AiTPO: KAN-UNet Heterogeneous Network for Timing Prediction and Optimization at Global Routing

HE LIU, School of Electronic and Computer Engineering, Peking University, China
ZHISHENG ZENG, Pengcheng Laboratory, China
SIMIN TAO, Pengcheng Laboratory, China
ZHIPENG HUANG, Pengcheng Laboratory, China
YIFAN LI*, Pengcheng Laboratory, China
BIWEI XIE, Institute of Computing Technology, Chinese Academy of Sciences, China
WEI GAO, School of Electronic and Computer Engineering, Peking University, China
XINGQUAN LI, Pengcheng Laboratory, China

Routing is a critical stage in achieving timing closure in integrated circuit design. Due to the time-consuming flow of detailed routing (DR), the lack of accurate routing information, and the impact of congestion during global routing (GR), rapidly obtaining precise timing information at the global routing stage to guide subsequent timing optimization is a significant challenge. These challenges lead to substantial discrepancies between the estimated timing at GR stage and the actual results after post-DR, resulting in inaccurate evaluations of chip performance. To address this issue, we propose an effective timing prediction and optimization framework, AiTPO. The innovative KAN-UNet heterogeneous timing prediction model effectively combines UNet and KAN networks. By fusing spatial features extracted by UNet with numerical data, the model gains the capability to learn complex relationships across multi-modal data, thereby enhancing robustness and accuracy. Additionally, with the accurate timing evaluation, we introduce two timing optimization strategies during global routing to enhance timing performance. The first strategy involves net ordering based on predicted significant delay nets, prioritizing the routing of more timing-critical nets to reduce detours caused by congestion. The second strategy employs timing estimation to select the most optimal topology from multiple candidates generated by the enhanced A* algorithm, where congestion is considered as a cost factor. Which contributes to optimizing Worst Negative Slack (WNS) and Total Negative Slack (TNS). Experimental results on the real circuits under 28nm process node show that the wire delay prediction accuracy with the proposed KAN-UNet model improves by 34.6% and 25.4% in terms of Mean Absolute Error (MAE) and Max Absolute Error (MaxAE), respectively, compared to GR-based estimations and demonstrate the effectiveness of our timing optimization strategies, which lead to a 2.0% and 4.2% improvement in TNS and WNS, respectively.

CCS Concepts: • **Hardware → Static timing analysis**; **Wire routing**.

*Corresponding author

This work is supported in part by the Major Key Project of PCL (No. PCL2023A03), and the Natural Science Foundation of Fujian Province under Grants (No. 2024J09045).
Authors' Contact Information: He Liu, School of Electronic and Computer Engineering, Peking University, Shenzhen, China, liuh@stu.pku.edu.cn; Zhisheng Zeng, Pengcheng Laboratory, Shenzhen, China, zengzhsh@pcl.ac.cn; Simin Tao, Pengcheng Laboratory, Shenzhen, China, taosm@pcl.ac.cn; Zhipeng Huang, Pengcheng Laboratory, Shenzhen, China, huangzhp@pcl.ac.cn; Yifan Li, Pengcheng Laboratory, Shenzhen, China, liyf03@pcl.ac.cn; Biwei Xie, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, xiebiwei@ict.ac.cn; Wei Gao, School of Electronic and Computer Engineering, Peking University, Shenzhen, China, gaowei262@pku.edu.cn; Xingquan Li, Pengcheng Laboratory, Shenzhen, China, fzulxq@gmail.com.

ACM 1557-7309/2025/XX-ARTXXX
https://doi.org/XXXXXXX.XXXXXXX

Additional Key Words and Phrases: Global routing, timing prediction, timing optimization, UNet, KAN, re-route

ACM Reference Format:
He Liu, Zhisheng Zeng, Simin Tao, Zhipeng Huang, Yifan Li, Biwei Xie, Wei Gao, and Xingquan Li. 2025. AiTPO: KAN-UNet Heterogeneous Network for Timing Prediction and Optimization at Global Routing. *ACM Trans. Des. Autom. Electron. Syst.* XX, X, Article XXX (XX 2025), 28 pages. https://doi.org/XXXXXXX.XXXXXXX

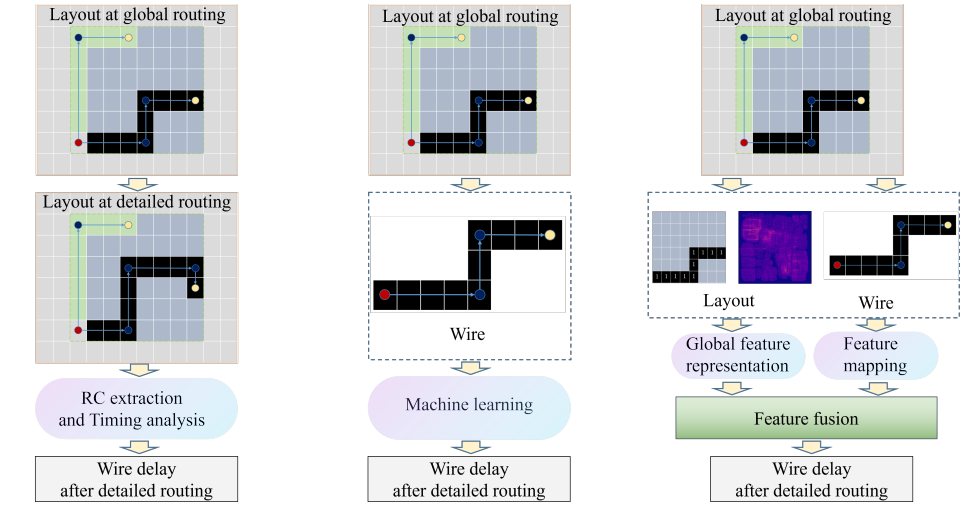
1 Introduction

In the design of modern integrated circuits (ICs), the flow from global routing (GR) to detailed routing (DR) is a critical juncture that significantly influences the final performance of the chip. The GR stage is responsible for generating preliminary routing paths, providing global guidance for the connections between various components on the chip. This stage conveys these path details through guide files, laying the foundation for precise routing in the DR stage. Detailed routing builds upon this foundation, performing more refined routing to ensure compliance with all electrical rules and performance requirements [11]. By effectively transitioning from GR to DR, the design process can achieve significant improvements in optimizing routing resources, enhancing timing performance, and adhering to manufacturing rules, ultimately ensuring the success and efficiency of the design.

The guide files generated during the Global Routing stage can be used for preliminary timing estimation. Although these assessments are somewhat coarse, they provide valuable insights for subsequent timing optimizations. Detailed routing tools utilize these guide files for timing optimization tasks, such as buffer insertion and gate delay adjustments. Additionally, by evaluating the results after the detailed routing stage, designers can return to the GR stage to adjust routing plans and layer assignments, further optimizing timing and power consumption. As is shown in Fig. 1(a), to obtain accurate wire delay information, it is necessary to extract the actual parasitic parameters through the detailed routing stage before performing timing analysis.

However, the detailed routing stage requires careful consideration of numerous physical design rules (such as line width, spacing, resistance, capacitance, etc.) and must precisely determine the final positions and layers for each net [13]. This results in a computational complexity far greater than that of global routing, demanding significant computational resources and time. For large designs, completing DR may take several hours or even days, which is unacceptable for iterative optimization [13]. If timing prediction and optimization are not adequately addressed during the GR stage, more challenging timing violations may emerge during the DR stage. At that point, attempting to correct these issues through buffer insertion, gate-level adjustments, and other techniques can further complicate the routing, potentially introducing new timing issues or routing congestion [22]. Therefore, performing timing prediction and preliminary optimization during the GR stage is crucial for reducing design iterations and improving overall design efficiency.

In the Global Routing stage, the topology of the steiner tree only describes the connectivity and path planning between network nodes, without including detailed physical information such as wire width, thickness, material properties, and the parasitic resistance and capacitance of each metal layer. These physical parameters can only be accurately determined during the detailed routing stage, and they have a crucial impact on signal propagation delay. Approximate wirelength models, such as the Half-Perimeter Wirelength (HPWL) model, are typically used prior to the global routing stage to estimate timing. However, wire analysis based on global routing provides more accurate results than HPWL. These models assume that signal paths are ideal and linear, ignoring complex parasitic effects and routing details. The lack of accurate parasitic information in GR means that timing estimates at this stage are often overly optimistic or pessimistic, which can mislead the subsequent design steps. This idealized approximation can lead to significant discrepancies between the estimated timing and the actual results obtained in the DR stage.



(a) Standard wire delay analysis framework

(b) Existed machine learning based wire delay prediction framework

(c) Proposed multi-modal model framework for wire delay prediction

Fig. 1. Analysis of traditional timing analysis method, existed machine learning-based prediction methods, and our proposed multi-modal model framework.

With the continuous improvement of Electronic Design Automation (EDA) infrastructure [2] and the growing abundance of high-quality datasets [15], the application of machine learning technologies in the EDA domain is deepening and expanding. Therefore, it becomes highly necessary to adopt machine learning methods for timing prediction and optimization during the GR stage [3]. Utilizing machine learning techniques at this stage can significantly enhance the accuracy of timing predictions, while simultaneously guiding timing optimizations, allowing designers to proactively address potential timing violations before they become critical issues in later stages. This dual approach reduces design iterations and accelerates the design process. By integrating timing prediction with optimization, efficient machine learning methods not only ensure that the routing decisions made during the GR stage are robust enough to meet the stringent timing requirements that will be validated in the subsequent DR stage but also significantly improve the overall efficiency and success rate of integrated circuit design.

Previous works. At different stages, timing metric results can be inconsistent with the final post-DR stage due to uncertainty estimation or varying calculation methods, which may lead to a failed sign-off after physical design. To improve timing consistency, it's essential to calibrate and ensure alignment with the final sign-off timing result at different stages. The authors propose Net^2 [24, 25], a graph attention network method for pre-placement net length estimation. Based on net length predictions, they further develop a pre-placement timing estimator, which shows significant improvements in correlation with ground truth compared to commercial tools' pre-placement timing reports. At the placement stage, The work [10] extracts the full timing features using a look-ahead RC network to predict net delay accurately. The authors of [28] use Transformer network and residual model to predict post-routing path delay at pre-routing stage. The work in [8] presents a timing engine-inspired graph neural network (GNN) to predict arrival time and slack at timing endpoints for timing-driven placement. At the global routing stage, the approach used in [4] improves timing consistency between the global route and detailed route using machine

learning techniques and show higher accuracy than estimated timing. The work in [19] proposes a timing prediction and optimization framework, TSteiner to refine the Steiner point after the Steiner tree is constructed for explicit sign-off timing optimization. At the signoff stage, the work [1] uses a machine-learning-based wire timing model to break the RC network loop of the non-tree nets to predict fast and accurate wire delay compared with the sign-off timer. This work [16] develops multiple net delay models with equivalent capacitance as enhanced features to bridge the gap between open-source and commercial timing analysis tool. On the other hand, optimization-aware timing prediction has been proposed for optimization at different design flow stages [9, 23].

