874

ACM Reference Format:
Hao Chen, Kai-Chieh Hsu, Walker J. Turner, Po-Hsuan Wei, Keren Zhu, David Z. Pan, and Haoxing Ren. 2023. Reinforcement Learning Guided Detailed Routing for Custom Circuits . In *Proceedings of the 2023 International Symposium on Physical Design (ISPD '23), March 26–29, 2023, Virtual Event, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3569052.3571874

1 INTRODUCTION

Detailed routing is extremely intricate, time-consuming, and the most complicated procedure in modern layout automation flows and thus has become a determinant factor for enabling automation in advanced technology nodes. Recent research on detailed routing for digital integrated circuits (ICs) shows promising achievements in fulfilling real-world physical design demands [26, 13, 12]. However, despite being broadly investigated in the past decades, detailed routing for custom designs, especially in the analog/mixed-signal (AMS) domain, remains heavily manual in current industrial custom layout flows. This shortcoming is mainly caused by the sensitive nature and high complexity of custom circuits [7].

Lacking simple proxy design objectives and comprehensive performance modeling methods across a wide range of custom circuits, a custom detailed router cannot adopt specialized layout strategies for specific circuit classes like human layout experts. To overcome this issue, circuit designers often abstract design aspects for performance concerns (e.g., current balancing, parasitics reduction) into geometrical and electrical constraints for efficiency [15]. A robust custom detailed router should handle these additional specified constraints (e.g., path-matching, symmetry) on top of the conventional objectives such as wirelength optimization and design-rule-checking (DRC) to produce high-quality routing solutions that can pass sign-off checks. Failing to satisfy the constraints can lead to a drastically degraded post-layout performance [28, 19]. Figure 1 illustrates an example of a cross-coupled differential net pair specified with path-matching constraints on source and sink terminals. Though the routing solution in Figure 1(a) has a more optimized total wirelength, the solution in Figure 1(b) minimizes source-to-sink

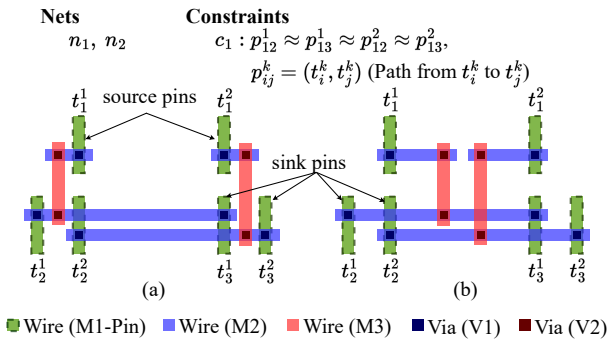

Figure 1: Example of different detailed routing solutions. (a) A solution with more optimized wirelength. (b) A solution with better matching between paths p_{12}^1, p_{13}^1, p_{12}^2, and p_{13}^2.

path mismatches across the connecting pins, resulting in better balanced post-layout performance.

Recent progress in reinforcement learning (RL) has demonstrated great potential in various applications in the electronic design automation (EDA) domain, including parallel prefix circuit design [24], logic optimization [17], macro placement [22], Steiner tree construction [21], global routing [14], net-ordering [27], and standard cell generation [23]. However, no previous work has attempted to tackle the custom detailed routing core problem using RL techniques. In this work, we devise a novel reinforcement learning guided custom detailed routing framework capable of handling sophisticated design constraints and rigid design rules for industrial custom circuits. Highlights of this work are summarized as follows.

- An RL-guided custom detailed routing framework for advanced FinFET circuits is devised to generate sign-off-ready routing solutions by optimizing routing objectives while considering specified constraints and industrial design rules.
- A heterogeneous graph representation is presented to effectively model routing topologies and nearby physical structures of targeted nets.
- A rip-up and re-routing scheme integrated with well-trained RL policies is shown to optimize routed nets iteratively.
- The proposed RL-based methodology can easily adapt to future design constraints with minimal adjustment to the reward function, the graph representation of routing solutions, and the GNN architecture.
- Experimental results on advanced industrial circuits show that our framework outperforms previous work and achieves aligned or even better post-layout performance than highly optimized manual layouts.

2 RELATED WORK

Existing routing methodologies can be organized into three categories: 1) template-based methods, 2) simulation-based techniques, and 3) constraint-driven approaches.

Template-based methods leverage well-optimized manual design modules for implementation. In [10, 20], template-based routing considers design-specific concerns (e.g., diffusion sharing, interdigitation) that are hard to handle by automated procedures. This type of methodology can realize robust and high-performance layouts. However, it suffers from scalability issues as the complex templates require significant labor efforts, thus are more restricted in large-scale systems such as high-speed serializer/deserializer (SerDes) and phase-locked loop (PLL).

Simulation-based techniques perform sensitivity analyses to capture critical information (e.g., parasitics mismatch, current flow) of the current routing solution to ensure that the specification is met. In [1], simulations are integrated with the routing process to identify critical nets and matching information. The simulation-based methods provide accurate performance feedback and can be generalized to consider various performance metrics (e.g., phase margin, power dissipation) across circuit classes. Still, the long execution time and resource-hungry computations that come with the simulations make implementing large-scale systems impractical.

Constraint-based approaches abstract layout conventions and restrictions for circuit performance without sacrificing the routing scalability in practice and are thus widely adopted in existing custom routing studies. In [3], a maze routing algorithm supporting mirror-symmetry constraints is proposed. In [5], an integer linear programming (ILP) formulation is presented to tackle various forms of path-matching, including symmetry, common-centroid, topology-matching, and length-matching constraints. The work [16] further extends the concept of symmetry into four variants (i.e., mirror-symmetry, self-symmetry, cross-symmetry, and partial-symmetry) to satisfy aesthetic needs in custom detailed routing. However, solutions for sophisticated constraints (e.g., simultaneous path-matching in multiple nets, as shown in Figure 1) are still absent in current literature. Furthermore, traditional planar technologies [18] and advanced FinFET technologies [25] have different design rules and routing conventions, resulting in diverse constraint sets. Therefore, developing new algorithms that can cover more complex settings and easily generalize to unseen constraints is necessary for real-world physical design needs.

3 PRELIMINARIES

3.1 Custom Routing Constraints

To ensure performance and layout robustness, circuit designers typically follow geometrical and electrical constraints during the custom layout implementation procedure.

