Invited: AI-assisted Routing

Qijing Wang*
CSE Department, CUHK
NT, Hong Kong SAR
qjwang21@cse.cuhk.edu.hk

Liang Xiao*
CSE Department, CUHK
NT, Hong Kong SAR
lxiao23@cse.cuhk.edu.hk

Evangeline F.Y. Young
CSE Department, CUHK
NT, Hong Kong SAR
fyyoung@cse.cuhk.edu.hk

Abstract

Routing is an important but complicated step in physical synthesis. Considering the potential of leveraging AI to seek higher efficiency and better quality in solving routing problems, we study in this work the methodology of AI-assisted routing in a systematic way. Decoupling the functionalities of different routing components will give a high flexibility in determining where and how AI can be used in an effective manner, while maintaining a high degree of interpretability. Two applications along this direction are presented, aiming at tackling the difficulties in routing with AI assistance. These provide examples of how to implement the methodology in practice, while revealing its effectiveness and potential.

CCS Concepts

• **Hardware → Physical design (EDA)**; • **Computing methodologies → Artificial intelligence**.

Keywords

Artificial Intelligence, Assistance, Routing, Design Automation

ACM Reference Format:
Qijing Wang, Liang Xiao, and Evangeline F.Y. Young. 2025. Invited: AI-assisted Routing. In *Proceedings of the 2025 International Symposium on Physical Design (ISPD '25), March 16–19, 2025, Austin, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3698364.3709125

1 Introduction

As one of the most complicated and time-consuming components in physical synthesis, routing has been widely studied and developed for decades. The whole routing process is usually divided into global routing and detailed routing to conquer the high complexity. The former aims at planning the net routing at a coarsened granularity while optimizing objectives like wirelength, congestion and timing, and producing a coarse roadmap, based on which the latter attempts to determine the exact locations and shapes of each components without violating any design rules.

A large part of the difficulty in routing comes from two aspects. On one hand, the topology of nets need to be determined carefully to optimize certain objectives, but even as straight-forward as the rectilinear Steiner minimum tree construction [5] problem is NP-complete. On the other hand, it is non-trivial to organize and route multiple nets under limited resources and constraints. For example, the congestion issues in global routing are caused by multiple nets' excessive consumption of resources at the same location, and the spacing violation in detailed routing happens when the segments of different nets are placed too close to each other [3, 18].

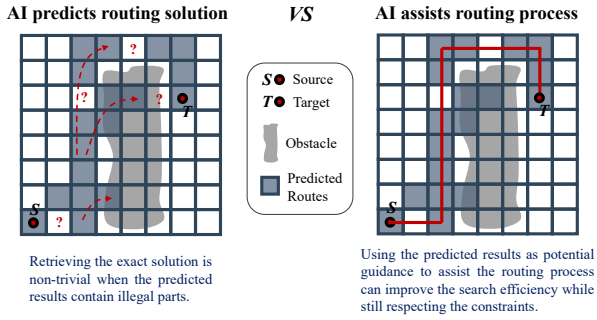


Figure 1: An example of the comparison between two ideas for applying AI in routing.

With the development of artificial intelligence (AI) and the deep learning (DL) techniques in recent years, people have greatly benefited from its unlimited potential for approximation and modeling to solve complex, computationally expensive or high-dimensional problems, e.g., classifying images [13] and playing Go [25]. Researchers also made some efforts to apply DL in solving routing problems. For example, reinforcement learning (RL) is used in [16] to connect two-pin sub-nets during global routing and in [23] to determine net ordering in detailed routing. A recent work [27] is a novel attempt to apply generative AI in routing, but it needs to sample from numerous different generations for each single case to give good result. The generated routing paths may have opens or unconnected components to be fixed.

The question is, where and to what extent AI should be used? Figure 1 shows an example of a comparison between two different ideas for applying AI in routing. In (a), the AI's prediction is directly adopted as the final routing solution. However, this will bring a lot of uncertainties, and retrieving a good solution is non-trivial due to the occurrence of some illegal (open or dangling) parts. However, we can definitely retrieve a lot of useful information from the prediction. As shown in (b), if the prediction is used as a potential guidance to the router, the searching efficiency can be improved while the constraints will still be respected.

In this paper, we will look into this AI-assisted routing problem to study how one can leverage AI to seek higher efficiency and better quality in solving the complext routing problems. Our main contributions are as follows:

- We discuss the methodology of AI-assisted routing and proposed a systematic optimization framework.

*Both authors contributed equally to this research.

ISPD '25, Austin, TX, USA

ACM ISBN 979-8-4007-1293-7/25/03
https://doi.org/10.1145/3698364.3709125

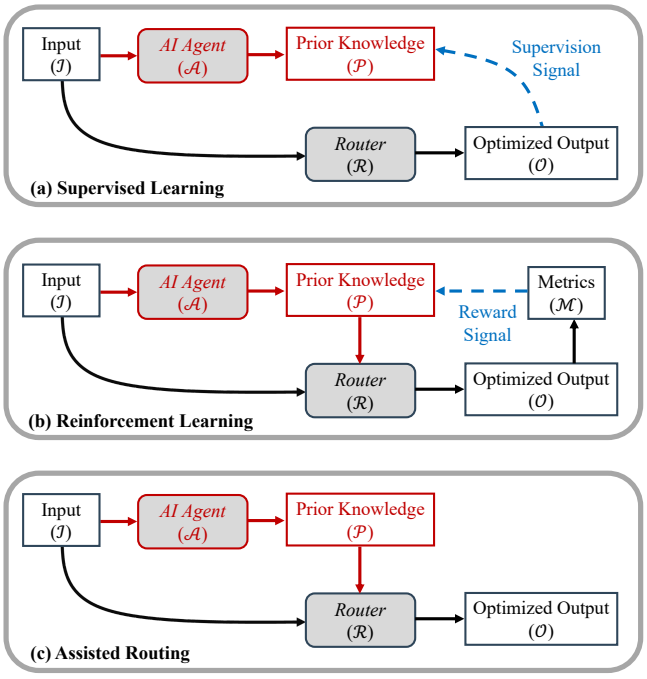


Figure 2: Overview of our AI-assisted routing methodology.

- Against the difficulty in determining the topology of nets, we use AI to assist the shallow light tree construction to achieve advanced quality in shallowness and lightness.
- Against the difficulty in multi-net routing, we use AI to assist the ripup-and-reroute process to obtain faster solutions with fewer violations.