Contributions. As is shown in Fig. 1(b) and Fig. 1(c), existing ML methods may fail to precisely identify the causes of the timing gap between GR and DR, thus limiting their capacity to propose more appropriate timing optimization strategies. When using machine learning-based methods for wire delay prediction, it is essential to consider not only the wire-based influence features but also the spatial characteristics of the surrounding layout area. For instance, layout features such as congestion maps provide critical spatial context that directly influences routing behavior. These map-based features help capture the spatial characteristics around the wire, offering a more comprehensive understanding of how the local layout conditions impact wire delay. By integrating both wire-based and map-based features, the predictive model can achieve greater accuracy, reflecting the complex interplay between individual wire characteristics and the broader layout environment. In this paper, we primarily focus on timing delay prediction that considers spatial information features during the global routing stage. Furthermore, timing delay prediction plays a key role in guiding the timing optimization process. We propose the AiTPO framework, which integrates a timing prediction model with two timing optimization strategies. The key contributions are summarized as follows:

(1) We propose the AiTPO framework, a comprehensive approach that effectively integrates an innovative timing prediction model with two timing optimization strategies: net reorder and net re-route. Timing prediction serves as a crucial guide for the subsequent timing optimization process.
(2) We introduce a KAN-UNet timing prediction model that seamlessly integrates the UNet's encoder block into the Kolmogorov-Arnold Networks (KAN), enabling effective multi-scale feature extraction, capturing both large-scale spatial patterns and fine-grained details, while also incorporating these with numerical features for a more comprehensive and accurate multi-modal features representation.
(3) We propose two optimization strategies: net reorder and net re-route. The first strategy reorders nets based on predicted significant delay nets, prioritizing timing-critical routes to mitigate congestion-induced detours. The second strategy employs timing estimation to choose an optimal topology from multiple A* algorithm candidates, explicitly treating congestion as a cost. By prioritizing the routing of nets with larger net delays and re-route critical nets, these strategies contribute to optimizing post-DR Total Negative Slack (TNS) and Worst Negative Slack (WNS).
(4) Experimental results on real circuits demonstrate that the wire delay prediction accuracy with the proposed KAN-UNet model improves by 34.6% and 25.4% in terms of MAE and MaxAE, respectively, compared to estimations made during the GR stage. In terms of timing optimization, after applying the combined net reorder and net re-route strategies, post-DR Total Negative Slack (TNS) and Worst Negative Slack (WNS) are improved by 2.0% and 4.2%, respectively, indicating effective optimization ability and generalizability.
(5) Our framework is easy to extend, and can be easily applied to global routing engines in a simple manner.

The subsequent sections of this paper are organized as follows. Section 2 primarily introduces the global routing problem, A* maze algorithm, and UNet network. Section 3 analyzes the timing evaluation in routing and introduces our proposed framework. Specifically, in Section 4, we elaborate on the timing learning. Section 5 presents our two timing optimization algorithms. In Section 6, we present and analyze the experimental results. Finally, Section 7 summarizes our work.

2 PRELIMINARIES

2.1 Global Routing

Global routing is a crucial step in physical design. During the global routing stage, the entire chip is divided into smaller grids known as Global Routing Cells (GCells), each containing tracks that serve as routing resources. These GCells represent the available routing capacity. Routing resource management is then performed by analyzing the utilization of these resources within each GCell to mitigate congestion issues. Finally, global routing algorithms determine a routing region, composed of groups of GCells, for each routing task. This GCell-level routing region serves as the outcome of the global routing process, which subsequently guides the detailed routing stage.

Traditional frameworks for solving large-scale global routing problems are mainly divided into two types: three-dimensional (3D) routing frameworks and two-dimensional (2D) routing with layer assignment. 3D routing directly obtains the final result on the graph, offering significant flexibility and high-quality solutions. However, it involves a larger search space and higher time complexity. To accelerate computation, some approaches adopt a multi-level framework [17, 26]. The method of 2D routing with layer assignment first compresses the multi-layer routing resources onto a 2D plane for routing. Subsequently, through layer assignment, the 2D results are mapped to different layers, completing the transformation from 2D to 3D. Compared to the 3D framework, this approach offers greater flexibility and lower time complexity [6, 27].

Our global routing flow is based on 2D routing with layer assignment. It first generates the rectilinear Steiner tree topology for each net in the design using the rectilinear Steiner tree lookup table FLUTE [5]. In this work, we employ two optimization strategies: reorder and re-route. Within the reorder strategy, we continue to use the Steiner tree topology generated by FLUTE. In the second optimization strategy, since FLUTE aims for the shortest wire length, to find the optimal delay Steiner tree topology, we decompose the original topology result into multiple two-pin nets. For each net, we employ the enhanced A* search algorithm for routing, integrating perturbations into the heuristic function and considering the congestion influence to generate multiple diverse candidate routing solutions. Among these candidates, the optimal delay Steiner tree is selected as the 2D routing result, which is subsequently transformed into a 3D routing solution through layer assignment.

2.2 A* Maze Routing Algorithm

The A* algorithm is a classic and powerful heuristic search algorithm, widely used in pathfinding and graph search fields. It combines the strengths of Dijkstra's algorithm with heuristic search strategies, considering both the actual cost and the estimated cost of the path. This approach enhances search efficiency while ensuring optimality. In the A* algorithm, each node has an associated cost function $f(n) = g(n) + h(n)$, where $g(n)$ is the actual path cost from the start node to the current node, and $h(n)$ is the heuristic estimate from the current node to the target node (usually the straight-line distance to the goal). A* prioritizes the expansion of nodes with the lowest $f(n)$ value, gradually moving closer to the target node and effectively finding the global shortest path.

In the global routing stage of VLSI design, the A* algorithm has become a cornerstone for efficient and optimal pathfinding. The primary objective of global routing is to establish connectivity between

all logic cells on the chip layout while adhering to stringent design rules and optimizing critical metrics such as wire length, signal delay, and congestion minimization. The A* algorithm achieves this by leveraging a combination of actual path length and heuristic cost estimates, allowing it to not only determine the shortest feasible path but also to avoid regions of high congestion or areas that might introduce excessive delay, power consumption, or routing violations.

A* algorithm can dynamically adapt to the routing landscape, accommodating design complexities such as irregular obstacle distributions and multi-layered routing requirements. In modern integrated circuit designs, where a net may involve multiple sinks, the routing challenge becomes exponentially more intricate. The A* algorithm effectively navigates these complexities by iteratively refining its search to ensure that all endpoints are optimally connected, often utilizing techniques like cost penalties for turns and congestion to guide path selection. This capability is critical in balancing trade-offs between timing, area, and power in advanced technology nodes, ensuring robust and scalable routing solutions in high-density chip designs.

2.3 UNet Network

The UNet[21] model is a fully convolutional neural network architecture specifically designed for image segmentation. Due to its innovative encoder-decoder structure and skip connections, UNet has gained widespread adoption and recognition in the field of image processing, and it has also been successfully applied to various other image segmentation tasks.

The UNet architecture is characterized by its symmetric "U" shape, comprising two main components: the encoder (downsampling path) and the decoder (upsampling path), which are symmetrically arranged. The encoder consists of a series of convolutional layers followed by max-pooling layers, primarily aimed at extracting multi-level features from the input image. Each convolutional block typically includes two successive 3×3 convolutions (with Rectified Linear Unit activation functions [7]) and a 2×2 max-pooling operation. Through progressive downsampling, the encoder captures higher-level semantic information while reducing the spatial resolution of the feature maps. The decoder is designed to progressively restore the spatial resolution of the feature maps and reconstruct them into a segmentation map that matches the size of the input image. At each upsampling step, transposed convolutions are used to increase the resolution, and skip connections are employed to concatenate the feature maps from corresponding encoder layers. This integration of low-level spatial details with high-level semantic information enables the generation of highly accurate segmentation results. Skip connections play a critical role in the UNet architecture by directly concatenating high-resolution feature maps from the encoder with the corresponding feature maps in the decoder. This design effectively mitigates the loss of spatial information caused by multiple downsampling operations, enhancing the model's localization ability and leading to superior performance in semantic segmentation tasks.

In the feature selection stage of the time prediction model described in this paper, we transformed the congestion map as the image-based feature. By leveraging the powerful feature extraction capabilities of the UNet's Encoder, we performed comprehensive feature extraction. This approach allowed us to capture complex spatial-temporal patterns and multi-scale information that might not be easily discerned through traditional methods, thereby enhancing the overall predictive performance of the model.

3 Analysis and Framework

3.1 Timing Analysis in Routing

3.1.1 Wirelength and Timing Delay. In recent years, global routing methods have predominantly focused on minimizing total wirelength as a primary optimization objective. While this approach

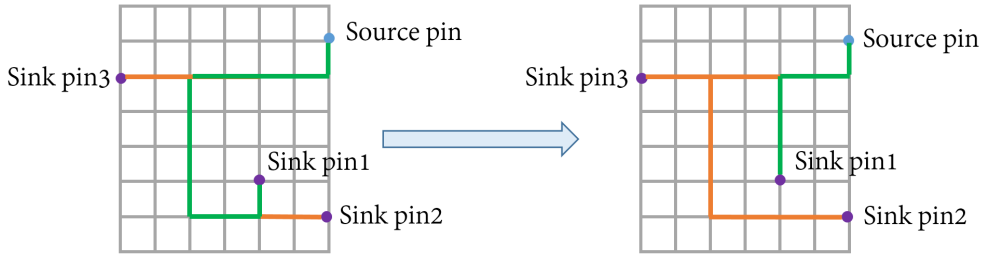


Fig. 2. The impact of optimized topology routing on wirelength and delay. From left to right, the topology changes, resulting in an overall increase in total wirelength. However, the wirelength for the critical path from the source pin to sink pin 1 (green line) is reduced.

effectively reduces overall resource usage, it often overlooks the critical impact of timing delays on the performance of integrated circuits. This paper argues that optimizing for wirelength alone is insufficient, particularly in scenarios where timing closure is crucial.

We present a comparative analysis that highlights the trade-offs between total wirelength minimization and critical path length optimization. As is shown in Fig. 2, as the topology changes from left to right, the overall wire length increases, but the wirelength for the critical path from the source pin to sink pin 1 (green line) is shortened, potentially reducing the delay. Our findings indicate that while optimizing for timing delay may lead to a modest increase in total wirelength, it significantly reduces the length of critical paths, thereby enhancing the overall timing performance of the design.

The motivation behind this shift in focus stems from the observation that traditional GR methods, which prioritize wirelength reduction, do not adequately address timing delays. By incorporating timing delay as a key optimization objective, we aim to achieve better timing closure, even if it comes at the cost of increased wirelength. The results of our study underscore the importance of balancing wirelength and timing delay in global routing to meet the stringent performance requirements of modern integrated circuits.