Matching is an essential concept in custom layouts since it is the foundation of several commonly geometrical constraints, such as variants of symmetry, common-centroid, and topology-matching constraints. Symmetry constraints are specified to route a matched net pair symmetrically along some joint horizontal or vertical axes. Variants of symmetry can be extended for more complicated situations. For example, the routing of a net pair with pins on both sides of the symmetry axis cannot be perfectly symmetric; otherwise, an overlap will occur when crossing the axis and thus require detours. Common-centroid constraints are specified for routing a matched net pair symmetrically regarding some common centers and are commonly adopted for nets attached to array structured cells. Topology-matching constraints are specified to produce structurally-similar routing patterns and are often applied in critical but asymmetric nets. All the constraints above follow the matching principle and can be formulated as *generalized path-matching constraints*, where a path $p_{ij}^k = (t_i^k, t_j^k)$ is a tuple that signifies

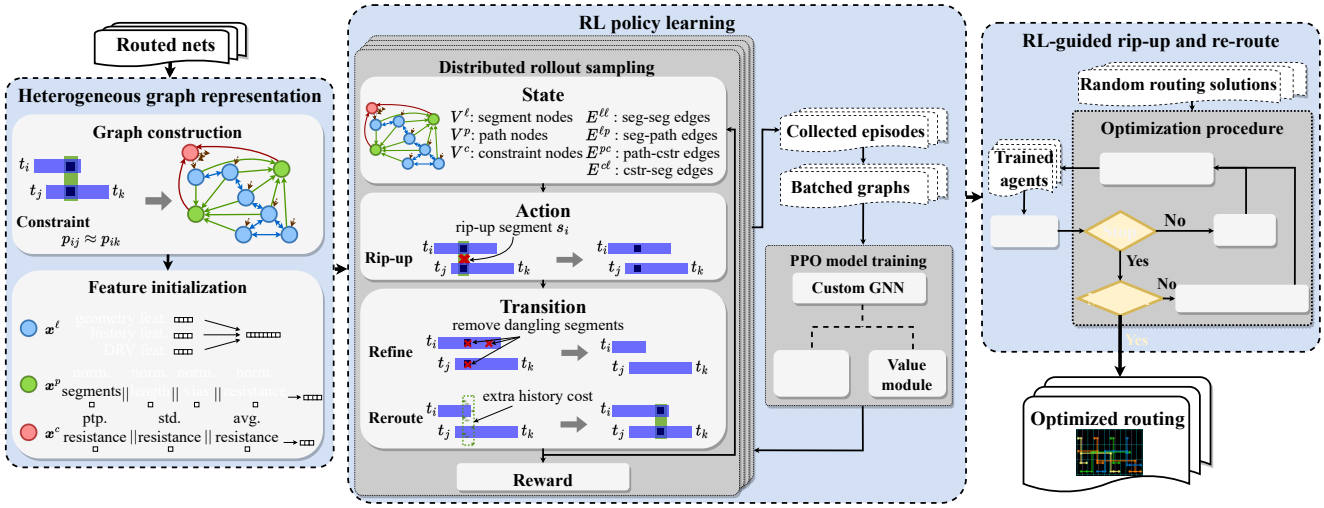

Figure 2: Computation flow of the proposed RL-guided custom detailed routing optimization framework.

the connection from pin t_i^k to pin t_j^k in net n_k. Generalized path-matching constraints minimize the mismatches between pin pairs in a group of nets. In this work, we focus on solving generalized path-matching constraints as they cover the most widely adopted geometrical constraints in real-world practice and beyond. Figure 1 gives an example of two crisscrossing nets where symmetry constraints could fail at handling the non-symmetric crossing of the Y-axis. However, general path-matching constraints can effectively describe the path-matching objective. We can achieve layout with better performance by specifying a generalized path-matching constraint such that p_{12}^1, p_{13}^1, p_{12}^2, and p_{13}^2 have similar path resistances, as shown in Figure 1(b).

Electrical constraints ensure that post-layout circuit performance is not limited by extra parasitics on critical nets along with path mismatches across critical signals. To efficiently handle these constraints, we pre-defined suitable wire widths and via templates on a non-uniform track configuration for each layer. The widths and via templates are determined according to simulation data and design rules. To reduce coupling for certain critical nets, shielding constraints are imposed by blocking signals to be routed on neighboring tracks.

3.2 Negotiation-Based Rip-Up and Re-Route

Rip-up and re-routing schemes are broadly adopted in global and detailed routing literature to eliminate design rule or electromigration (EM) violations. In [2], a negotiation-based rip-up and re-routing (NRR) method is presented and has become the mainstream technique used in state-of-the-art routers due to its superb ability to alleviate congestion. The core concept of NRR is to add additional *history cost* on routing grid edges if violations exist at the previous iteration. This way, the path-finding engine will avoid passing through previously violated regions as those are likely to be congested or resource-insufficient. For an edge e in the routing grid graph, the negotiation-based routing cost at the i^{th} rip-up and re-routing iteration is defined as follows in [2].

$$cost(e) = \left(cost_b(e) + cost_h^{(i)}(e)\right) \times penalty(e), \tag{1}$$

where $cost_b(e)$ is the base cost of routing through e, $cost_h^{(i)}(e)$ is the history cost of e at iteration i, and $penalty(e)$ denotes the congestion penalty, which is related to the number of other nets presently using e. At the end of the i^{th} iteration, the history cost $cost_h^{(i+1)}(e)$ for the next iteration is updated as follows.

$$cost_h^{(i+1)}(e) = \begin{cases} cost_h^{(i)}(e) + cost_{inc}, & \text{if } e \text{ has violations,} \\ cost_h^{(i)}(e), & \text{otherwise,} \end{cases} \tag{2}$$

where $cost_h^{(0)}(e) = 1$ and $cost_{inc}$ is the incrementing constant. Extensions and variants of negotiation-based cost functions have also been investigated to improve violation solving [4, 12]. In our custom detailed routing framework, we adopt a revised negotiation-based routing cost to encourage different routing solutions and reduce repetition. Instead of updating history costs for violated edges, we accumulate the costs for all edges occupied by routed nets. This technique reduces the chances of repeated solutions and allows RL agents to refine routing patterns in a shortened sequence of actions.

3.3 Problem Formulation

Problem 1 (Custom Detailed Routing)**.** Given a set of placed cells, a set of nets $N = \{n_i \mid 1 \le i \le |N|\}$, a non-uniform routing grid configuration, and a set of routing constraints (discussed in Section 3.1), generate a DRC-clean routing solution for each net $n_i \in N$ such that n_i is connected, the total routing cost is minimized, and the specified constraints are satisfied.

4 RL-GUIDED CUSTOM ROUTING FLOW

The overall computation flow of the proposed RL-guided custom detailed routing framework is shown in Figure 2. First, the routing solutions of each net in matching groups specified with path-matching constraints are transformed into a proposed heterogeneous graph representation. A feature initialization process is then executed to set the input features of all vertices regarding their types. After the graph initialization stage, we enter the core of our framework, which consists of two main phases: 1) RL policy learning, which distributively samples sequences of observations, actions, and the

Table 1: Notations used in this paper.