The rest of the paper is organized as follows. Section 2 presents our AI-assisted routing methodology. Section 3 and Section 4 demonstrate how our methodology can be used to assist the shallow light tree construction and multiple net routing in practice. Section 5 further discusses some potential directions, followed by a conclusion in Section 6.

2 AI-assisted routing methodology

In this section, we will discuss the methodology of AI-assisted routing and present a systematic framework to solve the problem in general.

2.1 Components

Figure 2 gives an overview of our AI-assisted routing framework. We define the following terms to describe the nature and functionality of each component in the system:

Input ($\mathcal{I}$) and *Output* ($\mathcal{O}$). They are typically the input and output of the routing task, e.g., the input is a set of pin locations while the output is the corresponding routing paths. Note that some intermediate information can also be used as input, such as the results from previous steps in some iterative process.

Metrics ($\mathcal{M}$). It can either be those used in the original routing task, e.g., #violations, or some closely related auxiliary indicators, e.g., congestion level.

Router ($\mathcal{R}$). It is defined as the tool for solving the original routing task, regardless of its objective and the form of realization. In our framework, we assume this router contains variables, and the values of these variables are determined by some inherent algorithms or heuristics in the original router, e.g., the order of connecting multiple pins, etc.

AI Agent ($\mathcal{A}$). In most cases, the variables defined above are set according to the information at hand for specific operations. In order to leverage the experience and knowledge accumulated previously, this AI agent is incorporated to provide such assistance. This agent can be instantiated into various forms, e.g., neural networks (NN) or a simple lookup table.

Prior Knowledge ($\mathcal{P}$). It is defined as the output of the AI agent, which will be used to modulate the variables in the router according to on-site input, thereby achieving speedup or better quality. It enjoys a great flexibility in how it is adopted. For instance, when trying to determine pin ordering, the prior knowledge could be a set of weights predicted for the pins. This way, they can not only be used directly by value for the decision, but also combined with the weights calculated by the original heuristic as the final criteria.

The decoupling of $\mathcal{A}$, $\mathcal{P}$ and $\mathcal{R}$ is crucial to provide a systematic view to control where and to what extent AI can be used, while maintaining a high degree of interpretability.

2.2 Pipeline

In order to obtain the AI agent that can produce useful prior knowledge, there are two schemes to facilitate the learning process, as shown in Figure 2(a) and (b). In our assumption, $\mathcal{R}$ is non-differentiable, which disables the direct gradient back-propagation through $\mathcal{O} \rightarrow \mathcal{R} \rightarrow \mathcal{P} \rightarrow \mathcal{A}$.

In the first scheme (a), the AI agent will be trained using the supervision signal derived from the optimized output $\mathcal{O}$. In simple terms, it learns what a good prior is from expert demonstration. For example, one can use a prior that reaches the best quality after a computationally expensive process (e.g., after many trials) as the ground truth to supervise the training. The second scheme (b), under the context of RL, we use $\mathcal{M}$ to calculate the reward and estimate the gradient, so as to update the AI agent. In other words, we let the AI agent to discover what a good prior is automatically. The implementation details and effectiveness of these two schemes are demonstrated in the following two applications in this paper.

We only discuss these two classic learning schemes here for clarity, but in practice, one may use various techniques for better training, e.g., un-/semi-supervised learning, pretrain-finetune pipeline, etc. After training, as shown in Figure 2(c), in addition to the standard flow of $\mathcal{I} \rightarrow \mathcal{R} \rightarrow \mathcal{O}$, $\mathcal{A}$ will also take current $\mathcal{I}$ as input to produce the corresponding prior knowledge, $\mathcal{P}$, which will be fed into $\mathcal{R}$ to adjust the internal variables.

Table 1 illustrates how the above components are instantiated in the two applications presented next.

3 AI-assisted shallow light tree construction

3.1 Background

In many routing tools [3, 7, 10, 18, 19, 22], a key step is Steiner tree construction. This step gives a guide for subsequent global routing and detailed routing steps. Several methods have been proposed to

Table 1: Illustration of the instantiation of the components in our AI methodology for the two applications.

Task	*SLT Construction* (Section 3)	*Multi-net Routing* (Section 4)
$\mathcal{I}$	Pin locations of one net	Pin locations of multiple nets
$\mathcal{O}$	Shallow light tree	3D Routing paths
$\mathcal{M}$	Lightness & shallowness	#Violations & routing length
$\mathcal{R}$	Shallow light tree constructor	Rip-up and reroute engine
$\mathcal{A}$	GNN-based model	Transformer-based model
$\mathcal{P}$	Critical Set	Route guides
Learning Scheme	Reinforcement Learning	Supervised Learning

use AI to assist Steiner tree construction [9, 15, 17, 29]. To generate a timing-friendly tree, Yang et al. [29] proposed their reinforcement learning model which balances between total wirelength and the longest source-sink path. Treenet[15] trains a neural network to choose between two heuristics SALT [4] and PD-II [1], and utilizes the potential of the two heuristics to optimize shallowness and average path lengths. In this section, we discuss an AI-assisted shallow light tree construction method based on the framework described in Section 2.

3.2 Preliminaries

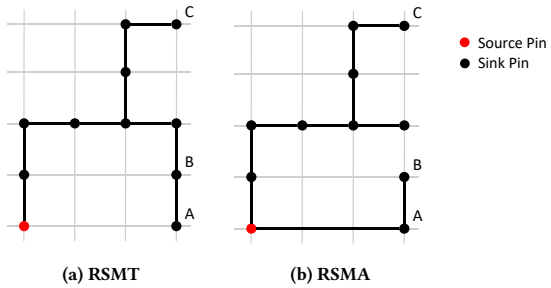
3.2.1 Problem Formulation. For rectilinear Steiner trees, the one with minimum tree weight is called a rectilinear Steiner minimum tree (RSMT), while the lightest one with all paths from root being shortest is a rectilinear Steiner minimum arborescence (RSMA). Shallow light tree (SLT) problem is a problem that tries to minimize source-pin paths and the total wirelength. Our optimization on the Steiner tree is under the problem formulation of [4, 6]. For a Shallow-Light tree, the definition of shallowness α and lightness β are:

$$\begin{aligned} \alpha &= \max\{[d_T(r,v)/d_G(r,v)]|v \in V\backslash\{r\}\}, \\ \beta &= w(T)/w(RSMT(G)), \end{aligned} \tag{1}$$