3.1.2 Global Routing and Detailed Routing. During the flow of physical design, the discrepancy between post-GR and post-DR metrics plays a crucial role in achieving optimal timing performance. Traditionally, global routing optimization has been guided by GR-specific metrics, which may not always align with the actual timing results at post-DR. This misalignment can lead to suboptimal timing closure, as the global routing metrics do not fully capture the complexities introduced during the detailed routing stage.

To illustrate this point, we present a comparative analysis of timing performance between GR and DR stages. As the Fig. 3 shown, the wire delay estimated during global routing often deviates significantly from the actual delay observed after detailed routing. Our findings indicate that optimizations relying solely on GR metrics frequently lead to suboptimal timing outcomes compared to those informed by DR metrics. This discrepancy highlights the need for predictive models that can better anticipate DR timing results during the earlier GR stage.

The motivation behind this approach is to bridge the gap between GR and DR, ensuring that the optimization efforts during GR are more closely aligned with the actual timing requirements of the final design. By integrating DR prediction into the GR process, we can achieve more accurate timing evaluations, leading to enhanced performance and more reliable timing closure in the final design.

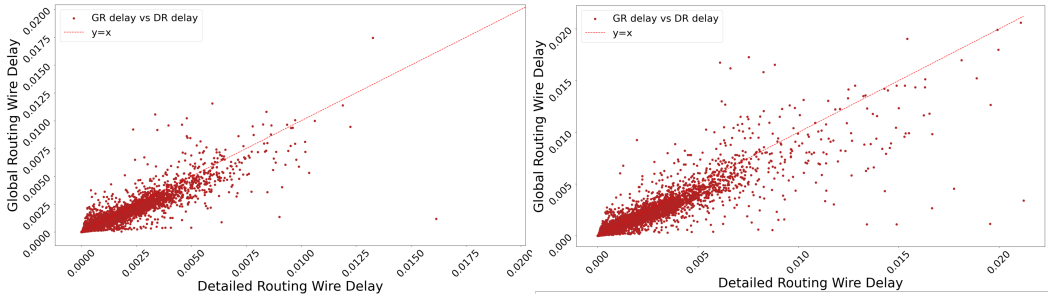


Fig. 3. Distinction between post-GR and post-DR wire delays of ysyx_1 (left) and ysyx_5 (right). The red dashed line represents the ideal case where the wire delay at post-GR would equal the wire delay in post-DR (i.e., the line y=x).

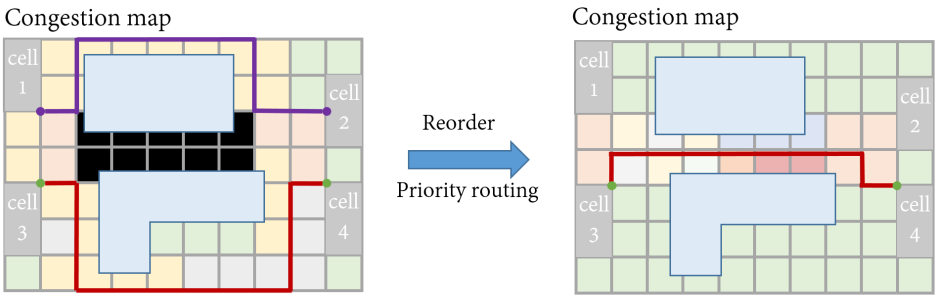


Fig. 4. Comparison of routing order impact. By reordering the routing sequence based on delay magnitude and prioritizing the routing of these critical nets, wire lengths can be reduced, leading to decreased delays.

3.1.3 Net Order in Detail Routing. In integrated circuit design, nets with larger net delays are usually located on critical paths, and these critical paths determine the overall clock frequency of the circuit. As is depicted in Fig. 4, critical nets with longer wirelength and higher delays after DR may have been detoured to avoid congested areas, leading to increased wire length. Fig. 5 illustrates the impact of different routing orders. When the routing sequence in the left diagram is a-b-c, congestion occurs between net *b* and *c*. In contrast, following the routing sequence c-b-a in the right diagram prevents the congestion. By prioritizing the routing of nets with larger delays, it is feasible to reduce their wire lengths to a certain degree, thereby optimizing the delay.

By prioritizing the routing of these nets, timing optimization can be addressed early, thereby reducing the delays on critical paths and meeting timing requirements. According to Fig. 6, we can observe that after applying the reorder optimization strategy, both post-DR TNS and WNS have been improved to some extent. Prioritizing the handling of nets with larger delays can effectively decrease the timing margin of critical paths, reduce the risk of timing violations, and enhance the reliability of the circuit. If nets with larger delays are prioritized for routing and optimization, timing bottlenecks in the design process can be quickly resolved, reducing the number of iterations required.

3.2 Our Timing Optimization Framework

By conducting timing optimization in the GR stage, potential timing problems can be identified and resolved early, reducing the number of design iterations needed later and thus accelerating the

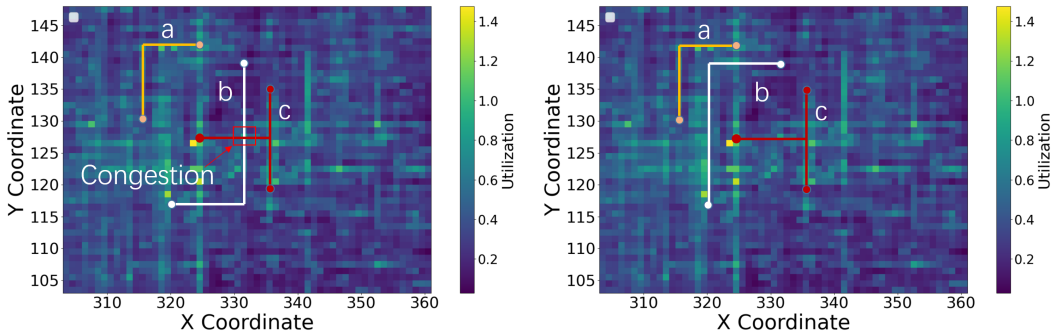


Fig. 5. The impact of different routing orders: When the routing order in the left figure is a-b-c, congestion occurs between b and c. However, if the routing order is changed to c-b-a as shown in the right figure, congestion does not occur.

Fig. 6. A comparison of the impact of routing order on WNS and TNS after detailed routing (DR).

overall design process. In this paper, we propose a novel timing-driven optimization framework for the global routing stage, as illustrated in Fig. 7. The entire framework consists of two parts. The first part is the timing prediction component, which predicts the post-DR timing during the global routing stage and uses the prediction results to guide the optimization strategies in the next stage. The second part is the timing optimization component, which primarily includes two optimization strategies: net reorder and net re-route.

3.2.1 Timing Prediction. The primary function of the timing prediction module is to predict the net delay after detailed routing during the global routing stage. Based on the prediction results, net reorder and net re-route operations are guided. In this work, net delay refers to wire delay (from source pin to sink pin). When a net has multiple pins, the two optimization approaches are discussed as follows: (1) In the reorder optimization approach, since the sorting is based on the net delay, net delay refers to the delay of the wire with the largest delay within the net; (2) In the re-route optimization approach, critical wires in the critical nets are re-routed, and in this case, net delay refers to the delay of the critical wires that are part of the critical path. Before model training, effective feature extraction is required. After completing the GR operation, an open-source static timing analysis engine is used to obtain features related to net delay, such as slew, resistance, capacitance, as well as the corresponding layout features. These features are primarily modeled as map-based features and numerical features. Based on the feature data type, we propose the

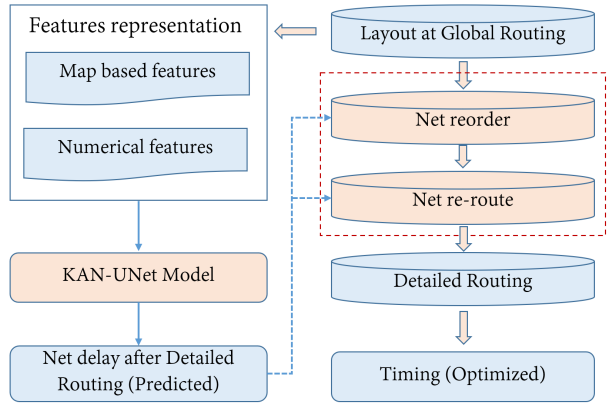


Fig. 7. Overall framework of the proposed AiTPO for timing prediction and optimization. **Stage I**: The process of constructing the timing prediction network to guide the timing optimization procedure. **Stage II**: The timing optimization procedure, consists of the net reorder and the net re-route strategies.

KAN-UNet network, where the model's output is the predicted net delay. The prediction results are then analyzed to guide the optimization process.

3.2.2 Timing Optimization: Net Reorder. In the physical design, nets with larger delays are typically on critical paths, which set the circuit's overall clock frequency. Prioritizing the routing of these nets allows early timing optimization, reducing critical path delays and ensuring timing requirements are met. Using the timing prediction model constructed in the previous stage, the net delay after detailed routing is predicted. The predicted net delay results are then sorted from largest to smallest, and priorities are assigned accordingly. Nets with higher priority are routed first, while those with lower priority are routed later. The re-route is performed based on the established priority weights.

3.2.3 Timing Optimization: Net Re-route. In the design process, timing bottlenecks in critical paths are often the main reason for multiple iterations. By re-route, these bottlenecks can be effectively addressed, accelerating timing convergence, reducing the number of design iterations, and shortening the design cycle. During the global routing stage, FLUTE uses a pre-computed lookup table to quickly generate an approximately optimal Steiner tree based on a given set of terminal points. This approximation serves to guide the subsequent detailed routing steps. The objective of the Steiner tree is to minimize the total wire length while meeting routing constraints. However, the routing resulting from this might not necessarily be optimal for timing after detailed routing. By using predictions from a timing prediction model, adjustments can be made to the routing of critical nets during the GR stage, prioritizing timing over wire length. Detailed routing is then performed based on the adjusted connections, aiming to improve timing performance.