Symbol	Description
M	The set of all matching groups.
m_i	The i^{th} matching group in M.
N, T, P, C	The set of all nets, pins, paths, and path-matching constraints specified in the circuit netlist, respectively.
N_m, T_m, P_m, C_m	The set of nets, pins, paths, and path-matching constraints specified in matching group m, respectively.
n_i, t_i^k, c_i	The i^{th} net in N, i^{th} pin of the k^{th} net, and i^{th} path-matching constraint in C, respectively.
p_{ij}^k	The path formed by the connections between (t_i^k, t_j^k).
Φ_N	The current routing result of nets in N.
ℓ_i	The i^{th} segment in the same routing solution.
V^ℓ, V^p, V^c	The set of segment, path, and constraint nodes in the proposed heterogeneous graph, respectively.
$E^{\ell\ell}, E^{\ell p}, E^{pc}, E^{c\ell}$	The set of segment-to-segment, segment-to-path, path-to-constraint, and constraint-to-segment edges in the proposed heterogeneous graph, respectively.
v_i^ℓ, v_i^p, v_i^c	The i^{th} node in V^ℓ, V^p, and V^c, respectively.
$e_i^{\ell\ell}, e^{\ell p}, e^{pc}, e^{c\ell}$	The i^{th} edge in $E^{\ell\ell}$, $E^{\ell p}$, E^{pc}, and $E^{c\ell}$, respectively.

corresponding rewards for efficient data collection, then trains an RL policy maximizing routing rewards, and 2) RL-guided rip-up and re-routing, which integrates well-trained RL policies to determine suitable segment removal sequences to guide the backbone detailed router to optimize routing patterns. Table 1 summarizes the notations used in this paper.

4.1 Heterogeneous Graph Representation

Routing solutions can be formulated into graphs naturally since they are formed by connections of segments/wires. To model routing solutions for a group of nets associated with some path-matching constraints, we propose a heterogeneous graph representation for improved accuracy, efficiency, and complexity.

Given a routed matching group, we construct a heterogeneous graph $G = (V, E)$, where V is a set of vertices and E is a set of edges. The set V can be further divided into three subsets, namely V^ℓ, V^p, and V^c, as described in Table 1. Each vertex $v_i^\ell \in V^\ell$ maps to a routed segment (i.e., horizontal wire, vertical wire, or via). Each vertex $v_i^p \in V^p$ maps to a path in some path-matching constraints, and each vertex $v_i^c \in V^c$ associates to a specified path-matching constraint. The set E can also be split into four categories, including $E^{\ell\ell}$, $E^{\ell p}$, E^{pc}, and $E^{c\ell}$, as described in Table 1. A *seg-seg edge* $e^{\ell\ell} = (v_i^\ell, v_j^\ell) \in E^{\ell\ell}$, $v_i^\ell, v_j^\ell \in V^\ell$ signifies that the mapped segment of v_i^ℓ connects with that of v_j^ℓ. A *seg-path edge* $e^{\ell p} = (v_i^\ell, v_j^p) \in E^{\ell p}$, $v_i^\ell \in V^\ell$, $v_j^p \in V^p$ marks that the mapped segment of v_i^ℓ exists in the corresponding path of v_j^p. A *path-cstr edge* $e^{pc} = (v_i^p, v_j^c) \in E^{pc}$, $v_i^p \in V^p$, $v_j^c \in V^c$ indicates that the path of v_i^p is specified in the associated path-matching constraint of v_j^c. Finally, a *cstr-seg edge* $e^{c\ell} = (v_i^c, v_j^\ell) \in E^{c\ell}$, $v_i^c \in V^c$, $v_j^\ell \in V^\ell$ means that the segment of v_j^ℓ should be considered in the associated constraint of v_i^c. Algorithm 1 sketches the heterogeneous graph construction process. First, we initialize a graph and build the vertex sets from the input matching group and constraints (Lines 1-4). We then enumerate all routed segments to add seg-seg edges between the corresponding vertices of connected segments (Lines 6-9). Finally, we check through every constraint

Algorithm 1 *ConstructHeterogeneousGraph*(m, Φ_m)

Input: A matching group m, path-matching constraints C_m with paths P_m, and its routing solution Φ_m.
Output: The heterogeneous graph representation G.

1: Initialize a heterogeneous graph $G = (V, E)$;
2: $V^\ell := \{Vertex(\ell) \mid \forall \ell \in \Phi_m\}$;
3: $V^p := \{Vertex(p) \mid \forall p \in P_m\}$;
4: $V^c := \{Vertex(c) \mid \forall c \in C_m\}$;
5: $E^{\ell\ell} := \emptyset$; $E^{\ell p} := \emptyset$; $E^{pc} := \emptyset$; $E^{c\ell} := \emptyset$;
6: **for each** segment $\ell_i \in \Phi_m$ **do** ▷ build $E^{\ell\ell}$
7: **for each** segment $\ell_j \in \Phi_m$ and $j \neq i$ **do**
8: **if** *isConnected*(ℓ_i, ℓ_j) **then**
9: $E^{\ell\ell} := E^{\ell\ell} \cup \{(Vertex(\ell_i), Vertex(\ell_j))\}$;
10: **for each** constraint $c \in C_m$ **do** ▷ build $E^{\ell p}$, E^{pc}, and $E^{c\ell}$
11: **for each** path p that exists in c **do**
12: $E^{pc} := E^{pc} \cup \{(Vertex(p), Vertex(c))\}$;
13: **for each** segment ℓ that exists in p **do**
14: $E^{\ell p} := E^{\ell p} \cup \{(Vertex(\ell), Vertex(p))\}$;
15: $E^{c\ell} := E^{c\ell} \cup \{(Vertex(c), Vertex(\ell))\}$;
16: $V := V^\ell \cup V^p \cup V^c$;
17: $E := E^{\ell\ell} \cup E^{\ell p} \cup E^{pc} \cup E^{c\ell}$;
18: **return** G;

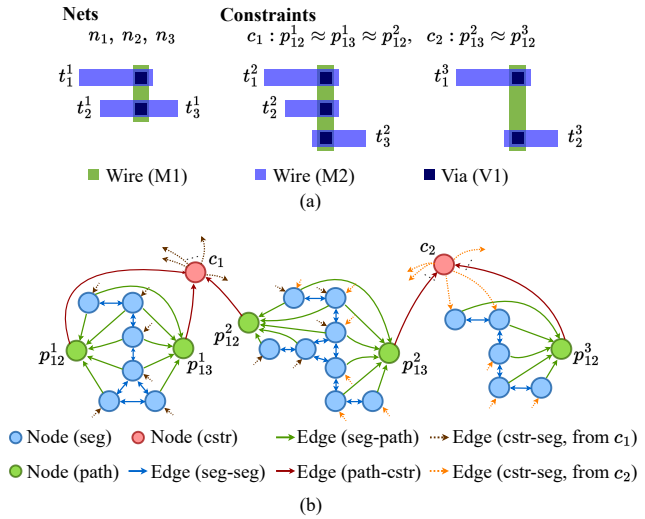

Figure 3: Example of the heterogeneous graph representation. (a) A routed matching group consists of nets n_1, n_2, and n_3 with path-matching constraints c_1 and c_2. (b) The corresponding heterogeneous graph representation of the group.

and its paths for constructing seg-path, path-cstr, and cstr-seg edges, then return the graph (Lines 10-18). Figure 3 illustrates the proposed graph representation for a matching group of three nets, and two path-matching constraints, where c_1 specifies that p_{12}^1, p_{13}^1, p_{12}^2 should be matched, and c_2 means that p_{13}^2 should match p_{12}^3.