where $d_T(r,v)$ is the path length from sink v to the source r, and $d_G(r,v)$ is the optimal distance to the source. In an $(\bar{\alpha},\bar{\beta})$-SLT, the distance from any pin to the source does not exceed $\bar{\alpha}$ times the optimal distance, and its total wirelength $w(T)$ does not exceed $\bar{\beta}$ times the optimal wirelength $w(RSMT(G))$ ($\alpha \le \bar{\alpha}, \beta \le \bar{\beta}$). For example, Figure 3a is a shortest wirelength tree, but for point B, its path length to source is 6, while the optimal distance is 4. For point A, its path length over optimal distance is $\frac{7}{3}$. Therefore, it is a $(\frac{7}{3}, 1)$ SLT. The wirelength of Figure 3b is 11, while the shortest total wirelength is 10 in the RSMT. Thus it is a $(1, \frac{11}{10})$ SLT. In this section, we use the wirelength obtained by FLUTE as our reference value for $w(RSMT(G))$.

We use ϵ to represent the shallowness constraint for nets. It means that a tree with shallowness $\alpha > 1+\epsilon$ is an illegal tree.

3.2.2 Related Works. To construct a tree with shallowness no greater than $1+\epsilon$, a straightforward approach is to directly connect points violating this constraint (illegal points) to the source pin. The KRY heuristic [11] implements this idea using DFS to traverse a minimum spanning tree. When finding an illegal point during the DFS, KRY connects it to the source and updates the tree structure accordingly.



(a) RSMT **(b) RSMA**

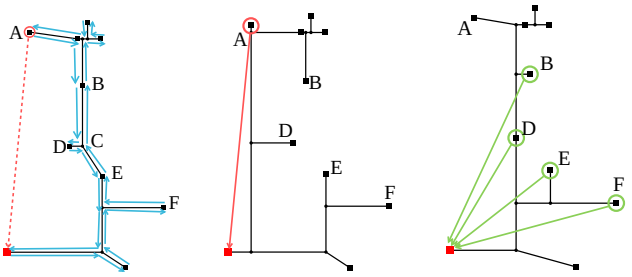
Figure 3: RSMT and RSMA.

SALT [4] also employs the same method to detect illegal points, but it further adapted the method to Steiner trees. Figure 4a illustrates such a DFS process. Starting from the source of an RSMT, SALT traverses the tree along the blue arrows to find illegal points that break the shallowness constraint. A is the only detected illegal point, which will be directly connected to the source. Note that the updated path will influence the points along the DFS traversal. For example, we might disconnect B from its original parent to save wirelength. In that case, B's only child will become its new parent. After the traversal, SALT utilizes a near-optimal RSMA algorithm to connect all the illegal points to the source, rather than using separate paths to directly connect them to the source. Additionally, SALT implements post-processing algorithms to further improve the solution. For instance, in Figure 4b, point D will be connected to the A-source path instead of to point E, thereby reducing the overall wirelength.

As discussed above, we see that there are two main stages for the two algorithms: (1) *Detect*: Both KRY and SALT use a common DFS traversal to detect the points that violate the shallowness constraint. Afterward, the connections between these points and their parents are removed. (2) *Repair*: KRY repairs the tree by connecting all the points to the source directly, while SALT improves the process by using an RSMA to reconnect these points. We summarize this two-stage SLT construction as a *Detect-Repair Framework*.

3.2.3 Limitations of Previous Works. Although the aforementioned Detect-Repair Framework is effective in SLT construction, we find that the detect stage still has a lot of room for improvement. In some cases, it is not necessary to rip up all the illegal points to meet the shallowness constraint. Sometimes ripping up some illegal points can lead to a drastic change of the tree structure, which might be followed by a significant wirelength degradation. Instead, we can achieve the shallowness constraint by adjusting other parts of the tree at a lower cost, even if those parts are already legal.

Figure 4 is an example from our benchmark. In Figure 4b, the only illegal point in the tree is point *A*. In SALT, it is ripped up from the connection to its parent node, and will be reconnected to the source. The result tree (as shown in Figure 4b) will have shallowness $\alpha = 1.3867$, lightness $\beta = 1.2435$. However, we can also rip up points $\{B, D, E, F\}$ instead as shown in Figure 4c. By ripping up these four points and using an RSMA to connect them to the source point, the final tree will have shallowness $\alpha = 1.3316$



(a) DFS on RSMT (b) SALT solution (c) Our solution

Figure 4: An example of suboptimal DFS based detection ($\epsilon = 0.5$). (a) DFS process on an RSMT (shallowness $\alpha = 1.5995$, lightness $\beta = 1$). (b) Solution of SALT ($\alpha = 1.3867$, $\beta = 1.2435$). Illegal points detected during DFS are circled red. (c) Solution given by our method($\alpha = 1.3316$, $\beta = 1.0826$). Points predicted by our model to be reconstructed are circled green.

and lightness $\beta = 1.0826$, which is both shallower and lighter than SALT's result shown in Figure 4b.

In this example, reconstructing point A's connection would significantly alter the original RSMT structure, while reconstructing the connections of $\{B, D, E, F\}$ disturbs the original RSMT structure less but also meets the constraint $\epsilon = 0.5$. Therefore, compared to reconstructing point A's connection, choosing the four points is a better solution. Note that there are multiple solutions that are better than reconstructing A's connection. For example, selecting points D, E, F may yield similar results to B, D, E, F.

To determine a good set of points for reconstruction, merely identifying illegal points is insufficient. We need to comprehensively consider the relationships of multiple points. Machine learning is widely recognized as an effective approach for automatically extracting global features and relationships. Therefore, we employ a learning-based model to predict a better set of points to reconstruct connections. We call this set of points the "**Critical Set**"

3.3 Algorithms and Framework

3.3.1 Overall Flow. The overall flow of our AI-assisted SLT construction is shown in Figure 5. First, the embedder transforms point positions and tree structures into vector embeddings. A transformer-based encoder and a graph neural network are then used to further learn the relationships between nodes. Finally, a Point Selector is employed to identify the Critical Set.

The Critical Set will be directly connected to the source by an RSMA. Some shallowness violations will be fixed in this process but not all. To ensure the legality of our tree, we will traverse the RSMT. If any points still violate the shallowness constraint, we also disconnect them from their parent nodes.

Finally, we repair the tree structure using an RSMA to connect all the disconnected points to the source, followed by post-processing steps adapted from [4]. The lightness and shallowness of the resulting tree are used to compute the reward and gradient for model training.