4 Timing Prediction Framework

By predicting the timing of detailed routing during the global routing stage, potential timing bottlenecks can be identified and optimized in advance. This allows design engineers to perform targeted optimizations based on these predictions, thereby reducing the number of iterations required in the detail routing stage. Traditionally, timing analysis after detailed routing necessitates the completion of the detailed routing process followed by parasitic parameter extraction, which is

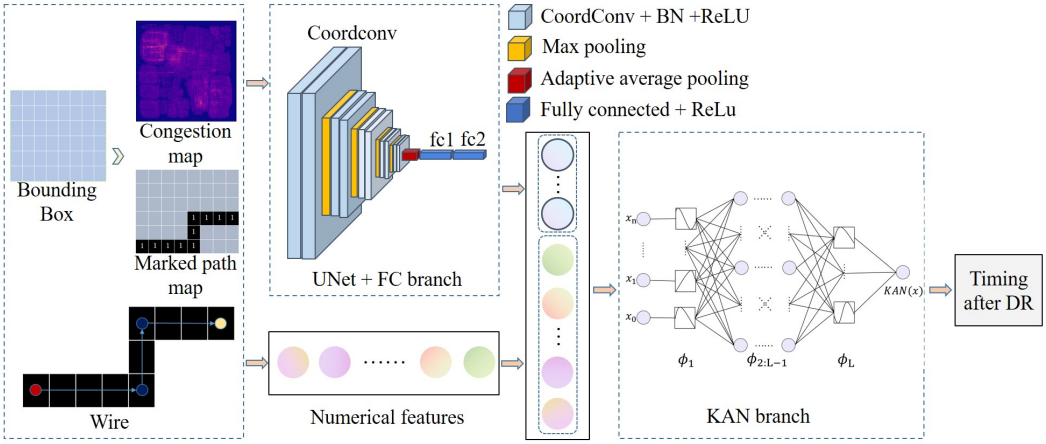


Fig. 8. Architecture of KAN-UNet for timing prediction at global routing stage. Our KAN-UNet consists of two branches: the encoder of UNet branch and the KAN branch.

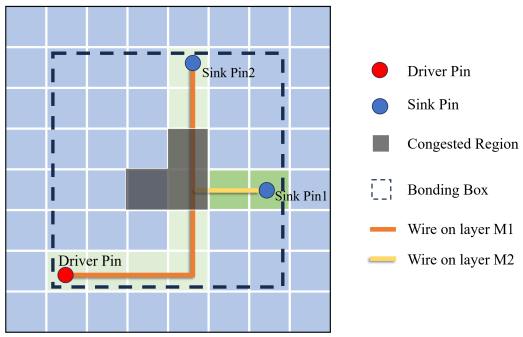


Fig. 9. A route guide for a three-pin net with a Driver and two sinks. It provides an initial routing plan for each net and also includes allocation information for each net across different metal layers.

time-consuming and impractical for scenarios requiring rapid optimization. To rapidly ascertain the delay results of critical nets, in this work, we propose a machine learning-based framework to quickly predict the wire delay after detail routing during the global routing stage, as illustrated in Fig. 8.

4.1 Feature Selection

In machine learning-based prediction tasks, feature selection plays a crucial role in determining the accuracy of the model. In this work, we have selected the following features to construct a timing prediction model for predicting net delay after detailed routing. The physical features of the nets we selected are extracted from the guide files generated during global routing. As shown in the Fig. 9, the guide files provide detailed information for each net, including pins, wires, layers, and vias. Additionally, the timing analysis engine computes critical parameters such as driver slew, estimated parasitics capacitances and resistances. Congestion maps are calculated based on the supply and demand of each routing layer, leveraging this comprehensive dataset. Based on the different structures of the feature data, we categorize them into map-based features and

numerical features. Based on the different extraction objects, the features can also be categorized into wire-based features and layout-based features.

Slew of the driver pin. The slew at the driver pin is one of the key factors in determining net delay. A slower slew (i.e., slower transition speed) increases signal propagation time, since the signal takes longer to reach the threshold level, ultimately resulting in increased delay. After constructing the Steiner tree from the driver to the load pin during the GR stage, the slew features are analyzed using a static timing analysis engine.

Parasitic resistance and capacitance. The total resistance and total capacitance values from the driver pin to a specific sink load in the net. The resistance value from the driver to the load is calculated by multiplying the resistance per unit length of each segment on each layer by its length, and then summing these values for all segments.

Net length. The total wire length of all segments from the driver pin to the load pin. The length of each segment from the driver to the load can be obtained from the guide files generated during the global routing stage. The strong correlation between source-sink length and these net delay characteristics underscores its importance in the predictive modeling of post-DR performance. Incorporating source-sink length into the analysis is essential for achieving precise and reliable post-DR timing predictions.

Number of sinks. In the field of VLSI design for nets with a large number of sinks, the Half-Perimeter Wire Length (HPWL) often significantly underestimates the actual wire length. This underestimation becomes particularly pronounced in multi-sink nets, where the post-GR and post-DR wire lengths exhibit substantial discrepancies. These discrepancies arise due to the increased likelihood of routing detours and the tendency for net lengths to exceed the simple bounding box estimation provided by the HPWL metric. Consequently, this deviation underscores the limitations of HPWL in accurately predicting the true wire length in complex multi-sink scenarios. Therefore, after using net length as a feature, we also include the number of sinks in the feature set.

Estimated wire delay at post-GR. The wire length is estimated by calculating the number of GCells that the signal path crosses and the size of each GCell, including its width and height. If the signal path is horizontal or vertical, the wire length equals the number of GCells crossed multiplied by the width or height of a GCell. Once the wire length is determined, the resistance and capacitance of the signal path can be estimated. Assuming the resistance and capacitance per unit length are known, the total resistance equals the wire length multiplied by the resistance per unit length, and the total capacitance equals the wire length multiplied by the capacitance per unit length. Finally, the Elmore delay model is used to estimate the signal propagation delay. The Elmore delay model is a commonly used method for delay estimation, which estimates signal propagation delay by multiplying the resistance and capacitance. In this case, the delay equals the product of the resistance and capacitance, which is also the square of the wire length multiplied by the resistance and capacitance per unit length. Although the estimated delay is not highly accurate, it can provide some guidance for predicting the net delay after detailed routing.

Congestion map. The congestion map for each net is constructed by mapping the congestion levels within the bins that lie inside the net's bounding box. This map serves as a detailed representation of the routing congestion within the net's specific area, capturing the intensity and distribution of congestion across the region. The congestion map provides valuable insights into where routing resources are heavily utilized, indicating potential areas of congestion that could lead to routing conflicts or increased delays. The congestion in a particular bin is essentially the ratio of track demand to track supply within that bin and can be expressed as: $C^v = \frac{\text{demand}(v)}{\text{supply}(v)}$. Since the bounding boxes have different sizes, the dimensions of the input maps vary as well. Before feeding the maps into the network, we process them by padding them to a uniform size. In this

work, we use a size of 360×360, which can be adjusted, but it should be larger than the size of the largest bounding box that the wire occupies. Based on the importance of the congestion map, we also selected the mean, variance, maximum, and minimum values from the congestion map as the numerical features for the wire.

Marked wire path map. For each net's bounding box, all bins that the wire path passes through from the driver pin to the load pin are marked as 1, while all other positions are marked as 0, forming the marked wire path map. Marking the trajectory of the wire within the bounding box to form a marked wire path map captures the spatial information of the wire layout, which has a significant impact on wire delay. By incorporating features such as the congestion map, it effectively integrates factors like congestion and neighboring interference into the feature set, further enhancing the model's performance and utility.

4.2 Neural Network Construction

Based on the selected map-based features and numerical features, we developed a KAN-UNet network, which integrates UNet and KAN architectures, for feature extraction and timing prediction. This hybrid network is designed to effectively capture both spatial and numerical characteristics of the data, enabling more accurate and robust timing predictions. As the detailed structure of the KAN-UNet network illustrated in Fig. 8, the entire KAN-UNet timing prediction framework consists of two structures: the UNet branch and the KAN branch.

Modified UNet branch. Unlike the traditional UNet's encoder part, we use CoordConv [18] to replace the Conv layers in that section. CoordConv is a variant of the convolutional layer designed to enhance the spatial awareness of convolutional neural networks. Traditional convolutional layers lack explicit awareness of the positional information in the input feature maps. CoordConv addresses this by adding coordinate information (x, y coordinates) to the input feature maps, and optionally, radius information (r) can be added. Allowing the convolutional layers to perceive the spatial position of the input data.

For the input map-based features $X_{map} \in \mathbb{R}^{B\times C\times H\times W}$, where B is the batch size, C is the number of channels, and H and W are the height and width of the feature map, respectively. For each input feature map, position information (x, y coordinates) is added to form the enhanced input, The normalized coordinate channels x_{channel} and y_{channel} are defined as:

$$x_{\text{channel}}(h, w) = \frac{2w}{W-1} - 1, \quad h = 0, \ldots, H-1; \quad w = 0, \ldots, W-1. \tag{1}$$

$$y_{\text{channel}}(h, w) = \frac{2h}{H-1} - 1, \quad h = 0, \ldots, H-1; \quad w = 0, \ldots, W-1. \tag{2}$$

Here, $x_{\text{channel}}, y_{\text{channel}} \in \mathbb{R}^{1\times H\times W}$.

If the radius channel is not used, the feature tensor after concatenation is given by:

$$X' = X_{\text{map}} \oplus x_{\text{channel}} \oplus y_{\text{channel}} \in \mathbb{R}^{B\times (C+2)\times H\times W}, \tag{3}$$

Where $\oplus$ represents the channel concatenation operation. And a standard 2D convolution is then applied to X', resulting in:

$$Y = \text{Conv}(X'). \tag{4}$$

The convolutional block used for feature extraction consists of a module composed of two CoordConv layers, batch normalization, and the ReLU activation function. The output Y of the input feature map after passing through the convolutional layers can be expressed as:

$$Y = \text{ReLU}(\text{BatchNorm}(\text{CoordConv}(\text{ReLU}(\text{BatchNorm}(\text{CoordConv}(X_{map})))))). \tag{5}$$

After reducing the resolution through a max-pooling operation and passing the output feature Y' through several convolution layers, global average pooling is then applied, followed by two fully connected layers for the output. These can be expressed as:

$$Y' = f_{\text{down}}(Y), \tag{6}$$

$$Z = \text{GlobalAvgPool}(Y'), \tag{7}$$

$$Z = \text{ReLU}(\text{Linear}(Z)), \tag{8}$$

$$X_{\text{map\_encoded}} = \text{Linear}(Z). \tag{9}$$

Here, f_{down} denotes the combined operation of multiple stages of downsampling and convolution.

KAN branch. Kolmogorov-Arnold Networks (KAN) [20] is a neural network architecture based on the Kolmogorov-Arnold super approximation theorem. The theorem states that any continuous multivariable function $f(x_1, x_2, \ldots, x_n)$ can be represented as a combination of a set of one-dimensional functions. Specifically, $f(x_1, x_2, \ldots, x_n)$ can be expressed in the following form:

$$f(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{2n+1} \phi_i \left(\sum_{j=1}^{n} \lambda_{ij} \cdot (x_j + \psi_j) \right). \tag{10}$$

where ϕ_i and ψ_j are certain continuous univariate functions, and λ_{ij} are appropriate constants.

For numerical feature input, $X_{\text{num}} \in \mathbb{R}^{B \times D}$, where D is the dimension of the numerical features. The encoded map features and numerical features are concatenated to form the fused feature:

$$X_{\text{fused}} = \text{Concat}(X_{\text{map\_encoded}}, X_{\text{num}}), \tag{11}$$

Then, the fused features are processed through the KAN module for timing prediction:

$$T = \text{KAN}(X_{\text{fused}}). \tag{12}$$

Where B is the batch size, and $T \in \mathbb{R}^{B \times 1}$ is the final output of the model, representing the predicted net delay.