After constructing the heterogeneous graph, we set the initial feature vector for each vertex. Table 2 summarizes the features of each node type and their dimension. For each segment node, we define a 20-dimensional feature vector, including the normalized endpoint coordinates, segment length, lower layer index, number of design rule violations (DRVs), history costs on neighboring routing grid

Table 2: Initial features of each vertex type in the proposed heterogeneous graph representation.

Type	Feature	Dim.
Segment	Norm. bottom-left coordinate	2
	Norm. top-right coordinate	2
	Norm. length	1
	Norm. lower layer index	1
	Norm. design rule violations	1
	Norm. history costs	11
	Boolean indicator for via	1
	Boolean indicator for terminal segment	1
Path	Norm. segment count	1
	Norm. length	1
	Norm. via count	1
	Norm. path resistance	1
Constraint	Norm. maximum difference of path resistances	1
	Norm. averge of path resistances	1
	Norm. standard deviation of path resistances	1

edges, and two additional Boolean indicators for vias and terminal segments (i.e., segments connected to pins). To calculate the history cost features for a vertical segment, we expand a bounding box whose width equals 11 pitch steps of vertical tracks (i.e., previous 5 tracks, following 5 tracks, and the current track) and height equals the segment length, then record the history cost summation on each track within the bounding box to form an 11-dimensional vector. The calculation for horizontal segments can be done similarly. The proposed formulation implicitly encodes the spatial distribution of history costs, which provides information for RL agents to understand the congestion status of the surrounding area. For each path node, we set an initial feature vector of 4 dimensions, including the normalized segment count, length, via count, and resistance of the path. While for each constraint node, we have the normalized maximum difference, average, and standard deviation of resistances of paths associated with the constraint as the initial feature vector.

4.2 RL Policy Learning

We model the custom detailed routing problem for a matching group m as a Markov decision process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, f_T, \gamma)$, where a state s in state space $\mathcal{S}$ is a heterogeneous graph representation for a routing solution (described in Section 4.1), an action a_k in action space $\mathcal{A}$ removes the k^{th} segment ℓ_k in the current solution while $a_0 \in \mathcal{A}$ terminates the decision process, $R\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function describing the desired pattern and routing constraints, $f_T : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is the transition function defining the distribution over next states given a state and an action, and $\gamma \in [0, 1]$ is the discount factor. In our implementation, f_T includes the underlying dangling wires refinement and re-routing and is unknown to the RL policy. We want to optimize a policy $\pi : \mathcal{S} \to \Delta\mathcal{A}$ such that the return is maximized as follows.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}_\pi \left[\sum_{i=0}^{\infty} \gamma^i R\left(s^{(i)}, a^{(i)}, s^{(i+1)}\right) \right], \tag{3}$$

where $\mathbb{E}_\pi$ denotes the expectation over $a^{(i)} \sim \pi(\cdot \mid s^{(i)})$, $s^{(i+1)} \sim f_T(\cdot \mid s^{(i)}, a^{(i)})$, and $s^{(i)}, a^{(i)}, r^{(i)}$ denotes state, action, and reward at timestep i. Starting from an initial state of a randomly routed

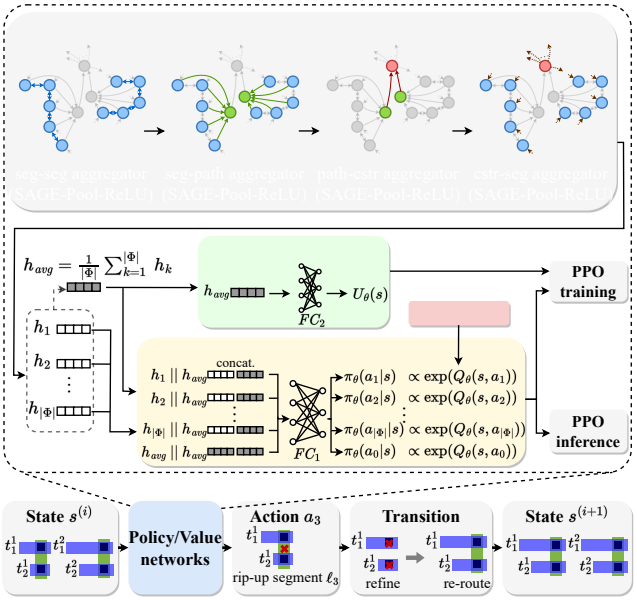

Figure 4: Network architecture for RL training and inference.

Table 3: Detailed parameters of the model in Figure 4.

Network	Layer	Input†	Output	Aggregator	Activation
GNN ($E^{\ell\ell}$)	SAGE	(20, 20)	128	pool	ReLU
	SAGE	(128, 128)	128	pool	ReLU
GNN ($E^{\ell p}$)	SAGE	(128, 5)	128	pool	ReLU
GNN (E^{pc})	SAGE	(128, 3)	128	pool	ReLU
GNN ($E^{c\ell}$)	SAGE	(128, 128)	128	pool	ReLU
Policy module	FC	256	256	-	ReLU
	FC	256	$\lvert\mathcal{A}\rvert$	-	-
Value module	FC	128	128	-	ReLU
	FC	128	1	-	-

† (*src*, *dst*) for GNNs represent the size of source and destination node features.

solution, we set the reward function as

$$\begin{aligned} r^{(i)} &= R\left(s^{(i)}, a^{(i)}, s^{(i+1)}\right) = \\ &w_{wl}\left(wl(s^{(i)}) - wl(s^{(i+1)})\right) + w_{via}\left(via(s^{(i)}) - via(s^{(i+1)})\right) + \\ &w_{ptp}\left(ptp(s^{(i)}) - ptp(s^{(i+1)})\right) + w_{drv}\left(drv(s^{(i)}) - drv(s^{(i+1)})\right), \end{aligned} \tag{4}$$

where $wl(s^{(i)})$ is the wirelength, $via(s^{(i)})$ is the via count, $drv(s^{(i)})$ is the total design rule violations, and

$$ptp(s^{(i)}) = \frac{1}{|C_m|} \sum_{c \in C_m} \max_{p_j, p_k \in c,\ j \neq k} \left| res(p_j) - res(p_k) \right| \tag{5}$$

is the average of maximum path resistance mismatch in each constraint in C_m. In practice, we set w_{wl}, w_{via}, w_{ptp}, and w_{drv} to 0.1, 0.1, 30, and 300, respectively, to encourage path-matching and penalize DRC violations. Note that the wirelength and via minimizing terms also help on path-matching since it encourages the agent to explore in a refined search space.