3.3.2 Model Structures. Our machine learning model to identify a Critical Set to guide the SLT construction mainly consists of an Encoder and a Point Selector. Initially, we select some features of points (coordinates, initial path lengths to the source, distance to the source) as the input, then we use a fully connected layer to generate a feature $F \in \mathbb{R}^{n\times m}$ (n is the number of points. m is the feature dimension, and is set to 64 in our implementation). This is also the input to the Encoder module.

The encoder module consists of three encoder blocks. In each block, we first use multi-head attention to capture the relationship between points, followed by a feed-forward module. We add a batch normalization after the attention and the feed-forward module to increase stability. Additionally, we incorporate residual connections for these two modules. The function of the multi-head attention block is given as follows:

$$\begin{aligned} &\text{SingleHead}(Q, K, M) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_s}}\right)M, \\ &S_j = \text{SingleHead}(F_i W_{Q,j}, F_i W_{K,j}, F_i W_{M,j}), \\ &F_M = \text{Multi-Head}(F_i) = \text{Concat}(S_1, \ldots, S_h)W_m, \end{aligned} \tag{2}$$

where $F_i \in \mathbb{R}^{n\times m}$ is the input feature of the i^{th} encoder block. For single head j, the input feature F_i is first projected into matrices Q, K, M with learnable weight $W_{Q,j}, W_{K,j}, W_{M,j} \in \mathbb{R}^{m\times d_s}$ ($d_s = 16$), then $S_j \in \mathbb{R}^{n\times d_s}$ is calculated with Q, K, M. Multi-head attention concatenates outputs of h single heads and multiply it by learnable weight $W_m \in \mathbb{R}^{hd_s\times m}$. The output of the multi-head attention, $F_M \in \mathbb{R}^{n\times m}$, retains the same shape as the input. After the encoder, we use a GNN block to enhance the feature, it propagates information along the edges of the initial Steiner tree given by FLUTE. Its output is $F_G \in \mathbb{R}^{n\times m}$

We then use a Point Selector to handle the features and predict a Critical Set C. The Point Selector predicts a sequence in descending order of importance $\mathcal{P} = (P_1, P_2, \ldots, P_n)$, which represents a permutation of point indices from 1 to n. We use the points with a higher importance rank than the source pin as our Critical Set, which can be formulated as $C = \{P_s \mid s < h; P_h \text{ is source point}\}$. The function of the Point Selector is to sequentially select points into the importance sequence $\mathcal{P}$.

The input to the Point Selector is the points' features $F_G \in \mathbb{R}^{n\times m}$. We also use an edge aggregation operation before the pointer to enhance features, which concatenates points' features with their parents' features (here the parent relationship is given by the RSMT tree structure). The enhanced feature for point i is $e_i = [x_i, x_{pa}]$, where $x_i, x_{pa} \in \mathbb{R}^m$ are the features of the i^{th} point and its parent node in F_G. By using the RSMT structure in edge aggregation, we merge some information of the original RSMT.

The pointer can choose one point according to the points' embeddings. Its details are provided as follows:

$$\begin{aligned} &p = \text{softmax}(C \times \tanh(l)), \\ &\text{where } l = [l_1, l_2, ..., l_n]^T, \\ &l_i = \begin{cases} -\infty, & \text{if point } i \text{ is selected,} \\ w^T \tanh(W_1 e_i + W_2 q), & \text{otherwise,} \end{cases} \end{aligned} \tag{3}$$

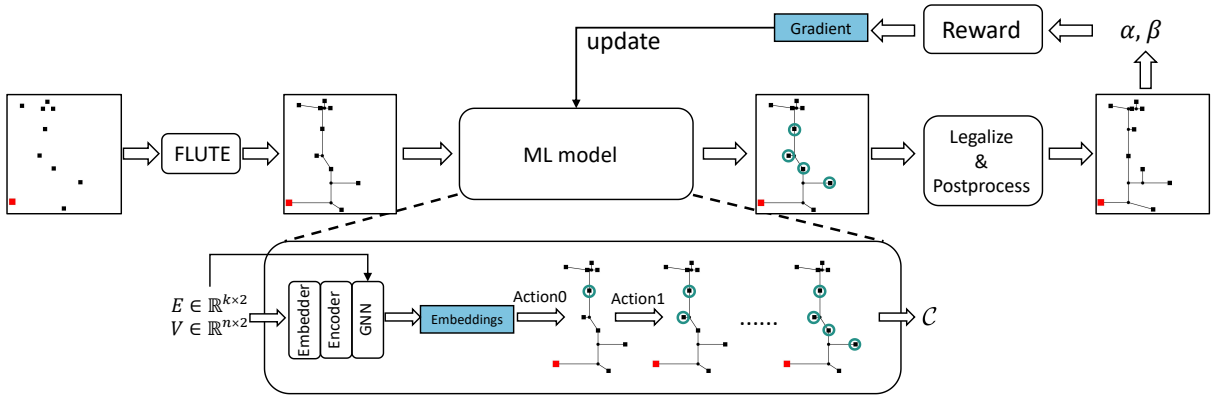


Figure 5: Overall Flow. Given the RSMT generated by FLUTE, our machine learning model encodes the tree structure to node embeddings. An encoder and GNN further learn node relationships. Our Point Selector then sequentially adds points to the critical set C. After legalization and post processing, we finally get a shallow-light tree. The lightness β and shallowness α are then used to compute the reward and the gradient.

where $p \in \mathbb{R}^n$ is the points' probability of being selected, $W_1 \in \mathbb{R}^{d_q \times 2m}$, $W_2 \in \mathbb{R}^{d_q \times 2m}$, $w \in \mathbb{R}^{d_q}$ are all trainable parameters ($d_q = 64$). C is a constant 10. This formulation incorporates information from both the query $q \in \mathbb{R}^{2m}$ and the embedding $e_i \in \mathbb{R}^{2m}$. The score l_i for feature i is computed by applying the weight vector w to the combined representation. Finally, a softmax function is applied to convert these scores into probabilities for point selection. By setting the scores of the points that have been selected to negative infinity, we can filter these points and prevent revisits. The query q is initially a one vector. In subsequent selections, it becomes the node embedding e of the point previously chosen, allowing the network to incorporate the context of prior selections.