5 Timing Optimization

The timing prediction of post-detailed routing net delays plays a crucial role in guiding timing optimization during the Global Routing stage. In this work, we introduce two optimization methods: net reorder and net re-route.

5.1 Timing Aware Net Reorder

As shown in Fig. 10 and Algorithm 1, the net reorder optimization strategy's process and algorithm implementation are detailed. First, it is necessary to identify the target nets for reordering. Based on the analysis in Section 3.1.2 for Fig. 3, as for most nets, the wire delay predicted during GR stage is relatively close to the actual wire delay observed at post-DR. However, there is a noticeable spread around the line, particularly as the delays increase. This observation suggests that, for certain nets, the wire delay estimated during global routing deviates significantly from the actual delay observed after detailed routing. When prioritizing the routing of nets with larger delays, there is substantial space for optimization, making it unnecessary to predict and reorder every net.

In this work, an initial round of detailed routing is performed, followed by timing analysis. The nets are then sorted in descending order based on their delay values, approximately the top 5,000 nets with the largest delays are selected. For these nets, delay predictions are made using a

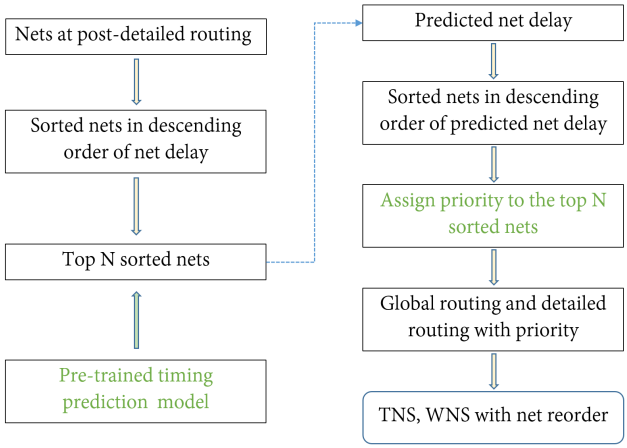


Fig. 10. Flow of the net reorder optimization strategy guided by timing prediction.

pre-trained timing prediction model, based on features from the global routing stage. This step corresponds to Step 1 in Algorithm 1. The nets are then reordered according to the predicted delay results, with priority assigned to those with the highest predicted delay values. This step corresponds to Step 2 in Algorithm 1. Nets with higher predicted delays are identified as critical and are given higher priority during the detailed routing process. By sorting nets according to their predicted delays, the DR stage can prioritize the routing of critical nets earlier in the process when routing resources are more available. This reduces the likelihood of these nets encountering congestion and suboptimal routing paths, which could exacerbate delay issues. Subsequently, routing is performed according to the assigned priorities, including both global routing and detailed routing, to obtain optimized timing results. This method effectively mitigates potential timing violations by addressing the most critical nets upfront, leading to a more optimized overall timing performance.

If further iterative optimization is needed, the previous step only needs to proceed to the global routing stage. Based on the results from global routing, the pre-trained model is used again to predict the selected wire delay, reorder the nets, and assign new priorities for the next iteration of optimization. After several iterations, the process proceeds to the detailed routing stage, where timing analysis is performed again to update the reordered nets for the next round of optimization. In this work. we do not include iterative optimization but provide extensibility for future iterative optimization efforts.

5.2 Net Candidate Generation

As shown in Fig. 11, the main process for generating candidate net path topologies is introduced. First, it is essential to identify the critical nets that require re-routing. After performing an initial round of detailed routing, a static timing analysis engine is used to report the critical timing paths. The nets within these critical paths are then sorted by delay, and nets with delays exceeding a certain threshold are selected as critical paths. This is because, although some nets may be part of a critical path, their delay may be too small to yield significant optimization benefits.

Once the critical nets are identified (including the net names, source pin names, and sink pin names), the enhanced A* algorithm described in Algorithm 2 is applied. This enhanced A* algorithm introduces perturbations when calculating the Manhattan distance, incorporates the

Algorithm 1 Net Reorder Algorithm for Timing Optimization

Input: List of nets N, Predicted wire delays D for each net
Output: Sorted list of nets with assigned routing priorities
Initialize an empty list P to store nets with their priorities
// **Step 1: sorting based on predicted delay**
Sort nets N based on the corresponding predicted delays D in descending order
for each net n_i in sorted list N **do**
    Calculate the priority p_i based on the predicted delay D_i
    Add (n_i, p_i) to list P
end for
// **Step 2: priority assignment**
Assign routing priority to each net in P, starting with the highest priority for the net with the largest predicted delay
for each (n_i, p_i) in list P **do**
    Route net n_i with priority p_i
    Ensure net n_i is routed with consideration of optimal wire length
end for
return List of routed nets with assigned priorities

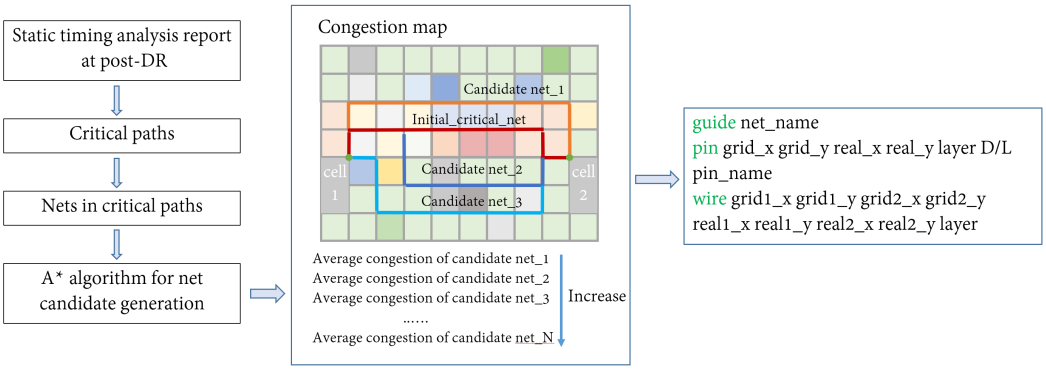
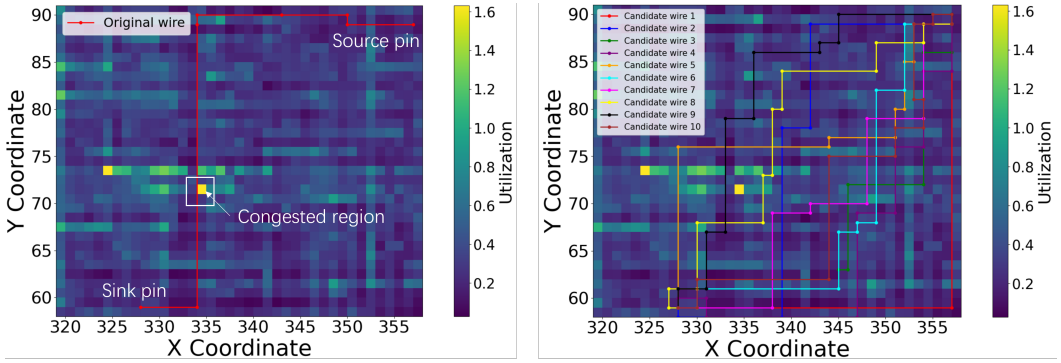


Fig. 11. The procedure for generating multiple candidate paths for the critical path and producing the corresponding guide file for each candidate.

utilization_map into the tentative_g_score, and accounts for turn penalties to discourage excessive turns in the routing path. The A* algorithm is thereby biased toward selecting paths that pass through regions with lower congestion. This helps to reduce congestion globally and optimize the overall design. The enhanced A* algorithm generates multiple topological paths, which are then sorted in descending order based on their average congestion value. To provide a more intuitive description and visual representation of the re-routing algorithm, Fig. 12 is presented. In the left image, we show the Steiner tree routing information for the critical wire in the original critical net, where the wire may pass through congested areas. In the right image, we use the improved A* algorithm with random perturbation and Steiner point penalties to generate ten candidate wires. By incorporating the utilization map into the g_score, the generated candidate wires avoid passing through congested areas. Additionally, the candidate wires are sorted in ascending order based on the average utilization of the Gcells they pass through. To expedite the optimization process, this



(a) Original wire in net (b) Candidate wires

Fig. 12. Comparison of routing paths between original wires and generated candidate wires in the congestion map. Original wires may traverse congested areas, whereas the generated candidate wires avoid these regions and are sorted based on their average congestion levels.

work selects only the 1st, 3rd, 5th, 7th, and 10th paths as candidate nets, and a guide is generated for each candidate net.

The Algorithm 2 implements the enhanced A* search algorithm for pathfinding from a driver pin to a target pin (sink pin), specifically designed for multi-net reconstruction tasks. The algorithm takes as input a utilization_map, which records congestion levels in the routing area; a start point (driver pin); a goal point (sink pin); a perturbation parameter that introduces randomness into the heuristic evaluation to explore multiple potential paths; and a turn_penalty coefficient that discourages frequent direction changes to promote straighter routes.

The algorithm begins by initializing the priority queue open_list with the start node and setting up the came_from, g_score, and f_score maps to track the path and costs. In each iteration of the main loop (lines 7-40), the node with the lowest f_score is selected as the current node. If the current node is the goal, the algorithm reconstructs and returns the optimal path by backtracking through the came_from map.

For each neighboring node in the four cardinal directions (up, right, down, left), the algorithm performs the following steps (lines 17-39):

Boundary and utilization check: It first checks if the neighbor is within the grid bounds and not in a highly congested area by referencing the utilization_map. Neighbors that are out of bounds or have a utilization value greater than 1 are skipped.

Turn detection and penalty: The algorithm determines whether moving to the neighbor would result in a turn by using the IsStraightLine function. If a turn is detected, a turn_penalty is added to the tentative g_score, encouraging straighter paths.

Score calculation and update: It calculates the tentative g_score for the neighbor. If this score is better than any previously recorded score for the neighbor, the came_from, g_score, and f_score maps are updated, and the neighbor is added to the open_list for further exploration.

If the open_list is exhausted without reaching the goal, the algorithm returns None, indicating that no viable path exists. However, in the actual physical design routing process, it is essential to find a viable path. The Heuristic function (lines 42-45) calculates the Manhattan distance between two nodes and adds a random perturbation to introduce variability in path selection. The IsStraightLine function determines whether moving from the previous node to the current node and then to the neighbor maintains a straight line, thereby detecting turns.