We train the policy by proximal policy optimization (PPO) [9], and the proposed neural network architecture is shown in Figure 4 and Table 3. Neural networks are parameterized by θ, which includes GNN for segment embeddings and fully-connected layers in

the value and policy modules. First, we generate segment embeddings by four sequential graph convolutions: seg-seg aggregation for $E^{\ell\ell}$, seg-path aggregation for $E^{\ell p}$, path-cstr aggregation for E^{pc}, and cstr-seg aggregation for $E^{c\ell}$. SAGE GNN layers [8] are used for feature aggregation on each edge type. The aggregation function for edges in $E^{\ell p}$ is shown as follows.

$$\begin{aligned} h_{\mathcal{N}(v)}^{(j+1)} &= pool\left(\{h_u^{(j)}, \forall u \in \mathcal{N}(v)\}\right), \\ h_v^{(j+1)} &= ReLU\left(W^{(j)} \cdot concat\left(h_v^{(j)}, h_{\mathcal{N}(v)}^{(j+1)}\right)\right), \end{aligned} \tag{6}$$

where $h_v^{(j)}$ denotes the hidden features of node v at the j^{th} layer, $\mathcal{N}(v)$ signifies the neighboring nodes of v, $pool(\cdot)$ is the element-wise max operator, $ReLU(\cdot)$ is the element-wise rectified linear unit activation function, $W^{(j)}$ is the linear transformation matrix at the j^{th} layer, and $concat(\cdot,\cdot)$ denotes the vector concatenation function. The aggregation for the remaining edge types are defined similarly.

After graph convolutions, we obtain the segment embeddings within the group and denote them by h_k, as in the second row of Figure 4. We average the segment embeddings to represent the overall graph information $h_{avg} = \frac{1}{|V^\ell|}\sum_{v_k \in V^\ell} h_k$. To obtain a permutation invariant control policy π_θ (i.e., independent of the segment order), we first construct a Q-function $Q_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ representing the expected return after taking an action a_k from a state s,

$$Q_\theta(s, a_k) = \begin{cases} FC_1\left(concat(h_{avg}, h_{avg})\right), & \text{if } k = 0, \\ FC_1\left(concat(h_k, h_{avg})\right), & \text{if } k \neq 0 \text{ and } \ell_k \text{ is not a via}, \\ -\infty, & \text{otherwise}, \end{cases} \tag{7}$$

where $FC(\cdot)$ denotes fully-connected layers with $ReLU$ activation. The policy is then recovered by a softmax operator

$$\pi_\theta(a \mid s) = \frac{\exp(Q_\theta(s,a))}{\sum_{a' \in \mathcal{A}} \exp(Q_\theta(s,a'))}. \tag{8}$$

Additionally, we also learn a value function $U_\theta : \mathcal{S} \to \mathbb{R}$,

$$U_\theta(s) = FC_2(h_{avg}), \tag{9}$$

which is used to reduce the variance of policy training.

In every training iteration, we sample actions from the policy to collect D episodes, whose states, actions, and rewards are indexed by superscript d, each with H timesteps. The parameters are updated by minimizing the empirical expectation loss

$$\begin{aligned} L(\theta) = \frac{1}{D}\sum_{d=1}^{D} \min\left(\eta_\theta^{(i,d)} A_\theta^{(i,d)},\ clip(\eta_\theta^{(i,d)}, 1-\epsilon, 1+\epsilon) A_\theta^{(i,d)}\right) \\ + \alpha\left(U_\theta(s^{(i,d)}) - U^{(i,d)}\right)^2 \end{aligned} \tag{10}$$

over a batch of trajectories, where $\eta_\theta^{(i,d)} = \frac{\pi_\theta(a^{(i,d)} \mid s^{(i,d)})}{\pi_{\theta_{old}}(a^{(i,d)} \mid s^{(i,d)})}$ is the probability ratio of the new policy and the old policy, $A_\theta^{(i,d)}$ is the estimated advantage, $clip(\cdot, 1-\epsilon, 1+\epsilon)$ is the clipping function with lower bound $1-\epsilon$ and upper bound $1+\epsilon$, $U_\theta(\cdot)$ returns the value of some state, $U^{(i,d)}$ denotes the discounted return, and ϵ, $\alpha \in \mathbb{R}$ are hyperparameters. $A_\theta^{(i,d)}$ and $U^{(i,d)}$ are defined as follows.

$$\begin{aligned} A_\theta^{(i,d)} &= \sum_{j=0}^{H-i-1} (\gamma\lambda)^j \delta_\theta^{(i+j,d)}, \\ \delta_\theta^{(i,d)} &= r^{(i,d)} + \gamma U_\theta(s^{(i+1,d)}) - U_\theta(s^{(i,d)}), \\ U^{(i,d)} &= \sum_{j=0}^{H-i-1} \gamma^j r^{(i+j,d)}, \end{aligned} \tag{11}$$

where $\lambda \in \mathbb{R}$ is a weighted constant. We minimize the loss in Equation (10) by the Adam optimizer [6]. In our implementation, D, H, ϵ, α, and λ are set to 72, 150, 0.3, 1, and 1, respectively. Note that when long routes are needed for matching, the history cost is not sufficiently accumulated to force the router to detour if the horizon is too short; on the other hand, the router will spend a considerable amount of time if the horizon is too long, thus inefficient.

Algorithm 2 *RLRR*(M)

Input: A circuit with matching groups M, path-matching constraints C with paths P, and an RL policy $\pi : \mathcal{S} \to \Delta\mathcal{A}$.
Output: The optimized routing result Φ.