In the training phase, we use Monte Carlo sampling to randomly sample from the probability distribution p for exploration and sequentially generate the importance sequence $\mathcal{P}$. In the inference phase, we select the point with the highest probability.

3.4 Experimental Results

We train our model on benchmarks of the ICCAD 2015 Contest [12]. We use the weighted sum of the two objectives (i.e., lightness and shallowness) as the reward to train our model. We used a 4-fold cross-validation strategy. i.e., when testing one quarter, three other quarters of the data are used for training. The improvement compared to SALT (**Imp(%)**) and TreeNet [15] (**Imp*(%)**) is recorded in Table 2. Since the optimal value of shallowness α is 1, we subtract 1 before comparing. For example, 1.05 to 1.04 is a 20% improvement, i.e., $((1.05-1)-(1.04-1))/(1.05-1) = 20\%$.

In the table, "WL deg." represents maximum wirelength degradation. That means we search through different ϵ to obtain the best result that satisfies the constraint that the resulting trees' wirelength doesn't deviate too much from the shortest wirelength. The result of SALT is slightly different from the result in [15], because our searching strategy may be different. The more refined the search, the shallower the results. We search through results with $\epsilon = 0, 0.05, 0.05\times1.5^1, 0.05\times1.5^2, \cdots, 0.05\times1.5^{14}$. The results show that we can outperform Treenet on most degrees and WL degradation thresholds. In total, we can outperform SALT by a range of 2.026% to 11.14%, and outperform Treenet by 0.019% to 6.026%.

Table 2: Shallowness in different wirelength constraints Comparison with

$\|V\|$	Method	WL deg. 0.00	0.05	0.10	0.15	0.20
Small (4-7 pins)	SALT	1.0462	1.0216	1.0078	1.0022	1.0006
	Treenet	1.0461	**1.0210**	**1.0074**	1.0021	1.0005
	Ours	**1.0460**	1.0211	1.0074	**1.0020**	**1.0005**
	Imp.(%)	0.3781	2.0937	4.9801	10.2974	22.2411
	Imp.*(%)	0.1123	-0.4845	-0.0405	5.2143	0.8121
Med. (8-15 pins)	SALT	1.3457	1.1776	1.0838	1.0391	1.0181
	Treenet	1.3435	1.1689	1.0790	1.0370	1.0172
	Ours	**1.3433**	**1.1658**	**1.0756**	**1.0346**	**1.0158**
	Imp.(%)	0.6971	6.6287	9.7666	11.6671	13.0070
	Imp.*(%)	0.0723	1.8424	4.3297	6.5581	8.2224
Large (16-31 pins)	SALT	1.7983	1.3550	1.1568	1.0727	1.0358
	Treenet	**1.7755**	1.3339	1.1481	1.0690	1.0341
	Ours	1.7764	**1.3243**	**1.1419**	**1.0655**	**1.0322**
	Imp.(%)	2.7460	8.6292	9.5302	10.0106	10.1444
	Imp.*(%)	-0.1114	2.8651	4.2151	5.1355	5.6437
Huge (32+ pins)	SALT	2.0126	1.4401	1.2084	1.0987	1.0466
	Treenet	1.9793	1.4152	1.1975	1.0941	1.0444
	Ours	**1.9783**	**1.3994**	**1.1894**	**1.0907**	**1.0428**
	Imp.(%)	3.3802	9.2569	9.1144	8.1004	8.0861
	Imp.*(%)	0.0978	3.8102	4.1081	3.5670	3.5856
All	SALT	1.2532	1.1175	1.0524	1.0236	1.0110
	Treenet	1.2481	1.1114	1.0495	1.0223	1.0104
	Ours	**1.2481**	**1.1088**	**1.0476**	**1.0212**	**1.0098**
	Imp.(%)	2.0261	7.3846	9.1248	10.2683	11.1447
	Imp.*(%)	0.0190	2.3244	3.8210	5.1436	6.0260

4 AI-assisted multi-net routing

4.1 Background

A main challenge in routing is to route multiple nets together, competing for constrained resources, which will be frequently encountered in global or detailed routing. In traditional routing approaches,

nets will be handled sequentially, while various heuristics were designed to assist such a process, e.g., negotiation-based scheme [21], net ordering scheme [2], or collision-aware scheme [20], etc. Handling multiple nets simultaneously is expected to give better solution but there is no effective way to achieve this in a very short running time up till now. Some attempts, including [8], propose to retain the flexibility of each net based on the idea of soft edges and use a network flow algorithm to solve the assignment problem.

This situation inspires us to find out if AI can help to resolve this multiple net routing problem effectively by providing prior knowledge to each individual net, such that a good collaboration planning can be established in advance.

4.2 Preliminaries

The rip-up and reroute (RRR) process is a typical scenario where the multi-net routing is performed. Specifically, the routing segments of those nets causing violations will be ripped-up (i.e., deleted) and reorganized using maze routing. In order to deliver our methodology clearly, in this work we treat RRR as the case study and focus on the routing in a 3D region of a given size $x \times y \times l$. This fixed size routing region approach can be applied to global or detailed routing by partitioning the whole routing region into sub-regions and with pseudo-pins on boundaries to connect paths crossing two sub-regions. As in practice, the routing directions of two adjacent layers will be orthogonal, i.e., alternating between horizontal and vertical. Besides, we only take into account the short violation between wire segments of different nets.

Algorithm 1 (ignoring the parts highlighted about the guide maps) demonstrates the pseudo-code for a standard RRR algorithm. The underlying routing algorithm could be any sophisticated searching based method and in this work, we adopted the simple and widely-used maze routing algorithm [14], namely the Lee algorithm. Since multiple nets are involved, RRR often takes several iterations to meet termination conditions.

Our objective is, given a 3D-routing region of regular size and k multi-pin nets (where the pins can be physical or pseudo), we want to develop an AI agent to generate a route guide to each of these k nets such that the RRR process can be accelerated and the quality can be improved. Those highlighted parts in Algorithm 1 demonstrates how the guides will be embedded into the router. The actual implementation is detailed in Section 4.3.2.