Algorithm 2 Enhanced A* Algorithm for Net Candidate Generation

Input: $start$ (Driver pin), $goal$ (Sink pin), $utilization\_map$, $perturbation$ (Perturbation parameter), $turn\_penalty$ (Turn penalty coefficient)
Output: $path$ (Multiple paths from the start point to the goal point)
Initialize priority queue $open\_list$ and push $(0, start)$ into $open\_list$
$came\_from \leftarrow \{\}$
$g\_score[start] \leftarrow 0$
$f\_score[start] \leftarrow Heuristic(start, goal, perturbation)$
while $open\_list$ not empty **do**
    Pop $(f, current)$ with lowest f_score from $open\_list$
    if $current = goal$ **then**
        $path \leftarrow []$
        while $current \in came\_from$ **do**
            Prepend $current$ to $path$
            $current \leftarrow came\_from[current]$
        end while
        Return $path$
    end if
    for all $direction \in \{(0,1), (1,0), (0,-1), (-1,0)\}$ **do**
        $neighbor \leftarrow current + direction$
        // Check if neighbor is out of bounds or over-utilized
        if $neighbor$ **then** out of bounds **or** $utilization\_map[neighbor] > 1$
            continue
        end if
        if $current \in came\_from$ **then**
            $prev \leftarrow came\_from[current]$
            $is\_turn \leftarrow \neg IsStraightLine(prev, current, neighbor)$
        else
            $is\_turn \leftarrow False$
        end if
        // Apply turn penalty if a turn is made
        $turn\_cost \leftarrow turn\_penalty$ **if** is_turn **else** 0
        // Calculate tentative g-score with turn penalty
        $tentative\_g \leftarrow g\_score[current] + 1 + turn\_cost$
        if $neighbor \notin g\_score$ or $tentative\_g < g\_score[neighbor]$ **then**
            $came\_from[neighbor] \leftarrow current$
            $g\_score[neighbor] \leftarrow tentative\_g$
            $f\_score[neighbor] \leftarrow tentative\_g + Heuristic(neighbor, goal, perturbation)$
            Push $(f\_score[neighbor], neighbor)$ into $open\_list$
        end if
    end for
end while
Return None
function HEURISTIC($a, b, perturbation$)
    // Heuristic with random perturbation to diversify paths
    return Manhattan_Distance(a, b) + perturbation × Random_Value()
end function

This enhanced A* algorithm effectively integrates random perturbation to diversify pathfinding, turn penalties to promote straighter routes, and a utilization map to avoid congested areas. These modifications make the algorithm particularly suitable for net routing.

5.3 Net Evaluation and Selection

As shown in Fig. 13, after generating the guide files for each candidate path topology of each critical net, these guides are employed in the global routing process, followed by timing analysis at the global routing stage. This process yields the candidate net features necessary for the timing prediction model. Subsequently, the pre-trained wire delay prediction model is utilized to predict the wire delays for these candidate nets. The predicted delays are compared with each other as well as with the delay estimates derived from the Steiner tree topologies generated via FLUTE. The guide topology corresponding to the candidate net with the lowest predicted delay is selected for further routing. After determining the optimal guide topology for each selected critical net, non-critical nets continue to utilize their initial Steiner tree topologies. Global routing and subsequent detailed routing are then executed based on these guide files, followed by an evaluation of the post-DR wire delay, specifically assessing any improvements in WNS and TNS.

The Algorithm 3 takes as input a list of critical nets N_{critical}, a utilization map U, and the start and goal points $(\text{start}_i, \text{goal}_i)$ for each net. Additionally, it requires a perturbation parameter P that is used in the A* algorithm to introduce randomness in the heuristic function. The output of the algorithm is the set of re-routed paths for the critical nets, where each path is optimized to minimize the delay based on the predicted timing results. In Step 1, the algorithm uses the A* algorithm to generate multiple candidate paths for each critical net. The paths are generated by considering the congestion information provided in the utilization map U and the start and goal points for each net. The perturbation parameter P is used to explore various potential routing paths. In Step 2, for each candidate path generated in the previous step, the algorithm calculates the predicted delay using a timing prediction model. This step involves evaluating the timing performance of each candidate path, which is essential for determining the best routing option. In Step 3, the algorithm identifies the optimal path for each critical net by leveraging the predicted delays calculated in the previous step. The path with the minimum predicted delay is selected as the best candidate. In Step 4, the algorithm compares the predicted delay of the selected optimal path with the delay of the initial routing path generated by the FLUTE algorithm. If the delay of the optimal path is less than that of the initial path, the algorithm selects the optimal path for re-route. Otherwise, it retains the initial path. Finally, the algorithm re-routes each critical net using the selected path, resulting in a set of re-routed paths that are optimized for timing performance. The re-routed paths are returned as the output of the algorithm. This process systematically re-routes critical nets using the enhanced A* algorithm for path generation, followed by timing prediction and comparison, ensuring that the best possible routing paths are chosen for optimal timing performance.

6 Experimental Results

6.1 Experimental Setting

The timing prediction algorithm and model were implemented using Python and Pytorch on a Linux machine with two NVIDIA A100 GPUs, 4 Intel Xeon Platinum 8380 CPUs at 2.3 GHz, and 1T RAM. We used the "ysyx" design set[1] with RTL to generate DEF files on the 28nm process using a commercial tool. A portion of the features and labels required for timing prediction were extracted using the open-source AiEDA framework [12], while the timing optimization framework is implemented based on the iEDA-iRT [14, 29] and open-source static timing analysis tool. The

[1] ysyx is a RISC-V processor chip talent plan.

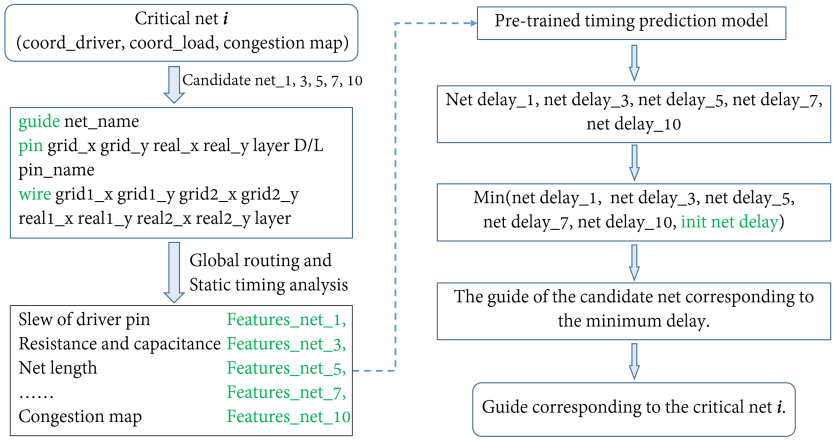


Fig. 13. The procedure for selecting the timing-optimal guide file for each critical path from the candidate paths with the timing prediction model.

Algorithm 3 Re-Route Strategy Using A* Algorithm for Path Generation

Input: List of critical nets $N_{critical}$, Utilization map U, Start (Source pin) and goal (Sink pin) points for each net $(start_i, goal_i)$, Perturbation parameter P
Output: Re-routed paths for critical nets
for net n_i in $N_{critical}$ **do**
    // Step 1: path generation with A* algorithm
    Generate multiple candidate paths $Paths_i$ using the A* algorithm with inputs U, $start_i$, $goal_i$, and P
    // Step 2: timing prediction for each path
    for path p_j in $Paths_i$ **do**
        Calculate predicted delay D_{ij} for path p_j using the timing prediction model
    end for
    // Step 3: selection of the optimal path
    Select the path p_{best} with the minimum predicted delay D_{ij} from $Paths_i$
    // Step 4: comparison with initial routing
    Compare the predicted delay D_{best} of p_{best} with the delay $D_{initial}$ of the initial FLUTE-generated routing path
    if $D_{best} < D_{initial}$ **then**
        Choose p_{best} as the final routing path for net n_i
    else
        Retain the initial routing path for net n_i
    end if
    Re-route net n_i using the selected path
end for
return Re-routed paths for critical nets

descriptive statistics of these benchmark circuits are summarized in Table 1. The upper six circuits are used for training along with the lower three are utilized for testing. The training was performed on a single NVIDIA A100 GPU with the datasets. We employed the AdamW optimizer with an

initial learning rate of 0.00056, adjusted via the CosineAnnealingLR scheduler, and used Mean Squared Error (MSE) as the loss function. Training the KAN-UNet model took approximately 8 hours in total, with each epoch taking about 12 minutes to complete. We trained for 40 epochs using a batch size of 32. However, this is a one-time cost, if trained in parallel on multiple GPUs, the total training time can be further reduced. In timing prediction tasks, Mean Absolute Error (MAE) and Max Absolute Error (MaxAE) are used to measure accuracy, while improvements in post-DR TNS and WNS are used to evaluate the effectiveness of optimization strategies during the timing optimization process.

Table 1. Training and Testing Data Statistics.

	Area (um×um)	Cells	Net num	Utilization	
ysyx_0	370.02 × 370	88860	86071	0.63	Train
ysyx_1	371 × 371	95581	92630	0.65	
ysyx_2	394.94 × 395	97207	98064	0.62	
ysyx_3	434 × 434	145160	141757	0.70	
ysyx_4	683.06 × 683	283285	275312	0.50	
ysyx_5	644 × 644	244476	239401	0.50	
ysyx_6	763.98 × 764	350494	341425	0.50	Test
ysyx_7	791.98 × 792	379679	371241	0.50	
ysyx_8	971.04 × 971	469977	457585	0.40	

Table 2. Experimental results assessing the accuracy of timing prediction for net reorder. The ysyx_0 to ysyx_5 were used for training, while the rest constituted the test set. “MAE” denotes the mean absolute error metric and the“MaxAE” means the max absolute error metric. “SCC” refers to the Spearman correlation coefficient.