1: Define a group order τ_M and sort M according to τ_M;
2: $\Phi := \emptyset$;
3: **for each** matching group $m \in M$ **do**
4: Define a net order τ_m and sort m according to τ_m;
5: $\Phi_m := Route(m)$; ▷ generate initial routing for m
6: **for** $iter = 1$ to K_{iter} **do**
7: $s^{(0)} := HeteroGraph(\Phi_m)$;
8: **for** $i = 1$ to K_{step} **do** ▷ actions
9: $a^{(i-1)} := \text{argmax}_a\, \pi(a \mid s^{(i-1)})$;
10: **if** $a^{(i-1)} \neq a_0$ **then**
11: $\Phi_m := Route(Refine(RipUp(\Phi_m, a^{(i-1)})))$;
12: $s^{(i)} := HeteroGraph(\Phi_m)$;
13: **else break**;
14: Reset history costs on the routing grid edges;
15: $\Phi := \Phi \cup \Phi_m$;
16: **return** Φ;

4.3 RL-Guided Rip-Up and Re-Route

In this step, we perform custom routing optimization for the specified critical matching groups. Integrating with well-trained RL policies, We devise an RL-guided rip-up and re-route (RLRR) scheme that executes intelligent action sequences to refine routing patterns.

Algorithm 2 shows the proposed RLRR procedure. Before starting the optimization loop, we define a routing order τ_M for the matching groups M inside the circuit and sort the matching groups according to τ_M, as shown in Line 1. In our implementation, τ_M is the ascending order of the constraint count of each group. With the order, we route and optimize each matching group.

For each matching group $m \in M$, we also define a net order τ_m and generate the initial routing for m regarding τ_m (Lines 4-5). In Lines-6-14, we start the RLRR core process by obtaining the initial state (i.e., the heterogeneous graph representation for the initial solution). Then, with a state as input, the trained RL policy decides which segment to be removed. Finally, we rip up the segment, perform solution refinement (i.e., remove dangling wires and vias), re-route the net, and update the current state until the maximum

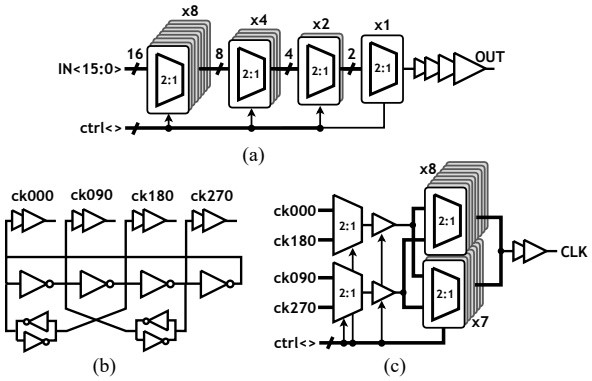

Figure 5: Circuit schematics. (a) 16:1 multiplexing buffer. (b) 4-stage ring oscillator. (c) 6-bit phase interpolator.

step K_{step} is reached or the terminating action is selected, where K_{step} is 150 in our implementation for both training and inference. The entire process repeats until hitting the maximum iterations K_{iter}. Note that the proposed procedure remains efficient when K_{iter} increases as the routing solution converges, and thus a well-trained policy π will terminate the inner loop (Lines 8-13) immediately. After finishing the routing and optimization for all matching groups, we perform standard signal routing on the remaining non-critical signal nets without constraints to obtain the final routing result.

5 EXPERIMENTAL RESULTS

The proposed framework is implemented in C++ and Python with the RLlib library [11]. All experiments are conducted on a Linux workstation with 128 2.7GHz CPUs with 512GB memory and an NVIDIA A100 GPU. Three industrial circuits designed under TSMC 5nm technology are used for evaluation, including a 16:1 multiplexing buffer (BUF), a 4-stage ring oscillator (OSC), and a 6-bit phase interpolator (PI), as shown in Figure 5. Our framework focuses on the custom handling for critical nets, while a digital detailed router completes the routing for non-critical nets (e.g., control/DC signals). Each circuit has a cell placement designed by experienced designers, so a comparison is solely dependent on routing.

Table 4: Statistics of the benchmark circuits.

Benchmark	#Cells	#Nets	#Pins	#Cstrs	#Paths	Area (μm^2)
BUF	42	66	228	6	30	7.40×7.50
OSC	21	16	102	6	14	5.97×5.26
PI	124	122	620	19	56	9.23×11.98

5.1 Routing Metrics Evaluation

We compare our framework with the analog detailed router of [16] in terms of total wirelength, via count, design rule violations, average peak-to-peak path resistance difference, and runtime of critical routing. The average value of maximum path resistance difference per constraint PTP_{res} (defined in Equation (5)) reflects the overall matching. Since [16] considers symmetry constraint variants but lacks the support for generalized path-matching constraints, we annotate additional symmetry constraints on matched nets to approximate the layout needs and optimize PTP_{res}. To demonstrate the importance of the proposed constraints, we generate three layouts for each benchmark with three different settings: 1) [16] without any constraints (Base-1), 2) [16] with symmetry constraints (Base-2), and 3) our framework with generalized path-matching constraints. Table 5 shows the comparisons between the routing results. The detailed router of [16] fails to complete the routing without DRVs in OSC and PI due to local congestion. The layouts generated by Base-1 and Base-2 result in 5.75× and 3.76× higher PTP_{res} than ours, respectively. Note that our framework clears all DRVs and achieves a much more optimized PTP_{res} in the trade-off of slightly increased wirelength and via count of less than 2% difference compared with the others. As for the runtime, the router of [16] shows advantages in sparser designs such as BUF, while the runtime of our framework is dominated by loading trained RL models. Though our framework has increased runtime in dense and congested designs such as OSC and PI, it can eliminate violations and achieve DRC-clean layouts. In contrast, the others terminate the optimization procedure before removing all the violations.

5.2 Post-Layout Performance Analyses

5.2.1 16:1 Digital Multiplexing Buffer (BUF). The 16:1 multiplexing buffer is designed to drive a large capacitive load with the ability to select between 16 input signals (IN<15:0>) using 4-bit control (ctrl<>). Six groups of path-matching constraints were applied across 30 individual paths to minimize per-stage and total input-to-output insertion delay variations across the 16 selectable signal paths. Table 6 summarizes the post-layout simulations comparing a manual layout to auto-routed layouts generated by Base-2 and our framework. On average, both auto-routed layouts produce lower insertion delays (per stage and total) with reduced delay variability compared with the manual layout. While Base-2 achieves lower average insertion delays (-2.3% across per-stage groups and -1.9% total w.r.t ours), our framework reduces delay variability by 26.1% across all per-stage groups and by 19.1% for the total input-to-output delays. The inherent symmetry of the BUF cell placement aids in the symmetry constraints achieving similar performance results to the use of path-matching constraints.