Algorithm 1 Rip-up and Reroute (RRR)

Require: Net set $\mathcal{N} = \{N_1, \ldots, N_k\}$, guide maps $G_1, \ldots, G_k$
1: **function** RRR($\mathcal{N}$, $G_1, G_2, \ldots, G_k$);
2: Reroute set $\mathcal{N}_{\text{re}} \leftarrow \mathcal{N}$;
3: **repeat**
4: **for** each net N_i in $\mathcal{N}_{\text{re}}$ **do**
5: Maze routing for N_i with current cost and G_i;
6: Update cost;
7: $\mathcal{N}_{\text{re}} \leftarrow$ get nets causing violations currently;
8: **until** $\mathcal{N}_{\text{re}} = \emptyset$ or max number of iterations
9: **return** rerouted results

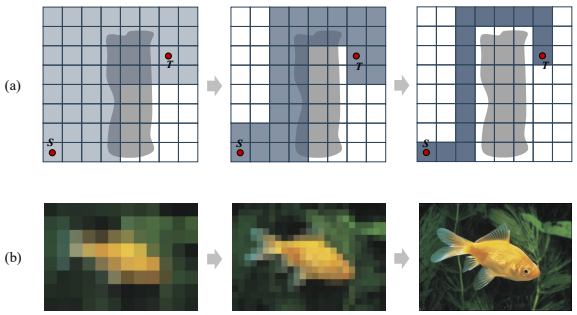


Figure 6: An analogy between the intuition of (a) multi-level maze routing, and (b) image generation based on next-scale prediction.

4.3 Algorithms and Framework

4.3.1 Generative Auto-Regressive Model. A previous work name CUGR [18] proposed to use multi-level maze routing, i.e., from coarse-grained to fine-grained, to compress solution space and reduce runtime, while still maintaining good path searching quality. See Figure 6(a) for an illustration. On the other hand, the work [26] proposed an ingenious method to generate images via the "next-scale prediction" scheme, as depicted in Figure 6(b). Different from other auto-regressive models based on raster-scan schemes that try to perform next-image-token prediction, it is designed to iteratively predict higher-resolution results from former lower-resolution results. Remarkably, this method is able to surpass the state-of-the-art (SOTA) image generation techniques like diffusion models [24] with much higher efficiency. The analogy in Figure 6 reveals a close relationship between the above two ideas, i.e., both perform coarse-to-fine execution processes, which are highly interpretable. Inspired by these works, we developed a generative auto-regressive model that can directly predict 3D routing solutions for multiple nets simultaneously, in a coarse-to-fine manner, according to particular input of pin and capacity information. Figure 7 demonstrates the model architecture.

Specifically, the original routing map will be down-sampled into different scales (i.e., resolutions) to serve as the ground truths during the training. On the other hand, the model performs prediction as follows. At the beginning, the sequences of physical-pins and pseudo-pins will be mapped into a pin embedding through a pin encoder, which is composed of several multi-head attention (MA) blocks (Equation (2)). Then, the first-scale routing map will be predicted by the Transformer. Next, the current prediction will be fed into a negotiation module to facilitate the communication among k different nets and produce the next token containing fused information. This module is also a stack of MA blocks. Afterwards, the second-scale prediction will be produced accordingly. This process will keep running until the last scale T is reached.

Note that the above description only shows the flow for one net, but it will be applied in parallel to all the nets being considered simultaneously.

4.3.2 Guidance Synthesis. Instead of using the model to predict routing paths directly, which likely will contain open or dangling parts, we will use the predicted results to synthesize a guide map G,

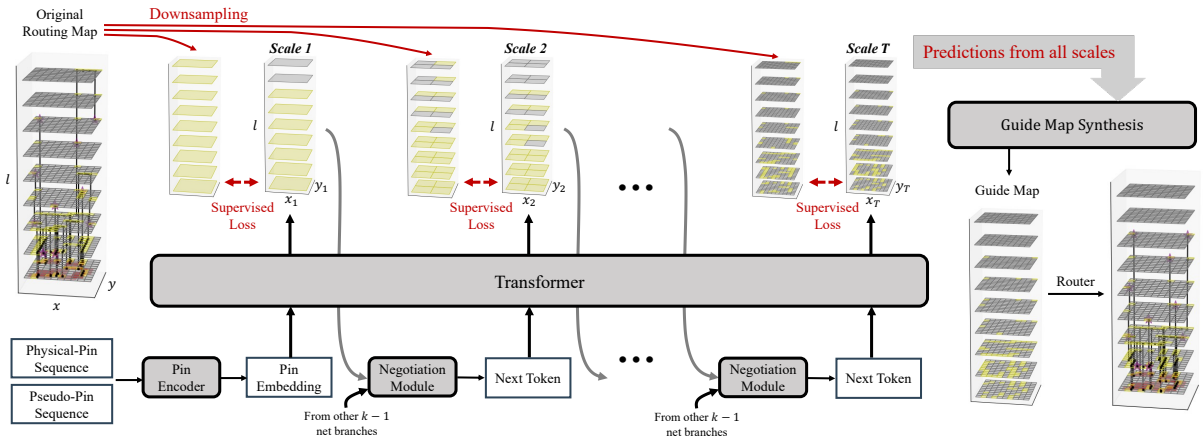


Figure 7: Illustration of the auto-regressive model and guide map synthesis process.

which can serve as a modulation against the costing scheme of an existing routing algorithm. The guide map synthesis is performed as follows. Each of the predicted result from different scale will be up-sampled into the original shape $x \times y \times l$, and an element-wise multiplication of all will then be calculated to produce a combined routing map Z. Then, the values in the guide map are calculated as

$$G(i,j,m) = \begin{cases} w_{\text{guide}}, & \text{if } Z(i,j,m) = 1, \\ 1, & \text{otherwise}, \end{cases} \tag{4}$$

where $i \in [1,x], j \in [1,y], m \in [1,l]$, and $0 \leq w_{\text{guide}} \leq 1$ is set as a small value to reduce the cost at the location that a route is likely to appear. Specifically, suppose the original cost map used in the maze routing (line 5 of Algorithm 1) is C, then the cost for location (i,j,m) is calculated as $C(i,j,m) \times G(i,j,m)$. This scheme enforces a multi-scale guarantee that a location is included in the guide if and only if all intermediate predictions reach consensus at that location, which can further reduce the risk of producing undesirable features.

4.3.3 Training scheme. In order to avoid error accumulation so as to stabilize and accelerate training, we use the teacher-forcing [28] scheme during model training. Specifically, between every two scales, instead of passing the predicted result as the input to the next scale, i.e., the way in inference flow, we will pass the ground-truth. Based on this, cross-entropy loss is used as the criteria to supervise the training.