Circuit	Post-GR			Machine learning [4]			KAN-UNet (Ours)			Runtime (s)		
	MAE(ns)	MaxAE(ns)	SCC	MAE(ns)	MaxAE(ns)	SCC	MAE(ns)	MaxAE(ns)	SCC	DR+STA	KAN-UNet	Speed-up
ysyx_0	0.0008	0.0180	0.86	0.0006	0.0058	0.97	0.0006	0.0084	0.91	1920	5.69	337×
ysyx_1	0.0007	0.0110	0.88	0.0006	0.0004	0.97	0.0005	0.0066	0.92	2040	3.82	534×
ysyx_2	0.0008	0.0215	0.85	0.0006	0.0071	0.96	0.0006	0.0116	0.89	2100	3.74	561×
ysyx_3	0.0011	0.0107	0.79	0.0004	0.0066	0.99	0.0008	0.0093	0.86	2580	3.81	377×
ysyx_4	0.0019	0.0247	0.68	0.0007	0.0093	0.99	0.0013	0.0138	0.81	3569	3.50	1020×
ysyx_5	0.0019	0.0218	0.70	0.0007	0.0068	0.99	0.0012	0.0135	0.81	3120	3.76	830×
Average	1.000	1.000	0.79	0.500	0.334	0.98	0.692	0.585	0.87	2555	4.05	631×
ysyx_6	0.0023	0.0222	0.60	0.0017	0.0232	0.75	0.0015	0.0154	0.77	4140	3.54	1169×
ysyx_7	0.0024	0.0231	0.55	0.0017	0.0291	0.72	0.0016	0.0180	0.74	4320	3.45	1252×
ysyx_8	0.0030	0.0242	0.45	0.0023	0.0215	0.64	0.0020	0.0185	0.67	4860	3.36	1446×
Average	1.000	1.000	0.53	0.731	1.060	0.70	0.654	0.746	0.73	4440	3.45	1287×

6.2 Accuracy Evaluation on Timing Prediction

To evaluate the performance of the timing prediction model, we compare our results with recent work, specifically, the “Machine Learning [4]” study. It is important to note that the proposed machine learning-based model used in [4] differs from ours, the work utilized the wire-based features, which did not effectively capture the spatial characteristics of layout congestion. Table 2 shows the experimental results assessing the accuracy of timing prediction for net reorder. “MAE” denotes the mean absolute error metric and the “MaxAE” means the max absolute error metric. "SCC" refers to the Spearman correlation coefficient, which is employed to assess the correlation

between two variables. In this study, it is utilized to measure the correlation between wire delays during the GR and DR stages, as well as the correlation between the wire delays predicted by different models and the actual wire delays after DR. Column "Post-GR" shows the mean absolute error and the max absolute error between the estimated wire delay at post-GR using the method introduced in Section 4.1 with the ground truth wire delay calculated at post detailed routing. The "Machine learning" column shows the results using the proposed prediction model in [4] with the features in the work. The "KAN-UNet (ours)" column exhibits the results of the KAN-UNet timing prediction model. As for the trained circuits, compared to the wire delay estimated at post-GR stage, the proposed prediction framework achieves a 30.8% and 41.5% improvement in terms of MAE and MaxMAE, respectively. On the test circuits, our model achieved improvements of 34.6% in MAE and 25.4% in MaxAE, respectively. Although the machine learning-based method performed well on the training set, its performance on the test circuits showed a 26.9% improvement in MAE, which is slightly lower than our model's results. Additionally, its MaxAE decreased by 6.0%. This suggests that the improvements observed on the training set may be due to some degree of overfitting.

In this study, we assess the correlations between wire delays at various stages and their predictions from different models by employing the Spearman correlation coefficient (SCC), which is defined as:

$$\mathrm{SCC} = 1 - \frac{6 \sum_{i=1}^{n} d_i^2}{n\left(n^2 - 1\right)} \tag{13}$$

Where d_i represents the difference in ranks for each pair of data points, n is the total number of data points.

The baseline correlation, measured by the Spearman correlation coefficient (SCC), between wire delays during the GR and DR stages, yields average SCC values of 0.79 on the training set and 0.53 on the test set. In contrast, a Machine Learning-based model achieves significantly higher average SCC values of 0.98 on the training set and 0.70 on the test set when predicting wire delays compared to the actual DR wire delays. Meanwhile, the proposed KAN-UNet model demonstrates average SCC values of 0.87 on the training set and 0.73 on the test set for the same task. The high SCC value of 0.98 achieved by the machine learning-based model on the training set suggests potential overfitting, as the model may have learned to capture specific patterns unique to the training circuits, which do not generalize well to the test circuits, resulting in a lower SCC of 0.70. On the other hand, the KAN-UNet model exhibits a more balanced performance, with a slightly lower but still strong SCC of 0.87 on the training set and a higher SCC of 0.73 on the test set compared to the machine learning-based model. This indicates that the KAN-UNet model generalizes better to unseen circuits, striking a more effective trade-off between fitting the training circuits and maintaining robustness on the test circuits.

For a detailed demonstration of the improvements, Fig. 14 presents the comparative results on the training circuit ysyx_0, the left figure compares the wire delays at the GR stage and the wire delays predicted by the proposed KAN-UNet model against the corresponding post-DR wire delays. We can observe that the wire delay distribution predicted by KAN-UNet model is more closely aligned around the $y = x$ line, demonstrating better consistency. The right figure presents a comparison of the wire delays predicted by the machine learning model and the KAN-UNet model against the corresponding DR wire delays. The wire delays predicted by the machine learning model align more closely with the $y = x$ line, indicating a potential issue of overfitting, as evidenced by the lower correlation on the test set. This phenomenon highlights the importance of balancing model complexity and generalization to ensure robust performance across different circuits. Similarly, Fig. 15 shows the comparison results for the test circuit ysyx_6, the wire delays predicted by the proposed KAN-UNet model are more closely clustered around the $y = x$ line when compared to the DR wire delays, significantly improving timing consistency. In contrast to the predictions from the

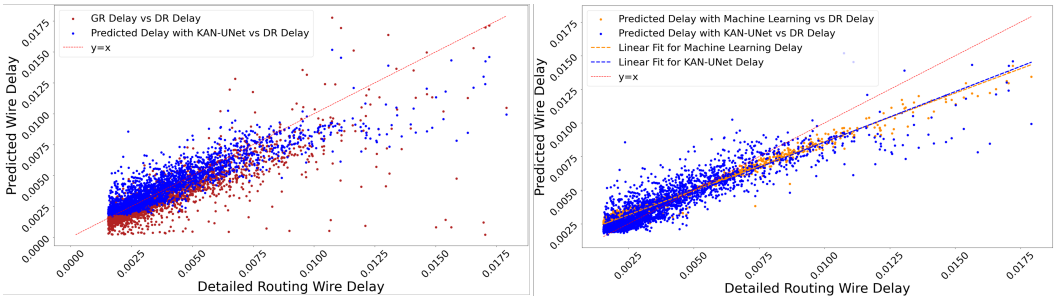


Fig. 14. Comparison of global routing wire delay and KAN-UNet predicted wire delay with post-DR wire delay (left figure) and the comparison of machine learning and KAN-UNet predicted delays with DR wire delay (right figure) for trained circuit ysyx_0.

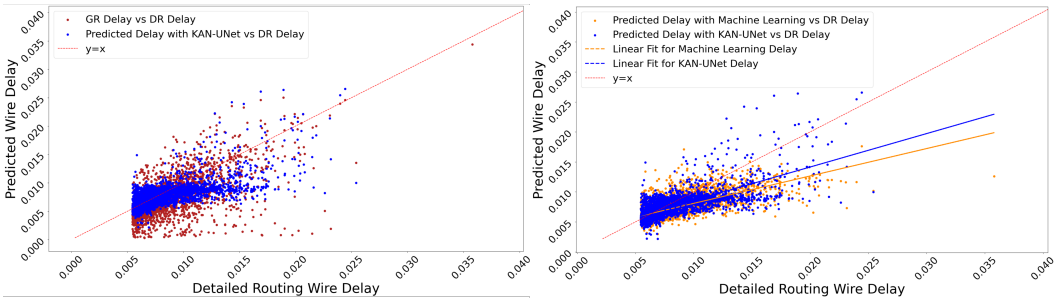


Fig. 15. Comparison of global routing wire delay and KAN-UNet predicted wire delay with post-DR wire delay (left figure) and the comparison of machine learning and KAN-UNet predicted delays with DR wire delay (right figure) for test circuit ysyx_6.

machine learning model, our approach demonstrates enhanced consistency. Further linear fitting of the predicted wire delays reveals that the results from the proposed model exhibit superior consistency. It is also important to note that during the net-reorder process, we need to rank the nets and assign priorities based on wire delay. Accordingly, Fig. 14 and Fig. 15 present the delay comparison results for thousands of nets that participated in the reorder optimization strategy.

The "DR+STA" column represents the time required to obtain post-DR delays by performing detailed routing followed by STA during the post-GR stage. The "KAN-UNet" column, on the other hand, refers to the inference time using the network proposed in this paper during the post-GR stage. Notably, the results indicate an average speedup of 1287 × on the test circuits, with a maximum improvement of 1446 ×. The acceleration becomes increasingly pronounced as the number of nets in the circuit grows, highlighting its necessity for efficient iterative optimization.

6.3 Timing Improvement on Timing Optimization

To validate the effectiveness of the optimization strategies proposed in this work, we conducted ablation experiments. Table 3 presents a comparison of TNS and WNS after detailed routing under three different conditions: using the basic iRT tool for global routing and detailed routing, applying the net reorder strategy, and combining both net reorder and net re-route strategies. These results help demonstrate the impact of each strategy on the timing performance of the circuits after DR.

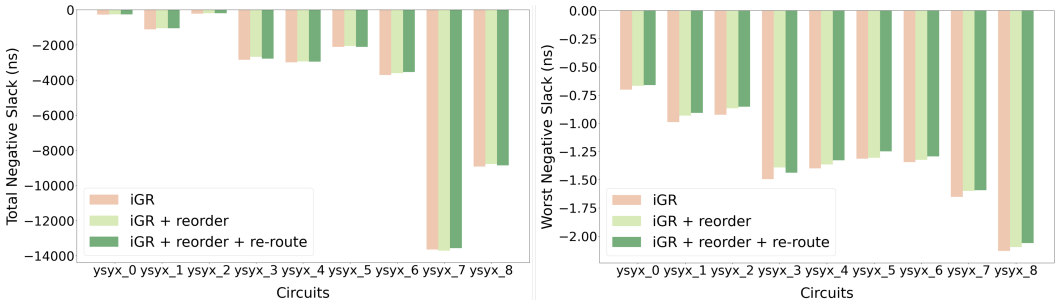


Fig. 16. The comparison of TNS (left) and WNS (right) under three different scenarios: the baseline using iRT, the results after incorporating the net reorder strategy, and the further improvements achieved by combining both the net reorder and re-route strategies.

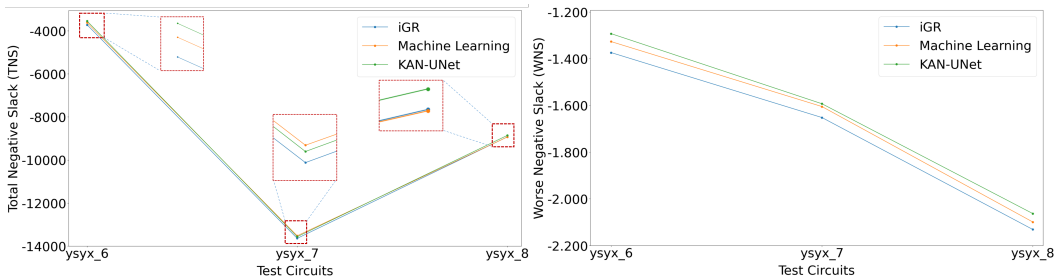


Fig. 17. The comparison of TNS (left) and WNS (right) under the combined strategies of net reorder and net re-route. The results are compared using the baseline iRT as a reference, along with the results guided by the machine learning method [4] and the proposed KAN-UNet model for timing optimization.

The iRT results show consistent baseline values for both TNS and WNS, normalized to 1.0000 across all circuits.