5.2.2 4-Stage Ring Oscillator (OSC). The 4-stage ring oscillator generates 10.3GHz complimentary in-phase (nodes ck000/ck180) and quadrature-phase (nodes ck090/ck270) clocks from a nominal 750mV supply voltage. To minimize duty cycle and quadrature-phase distortion, 6 groups of path-matching constraints are applied to 14 signal paths to reduce delay variations between the core oscillator stages, cross-coupled inverters, and output buffers. Table 7 compares the post-layout performance. Both Base-2 and our layouts produce duty cycles closer to the 50% target, along with lower quadrature (IQ) distortion magnitudes overall. The use of path-matching constraints further reduces IQ distortion values by an additional 30%.

5.2.3 6-Bit Phase Interpolator (PI). The 6-bit phase interpolator adjusts the output clock phase in 1.33ps increments (60 phase steps) by interpolating between 4-phase input clocks (ck000, ck090, ck180, ck270) at 12.5GHz. 19 path-matching constraint groups are applied across 56 individual signal paths to minimize routing asymmetry to the blending stages and thus reduce output phase deviations across input codes. Comparison of post-layout performance are

Table 5: Comparison of total wirelength (WL (μm)), total via count (VIA), design rule violations (DRV), the average value of maximum path resistance difference per constraint (PTP_{res} (Ω)), and runtime (sec).

Benchmark	Base-1 ([16] w/o constraints)					Base-2 ([16] w/ symmetry constraints)					Ours				
	WL	VIA	DRV	PTP_{res}	Runtime†	WL	VIA	DRV	PTP_{res}	Runtime†	WL	VIA	DRV	PTP_{res}	Runtime†
BUF	96.4	271	0	65.7	0.36	96.6	265	0	64.6	0.40	102.0	272	0	**22.3**	3.70
OSC	21.3	88	5	130.4	0.39	21.3	88	5	130.4	0.40	21.0	90	0	23.9	18.38
PI	390.0	830	0	337.5	32.23	380.7	779	1	111.6	123.94	385.8	790	0	38.2	132.27
Norm.	0.99	1.01	-	5.75	0.12	0.98	0.98	-	3.76	0.36	1.00	1.00	-	1.00	1.00

† The runtime of critical routing on nets with constraints. (RL policy training time excluded.)

Table 6: Comparison of the post-layout insertion delay and rise/fall times of BUF.

Stage		Insertion Delay (ps)			Rise / Fall Time (ps)		
		Manual	Base-2	Ours	Manual	Base-2	Ours
1	Avg	11.9	10.1	10.2	10.3 / 10.5	9.2 / 9.4	9.3 / 9.5
	Std	0.33	0.25	0.23	0.27 / 0.29	0.22 / 0.24	0.25 / 0.25
2	Avg	11.9	11.0	11.1	10.2 / 11.4	9.5 / 10.7	9.6 / 10.8
	Std	0.15	0.17	0.15	0.14 / 0.14	0.16 / 0.19	0.12 / 0.12
3	Avg	12.1	11.3	11.6	10.5 / 10.5	9.9 / 9.9	10.0 / 10.1
	Std	0.08	0.08	0.06	0.08 / 0.06	0.05 / 0.06	0.02 / 0.00
4	Avg	11.2	10.4	10.9	9.1 / 9.7	8.4 / 9.0	8.9 / 9.5
	Std	0.01	0.01	0.00	-	-	-
Total	Avg	77.7	72.7	74.1	-	-	-
	Std	0.53	0.42	0.34	-	-	-

Table 7: Comparison of the post-layout oscillation frequency, duty cycle and quadrature phase distortion of OSC.

Layout	Freq. (GHz)	Duty Cycle / Quadrature Phase Distortion (%)				
		ck000	ck090	ck180	ck270	Avg
						Std
Manual	10.27	49.0 / -	49.1 / 0.28	48.9 / 0.00	49.2 / 0.33	49.1 / 0.20
						0.10 / 0.15
Base-2	10.75	50.3 / -	50.3 / -0.17	50.1 / -0.03	50.1 / 0.27	50.2 / 0.16
						0.11 / 0.10
Ours	10.68	50.2 / -	50.3 / 0.15	50.2 / -0.13	50.1 / 0.05	50.2 / 0.11
						0.09 / 0.04

shown in the integral non-linearity (INL) plots in Figure 6, measured as the normalized ratio of the ideal and actual output phase deviations across phase codes (nominally 0). Since PI is sensitive to path-matching delays in and out of the trimmable multiplexers, post-layout performance of auto-generated layouts are compared in Figure 6(a) using path-matching (Ours), symmetry (Base-2), or no constraints (Base-1). The use of symmetry constraints resulted in worse average and maximum INL magnitudes (1.27 and 2.33, respectively) compared with path-matching constraints, while routing without any constraints resulted in significant performance degradation with average and maximum INL magnitudes of 1.91 and 4.60, respectively. Compared with the manual layout with average and maximum INL magnitudes of 1.07 and 2.19, our layout has lower average INL magnitudes (0.84) across the four phase code quadrants with a smaller peak INL value (1.70), as in Figure 6(b).

5.3 Policy Generalization

To evaluate the generalizability of our framework, we compare total training loss and the routing results of PI using policies trained from scratch and fine-tuned with a pre-trained policy over varying time stamps, as shown in Figure 7. We pre-trained a policy using only BUF and OSC, then included PI in the training environment for fine-tuning. The routing cost is calculated as $wl(\Phi) + via(\Phi) + 30ptp(\Phi)$,

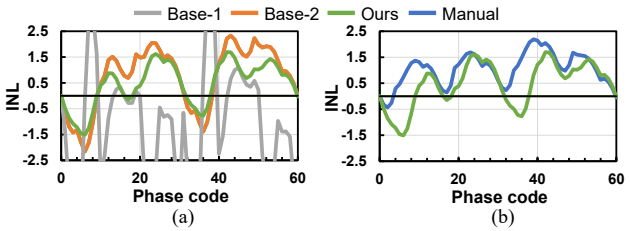

Figure 6: Post-layout integral non-linearity v.s. phase code for PI. (a) Base-1, Base-2, and ours. (b) Manual and ours.

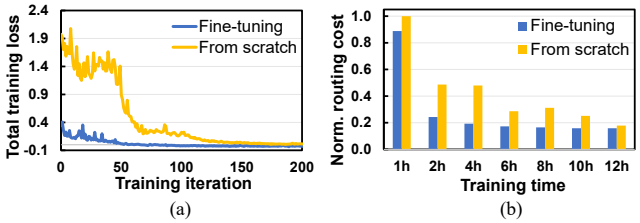

Figure 7: The total training loss and normalized routing cost of trained models from scratch and fine-tuning over varying amounts of time.

where Φ is a routing solution. As can be seen in Figure 7(a), the pre-trained network starts with a better initial solution with much lower training loss and converges after 50 iterations of fine-tuning, while the network trained from scratch struggles at the beginning and still has a higher loss after 200 iterations. As shown in Figure 7(b), the pre-trained network with fine-tuning quickly converges after two hours of training, while training from scratch requires much longer training time (>12 hours).