4.4 Experimental Results

In this work, we set the shape of each region to $x = 15, y = 15, l = 9$. The total number of scales T is set to 4, while the sizes for each scale are set as

$$x_1 = y_1 = 1, \quad x_2 = y_2 = 5, \quad x_3 = y_3 = 10, \quad x_4 = y_4 = 15. \tag{5}$$

In order to have a systematic study, we especially control the sample complexity to obtain a clean data distribution. Specifically, we form five groups L1~L5 with different number of nets need to be routed together, see the first two columns of Table 3 for details. For each net, we set its possible number of physical-pins and pseudo-pins to 1~5 both. For each run, the Algorithm 1 will be executed for at most 100 iterations. We collect 36K samples from each group, which totally gives 180K samples, for training.

After training, we systematically evaluate the effectiveness of our framework under different complexity, where we will use the number of violations (#Vio) and total routing length (TL=wirelength+via count) as the metrics. According to statistics, it takes about 0.03 seconds to generate the guide maps for one sample. Since the generation is only performed once for each region, the runtime overhead of this approach is negligible compared to the routing process. We re-collected 100 samples for each complexity group to serve as the unseen test cases. Figure 8 shows the comparison of the number of violations, as RRR iterates, between the default Algorithm 1 and the version equipped with the guidance generated by our framework, with the guidance weight $w_{\text{guide}} = 0.5$. The vertical axis indicates the number of violations while the horizontal axis represents iteration numbers in log scale. The solid curves denote the average results of 100 runs.

Note that some cases could be extremely hard to be resolved or may not have violation-free solutions. For these cases, the eventual number of violations is not close to zero, especially when the complexity group is L5. We do not remove these cases, as they can mimic some extreme situations. We can clearly see that the version equipped with guidance has much fewer violations right at the beginning even when there is no violation information and cost update, and the number of violations can be successfully reduced to a relatively low level much faster than the default flow. Furthermore, we can observe that the gap between the results at the end of the RRR process becomes more and more significant as the complexity increases, which indicates that such guidance is essential to improve the quality of final results, especially for those complicated cases.

In order to perform quantitative analysis, we capture the average results at the $1st$, $5th$, $10th$, $50th$ and $100th$ iterations and record them into Table 3. We can see that the version with guidance successfully achieves a smaller violation number than the default at every instance. Meanwhile, the guided version enjoys smaller total routing length values. This can be interpreted as some unnecessary

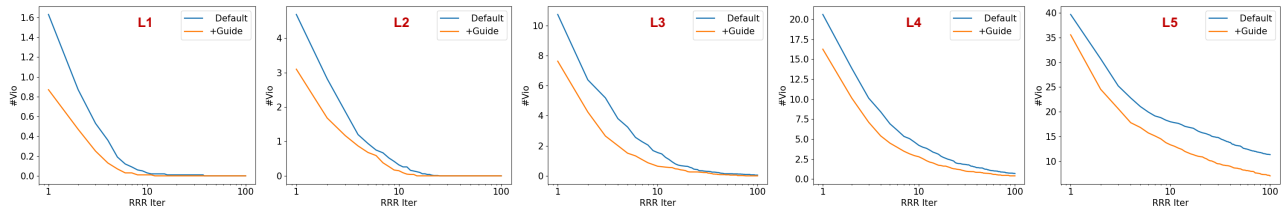


Figure 8: Comparison of the violation dynamics obtained by the default flow and the version equipped with our guidance in the systematic evaluation. For each complexity group, 100 runs are executed and the averaged results are reported.

Table 3: Comparison of the violation and routing length results obtained by the default flow and the guided version.

Complexity	#Nets	Iter 1				Iter 5				Iter 10				Iter 50				Iter 100			
		Default		+Guide		Default		+Guide		Default		+Guide		Default		+Guide		Default		+Guide	
		#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL	#Vio	TL
L1	6~10	1.6	298.0	0.9	303.7	0.2	300.8	0.1	304.5	0.0	301.5	0.0	305.0	0.0	302.0	0.0	305.1	0.0	302.0	0.0	305.1
L2	11~15	4.7	513.5	3.1	519.4	0.9	521.8	0.7	523.4	0.3	525.6	0.2	526.8	0.0	527.2	0.0	527.3	0.0	527.2	0.0	527.3
L3	16~20	10.7	737.9	7.6	742.2	3.3	761.3	1.5	755.6	1.5	771.7	0.6	762.3	0.1	779.3	0.1	763.8	0.1	780.8	0.0	764.4
L4	21~25	20.6	967.0	16.3	966.1	6.9	1023.9	4.5	1003.9	4.2	1037.8	2.8	1011.5	1.2	1040.5	0.7	1021.7	0.7	1041.7	0.4	1025.1
L5	26~30	39.7	1266.8	35.5	1264.8	21.1	1380.8	16.8	1353.4	18.0	1410.6	13.3	1373.2	13.1	1422.7	8.5	1386.7	11.4	1424.7	7.1	1387.6
Avg.		15.5	756.6	**12.7**	759.2	6.5	797.7	**4.7**	**788.2**	4.8	809.4	**3.4**	**795.8**	2.9	814.3	**1.9**	**800.9**	2.4	815.3	**1.5**	**801.9**
Improve. (%)		-	-	**18.00**	-0.34	-	-	**27.11**	**1.20**	-	-	**29.80**	**1.69**	-	-	**36.03**	**1.65**	-	-	**38.46**	**1.64**

detours can be effectively avoided while a more accurate resource allocation can be achieved.

5 Discussion

5.1 Steiner tree construction

In the stage of Steiner tree construction, heuristics like FLUTE [5] can effectively optimize wirelength, but congestion and timing optimization still remains challenging. Recent works [15, 29] and our application in Section 3 demonstrate that ML approaches can potentially generate superior Steiner trees that consider multiple optimization objectives. Moreover, the Steiner tree construction stage, with simple problem formulation, flexible solution space, and relatively low runtime overhead, offers good opportunities for ML integration. These characteristics, in addition to AI's rich capability to encode features like congestion and obstacles into the ML models, create opportunities to enhance the overall routing solution through better initial Steiner routing tree construction.

5.2 Multi-net routing guidance

The benefit of injecting AI generated route guides to solve the multi-net routing problems has been demonstrated in Section 4. One can naturally extend this approach to the scenarios with other types of design rule violations and, as for our future direction, to the real detailed routing environment.

5.3 AI-Assistance Methodology

The proposed AI-assisted routing framework in Section 2 can be applied to various routing tasks in general, as long as the router involved contains variables that are crucial in affecting the results and performance. We believe that leveraging AI to assist routers is far more efficient and practical than replacing the router by an AI counterpart completely. We expect to see more works in this direction from the community.

6 Conclusion

In this paper, we discussed the methodology for AI-assisted routing that enjoys a great flexibility to control where and to what extent AI can be applied to solve the problem, in an efficient manner, while maintaining a high degree of interpretability. The effectiveness of this methodology is revealed in the applications of shallow light tree construction and multiple net routing, which also shed light on its potential to be extended to a wider range of scenarios.

References

[1] Charles J Alpert, Wing-Kai Chow, Kwangsoo Han, Andrew B Kahng, Zhuo Li, Derong Liu, and Sriram Venkatesh. 2018. Prim-Dijkstra revisited: Achieving superior timing-driven routing trees. In *Proceedings of the 2018 International Symposium on Physical Design*. 10–17.

[2] Yen-Jung Chang, Yu-Ting Lee, and Ting-Chi Wang. 2008. NTHU-Route 2.0: A fast and stable global router. In *2008 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 338–343.

[3] Gengjie Chen, Chak-Wa Pui, Haocheng Li, and Evangeline FY Young. 2019. Dr. CU: Detailed routing by sparse grid graph and minimum-area-captured path search. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 9 (2019), 1902–1915.

[4] Gengjie Chen and Evangeline F. Y. Young. 2020. SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 6 (2020), 1217–1230. doi:10.1109/TCAD.2019.2894653

[5] Chris Chu and Yiu-Chung Wong. 2008. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 1 (2008), 70–83. doi:10.1109/TCAD.2007.907068

[6] Michael Elkin and Shay Solomon. 2011. Steiner Shallow-Light Trees are Exponentially Lighter than Spanning Ones. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. 373–382. doi:10.1109/FOCS.2011.18

[7] Jiayuan He, Udit Agarwal, Yihang Yang, Rajit Manohar, and Keshav Pingali. 2022. SPRoute 2.0: A detailed-routability-driven deterministic parallel global router with soft capacity. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. 586–591. doi:10.1109/ASP-DAC52403.2022.9712557

[8] Jiang Hu and Sachin S Sapatnekar. 2000. A timing-constrained algorithm for simultaneous global routing of multiple nets. In *IEEE/ACM International Conference on Computer Aided Design. ICCAD-2000. IEEE/ACM Digest of Technical Papers (Cat. No. 00CH37140)*. IEEE, 99–103.

[9] Andrew Kahng, Robert Nerem, Yusu Wang, and Chien-Yi Yang. 2024. NN-Steiner: A Mixed Neural-Algorithmic Approach for the Rectilinear Steiner Minimum Tree Problem. *Proceedings of the AAAI Conference on Artificial Intelligence* 38 (03 2024),

13022–13030. doi:10.1609/aaai.v38i12.29200

[10] Andrew B. Kahng, Lutong Wang, and Bangqi Xu. 2021. TritonRoute: The Open-Source Detailed Router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 3 (2021), 547–559. doi:10.1109/TCAD.2020.3003234

[11] S. Khuller, B. Raghavachari, and N. Young. 1995. Balancing minimum spanning trees and shortest-path trees. *Algorithmica* 14, 4 (1995), 305–321.

[12] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. 2015. ICCAD-2015 CAD contest in incremental timing-driven placement and benchmark suite. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 921–926. doi:10.1109/ICCAD.2015.7372671

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems* 25 (2012).

[14] Chin Yang Lee. 1961. An algorithm for path connections and its applications. *IRE transactions on electronic computers* 3 (1961), 346–365.

[15] Wei Li, Yuxiao Qu, Gengjie Chen, Yuzhe Ma, and Bei Yu. 2021. TreeNet: Deep Point Cloud Embedding for Routing Tree Construction. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '21)*. Association for Computing Machinery, New York, NY, USA, 164–169. doi:10.1145/3394885.3431566

[16] Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. 2020. A deep reinforcement learning approach for global routing. *Journal of Mechanical Design* 142, 6 (2020), 061701.

[17] Jinwei Liu, Gengjie Chen, and Evangeline F.Y. Young. 2021. REST: Constructing Rectilinear Steiner Minimum Tree via Reinforcement Learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. 1135–1140. doi:10.1109/DAC18074.2021.9586209

[18] Jinwei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline FY Young. 2020. CUGR: Detailed-routability-driven 3D global routing with probabilistic resource model. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[19] Siting Liu, Yuan Pu, Peiyu Liao, Hongzhong Wu, Rui Zhang, Zhitang Chen, Wenlong Lv, Yibo Lin, and Bei Yu. 2023. FastGR: Global Routing on CPU–GPU With Heterogeneous Task Graph Scheduler. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 7 (2023), 2317–2330. doi:10.1109/TCAD.2022.3217668

[20] 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 Transactions on computer-aided design of integrated circuits and systems* 32, 5 (2013), 709–722.

[21] Larry McMurchie and Carl Ebeling. 1995. PathFinder: A negotiation-based performance-driven router for FPGAs. In *Proceedings of the 1995 ACM third international symposium on Field-programmable gate arrays*. 111–117.

[22] Min Pan, Yuehuan Xu, Yanheng Zhang, and Chris C. N. Chu. 2012. FastRoute: An Efficient and High-Quality Global Router. *VLSI Design* 2012 (2012), 608362:1–608362:18. https://api.semanticscholar.org/CorpusID:18414091

[23] Tong Qu, Yibo Lin, Zongqing Lu, Yajuan Su, and Yayi Wei. 2021. Asynchronous reinforcement learning framework for net order exploration in detailed routing. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1815–1820.

[24] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 10684–10695.

[25] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[26] Keyu Tian, Yi Jiang, Zehuan Yuan, Bingyue Peng, and Liwei Wang. 2024. Visual autoregressive modeling: Scalable image generation via next-scale prediction. *arXiv preprint arXiv:2404.02905* (2024).

[27] Qijing Wang, Jinwei Liu, Martin DF Wong, and Evangeline FY Young. 2024. A Multi-agent Generative Model for Collaborative Global Routing Refinement. In *Proceedings of the Great Lakes Symposium on VLSI 2024*. 383–389.

[28] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[29] Liying Yang, Guowei Sun, and Hu Ding. 2023. Towards Timing-Driven Routing: An Efficient Learning Based Geometric Approach. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9. doi:10.1109/ICCAD57390.2023.10323981