6 CONCLUSION

This work has presented a novel RL-guided detailed routing framework for advanced custom circuits. A heterogeneous graph representation for detailed routing solution modeling has been proposed. A learning model using graph neural networks for proximal policy optimization-based deep RL learning has been devised. A rip-up and re-routing scheme with RL policy integration has been shown to optimize routing objectives. With minimal adjustments to the reward function and model architecture, the proposed framework can quickly adapt to future design constraints. Experimental results have demonstrated the effectiveness of the proposed custom detailed routing framework in producing sign-off-quality layouts.

ACKNOWLEDGEMENT

This work is supported in part by NVIDIA Corporation, the DARPA IDEA program, and the NSF under Grant No. 1704758.

REFERENCES

[1] Umakanta Choudhury and Alberto Sangiovanni-Vincentelli. 1993. Constraint-Based Channel Routing for Analog and Mixed Analog/Digital Circuits. *IEEE TCAD*, 12, 4, 497–510.

[2] Larry McMurchie and Carl Ebeling. 1995. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proc. FPGA*, 111–117.

[3] Po-Cheng Pan, Hung-Ming Chen, Yi-Kan Cheng, Jill Liu, and Wei-Yi Hu. 2012. Configurable Analog Routing Methodology via Technology and Design Constraint Unification. In *Proc. ICCAD*, 620–626.

[4] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao. 2013. NCTU-GR 2.0: Multithreaded Collision-Aware Global Routing with Bounded-Length Maze Routing. *IEEE TCAD*, 32, 5, 709–722.

[5] Hung-Chih Ou, Hsing-Chih Chang Chien, and Yao-Wen Chang. 2014. Nonuniform Multilevel Analog Routing with Matching Constraints. *IEEE TCAD*, 33, 12, 1942–1954.

[6] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Proc. ICLR*, 1–15.

[7] Juergen Scheible and Jens Lienig. 2015. Automation of Analog IC Layout: Challenges and Solutions. In *Proc. ISPD*, 33–40.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *Proc. NeurIPS*, 1–11.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv, (2017).

[10] Eric Chang, Jaeduk Han, Woorham Bae, Zhongkai Wang, Nathan Narevsky, Borivoje Nikolic, and Elad Alon. 2018. BAG2: A Process-Portable Framework for Generator-Based AMS Circuit Design. In *Proc. CICC*, 1–8.

[11] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. 2018. RLlib: Abstractions for Distributed Reinforcement Learning. In *Proc. ICML*, 3053–3062.

[12] Fan-Keng Sun, Hao Chen, Ching-Yu Chen, Chen-Hao Hsu, and Yao-Wen Chang. 2018. A Multithreaded Initial Detailed Routing Algorithm Considering Global Routing Guides. In *Proc. ICCAD*, 1–7.

[13] Haocheng Li, Gengjie Chen, Bentian Jiang, Jingsong Chen, and Evangeline F. Y. Young. 2019. Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction. In *Proc. ICCAD*, 1–7.

[14] Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. 2019. A Deep Reinforcement Learning Approach for Global Routing. *Journal of Mechanical Design*, 142, 6.

[15] Hao Chen, Mingjie Liu, Xiyuan Tang, Keren Zhu, Nan Sun, and David Z. Pan. 2020. Challenges and Opportunities Toward Fully Automated Analog Layout Design. *Journal of Semiconductors*, 41, 11, 111407.

[16] Hao Chen, Keren Zhu, Mingjie Liu, Xiyuan Tang, Nan Sun, and David Z. Pan. 2020. Toward Silicon-Proven Detailed Routing for Analog and Mixed-Signal Circuits. In *Proc. ICCAD*, 1–8.

[17] Keren Zhu, Mingjie Liu, Hao Chen, Zheng Zhao, and David Z. Pan. 2020. Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network. In *ACM/IEEE MLCAD*, 145–150.

[18] 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, 19–26.

[19] Hao Chen, Keren Zhu, Mingjie Liu, Xiyuan Tang, Nan Sun, and David Z. Pan. 2021. Universal Symmetry Constraint Extraction for Analog and Mixed-Signal Circuits with Graph Neural Networks. In *Proc. DAC*, 1243–1248.

[20] Jaeduk Han, Woorham Bae, Eric Chang, Zhongkai Wang, Borivoje Nikolić, and Elad Alon. 2021. LAYGO: A Template-and-Grid-Based Layout Generation Engine for Advanced CMOS Technologies. *IEEE TCAS I*, 68, 3, 1012–1022.

[21] Jinwei Liu, Gengjie Chen, and Evangeline F.Y. Young. 2021. REST: Constructing Rectilinear Steiner Minimum Tree via Reinforcement Learning. In *Proc. DAC*, 1135–1140.

[22] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, Richard Ho, Roger Carpenter, and Jeff Dean. 2021. A Graph Placement Methodology for Fast Chip Design. *Nature*, 594, 207–212.

[23] Haoxing Ren and Matthew Fojtik. 2021. NVCell: Standard Cell Layout in Advanced Technology Nodes with Reinforcement Learning. In *Proc. DAC*, 1291–1294.

[24] Rajarshi Roy, Jonathan Raiman, Neel Kant, Ilyas Elkin, Robert Kirby, Michael Siu, Stuart Oberman, Saad Godil, and Bryan Catanzaro. 2021. PrefixRL: Optimization of Parallel Prefix Circuits using Deep Reinforcement Learning. In *Proc. DAC*, 853–858.

[25] Hao Chen, Walker J. Turner, Sanquan Song, Keren Zhu, George F. Kokai, Brian Zimmer, C. Thomas Gray, Brucek Khailany, David Z. Pan, and Haoxing Ren. 2022. AutoCRAFT: Layout Automation for Custom Circuits in Advanced FinFET Technologies. In *Proc. ISPD*, 175–183.

[26] Andrew B. Kahng, Lutong Wang, and Bangqi Xu. 2022. TritonRoute-WXL: The Open-Source Router With Integrated DRC Engine. *IEEE TCAD*, 41, 4, 1076–1089.

[27] Yibo Lin, Tong Qu, Zongqing Lu, Yajuan Su, and Yayi Wei. 2022. Asynchronous Reinforcement Learning Framework and Knowledge Transfer for Net-Order Exploration in Detailed Routing. *IEEE TCAD*, 41, 9, 3132–3142.

[28] Keren Zhu, Hao Chen, Mingjie Liu, and David Z. Pan. 2022. Automating Analog Constraint Extraction: From Heuristics to Learning: (Invited Paper). In *Proc. ASP-DAC*, 108–113.