In order to improve readability, the values within the "TNS (ns)" column were rounded before making comparisons. The data in "ΔTNS(%)" and "ΔWNS(%)" columns respectively indicate the improvement ratios of TNS and WNS, compared to the baseline iRT, after applying the optimization strategies. On the training circuits, the implementation of the net reorder strategy resulted in noticeable improvements in timing performance, with TNS and WNS increasing by 4.1% and 4.9%, respectively, following detailed routing. When the re-route strategy was subsequently integrated into the process, the overall improvements in TNS and WNS reached 4.4% and 6.3%, and the maximum improvements reached 11.1% and 8.2%, respectively. After incorporating the re-route step, the overall trend remained positive, indicating a substantial improvement in timing performance. For the test circuits, after applying the reorder strategy, TNS and WNS improved by 1.3% and 2.8%, respectively. When further combined with the re-route strategy, these improvements increased to 2.0% and 4.2%, respectively, with the maximum gains reaching 4.7% and 5.9%. This suggests that combining net reorder with net re-route effectively improves the circuit's timing characteristic, it also demonstrates that the two optimization strategies also possess strong generalization capabilities.

From the data in the Table 3, the total wirelength "T-WL", the total overflow "T-OF" and the design rule violation "#DRV" are listed. After the net reorder optimization, both total wire length and total overflow did not increase, indicating that the impact of reorder strategy on total wire

Table 3. Comparison of TNS (Total Negative Slack) and WNS (Worst Negative Slack) after detailed routing under three different conditions: using the basic iRT tool for global routing, applying the net reorder strategy, and combining both net reorder and net re-route strategies. The proposed KAN-UNet model is utilized for timing prediction to guide the two timing optimization strategies. "T-WL" denotes the total wire length, while "T-OF" signifies the total overflow. "#DRV" means the design rule violation.

Circuit	iRT [14]					iRT+reorder								iRT+reorder+re-route							
	T-WL	T-OF	#DRV	TNS(ns)	WNS(ns)	T-WL	T-OF	#DRV	ΔDRV(%)	TNS(ns)	ΔTNS(%)	WNS(ns)	ΔWNS(%)	T-WL	T-OF	#DRV	ΔDRV(%)	TNS(ns)	ΔTNS(%)	WNS(ns)	ΔWNS(%)
ysyx_0	938443	6442	214	−279	−0.702	938442	6442	234	9.3	−269	3.6	-0.668	4.8	940601	6678	248	15.9	−248	11.1	-0.661	5.8
ysyx_1	892956	4710	161	−1122	−0.989	892917	4710	154	−4.3	−1054	6.1	-0.933	5.7	894387	4944	168	4.3	−1057	5.8	-0.908	8.2
ysyx_2	965610	46	319	−214	−0.925	965765	46	319	0.0	−204	8.9	-0.867	6.3	967833	42	320	0.3	−202	9.8	-0.853	7.8
ysyx_3	1532578	14026	489	−2851	−1.495	1532530	14026	514	5.1	−2684	5.9	-1.391	6.9	1534469	14492	536	9.6	−2780	2.5	-1.438	3.8
ysyx_4	3098860	16818	333	−2990	−1.400	3099174	16818	363	9.0	−2936	1.8	-1.364	2.6	3102180	17334	360	8.1	−2951	1.3	-1.327	5.2
ysyx_5	2892361	13822	217	−2117	−1.345	2892394	13822	238	9.7	−2058	2.8	-1.306	2.9	2895863	14002	223	2.8	−2116	0.0	-1.249	7.1
Average	1.0000	1.0000	1.000	1.000	1.000	1.0000	1.0000	1.048	4.8	0.959	4.1	0.951	4.9	1.0016	1.0127	1.068	6.8	0.956	4.4	0.937	6.3
ysyx_6	3959577	18936	414	−3716	−1.374	3959517	18936	473	14.3	−3615	2.7	-1.324	3.6	3960936	19218	511	23.4	−3542	4.7	-1.293	5.9
ysyx_7	4183184	21548	579	−13645	−1.652	4183200	21548	534	−7.8	−13707	−0.4	-1.600	3.1	4185314	21632	587	1.4	−13567	0.6	-1.593	3.6
ysyx_8	5549399	20642	469	−8928	−2.131	5549494	20642	475	1.3	−8782	1.6	-2.096	1.6	5552401	21090	480	2.3	−8857	0.8	-2.063	3.2
Average	1.0000	1.0000	1.000	1.000	1.000	1.0000	1.0000	1.026	2.6	0.987	1.3	0.972	2.8	1.0005	1.0135	1.091	9.1	0.980	2.0	0.958	4.2

Table 4. Comparison of TNS and WNS after detailed routing under two different conditions: applying the net reorder strategy, and combining both net reorder and net re-route strategies. The baseline is based on the results of iRT from Table 3. The machine learning method [4] is utilized for timing prediction to guide the two timing optimization strategies. "T-WL" denotes the total wire length, while "T-OF" signifies the total overflow. "#DRV" means the design rule violation.

Circuits	iRT+reorder							
	T-WL	T-OF	#DRV	△DRV(%)	TNS (ns)	△TNS(%)	WNS (ns)	ΔWNS(%)
ysyx_6	3959839	18936	468	13.0	−3668	1.3	-1.338	2.6
ysyx_7	4183207	21548	576	−0.5	−13710	−0.5	-1.621	1.9
ysyx_8	5549343	20642	493	5.1	−8758	1.9	-2.126	0.2
Average	1.0000	1.0000	1.059	5.9	0.991	0.9	0.984	1.6
Circuits	iRT+reorder+re-route							
	Tot-WL	Tot-OF	#DRV	△DRV(%)	TNS (ns)	△TNS(%)	WNS (ns)	△WNS(%)
ysyx_6	3961037	19218	432	4.3	−3614	2.7	-1.327	3.4
ysyx_7	4185247	21632	596	2.9	−13523	0.9	-1.605	2.8
ysyx_8	5552444	21090	481	2.6	−8933	−0.1	-2.099	1.5
Average	1.0005	1.0135	1.033	3.3	0.988	1.2	0.974	2.6

length and total overflow is negligible. After applying the re-route optimization, on the training circuits, the total wire length and total overflow increased by 0.16% and 1.27%, respectively, while on the test circuits, they increased by 0.05% and 1.35%, respectively. Meanwhile, following the application of both reorder and re-route optimizations, the value of WNS increased by 6.3% and 4.2% on the training and test circuits, respectively. Compared to the WNS improvement, the increase in total wire length and total overflow is deemed acceptable. When timing optimization is guided by the KAN-UNet model proposed in this work, after applying the reorder strategy, the Design Rule Violations (DRVs) increased by an average of 4.8% on the training set and 2.6% on the test set. Further integration of the re-route strategy resulted in DRVs increasing to 6.8% on the training circuits and 9.1% on the test circuits. Among the three tested circuits, the DRVs increased by 23.4% (from 414 to 511), 1.4% (from 579 to 587), and 2.3% (from 469 to 480), respectively. Although the average increase was 9.1%, the maximum DRV value of 587 and the maximum increment of 97 both remains relatively small. This indicates that the violations are manageable and can be easily fixed without significantly impacting the overall design flow. The slight rise in DRVs is a trade-off for the timing improvements achieved, and the overall number of violations remains low, ensuring minimal additional effort for correction.

To validate the effectiveness of the proposed KAN-UNet model, the wire delays predicted by the machine learning model [4] were utilized to guide the reorder and re-route strategies for three test circuits: ysyx_6, ysyx_7, and ysyx_8. The results are shown in Table 4. After applying the reorder strategy, the TNS and WNS improved by 0.9% and 1.6%, respectively, which are lower than the improvements of 1.3% and 2.8% achieved with KAN-UNet model. While the wirelength and overflow remained unchanged, the DRVs increased by 5.9%. With the integration of the re-route optimization strategy, the TNS and WNS further improved to 1.2% and 2.6%, respectively, still lower than the 2.0% and 4.2% enhancements guided by the KAN-UNet model. The wirelength and overflow increased by 1.35%, and the DRV rose by 3.3%. Although the machine learning-based timing prediction model can also contribute to timing optimization, the proposed KAN-UNet outperforms the machine learning-based model in timing optimization, providing more accurate wire delay predictions. This leads to better reorder and re-route decisions, further achieving superior WNS and TNS improvements. This highlights the effectiveness and stability of KAN-UNet model, further emphasizing its advantages in achieving better timing optimization results.

To more clearly illustrate the improvements, Fig. 16 presents the enhancements in TNS and WNS under three different scenarios: the baseline using iRT, the results after incorporating the net reorder strategy, and the further improvements achieved by combining both the net reorder and re-route strategies. The figure effectively highlights the incremental benefits of applying a sequential combination of reorder and re-route strategies, confirming that these approaches significantly improve the circuit's overall timing performance. As shown in Fig. 17, to more vividly demonstrate the improvements in timing optimization achieved by the machine learning model and the proposed timing prediction model, we compared the enhancements in TNS and WNS resulting from the reorder and re-route strategies guided by the wire delays predicted using the machine learning model, and our proposed model. The results clearly illustrate the effectiveness of our proposed timing prediction model in guiding timing optimization strategies.

7 Conclusion

Routing is a pivotal stage in attaining timing closure in integrated circuit design. During the timing estimation process at the global routing stage, the lack of accurate detailed routing information and the underestimation of the effects of physical congestion can lead to significant discrepancies between the estimated timing and the actual post-detailed routing timing, ultimately resulting in potentially misleading assessments of the circuit's timing performance. In this work, we propose a KAN-UNet heterogeneous network to align timing between the global and detailed routing stage, taking into account global spatial information such as layout congestion. Additionally, we introduce two timing optimization strategies: net reorder for potential congestion-affected nets with significant delays and re-route critical nets with the selected timing-optimized topologies. Experimental results confirm the accuracy of our timing prediction network and the effectiveness of the optimization strategies.

References

[1] Hsien-Han Cheng, Iris Hui-Ru Jiang, and Oscar Ou. 2020. Fast and accurate wire timing estimation on tree and non-tree net structures. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218712

[2] Vidya A Chhabria, Jiang Hu, Andrew B Kahng, and Sachin S Sapatnekar. 2025. Toward an ML EDA commons: establishing standards, accessibility, and reproducibility in ML-driven EDA research. In *Proceedings of the 2025 International Symposium on Physical Design*. 93–101.

[3] Vidya A Chhabria, Wenjing Jiang, Andrew B Kahng, and Sachin S Sapatnekar. 2022. From global route to detailed route: ML for fast and accurate wire parasitics and timing prediction. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD*. 7–14.

[4] Vidya A. Chhabria, Wenjing Jiang, Andrew B. Kahng, and Sachin S. Sapatnekar. 2023. A machine learning approach to improving timing consistency between global route and detailed route. *ACM Trans. Des. Autom. Electron. Syst.* 29, 1, Article 18 (dec 2023), 25 pages. https://doi.org/10.1145/3626959

[5] Chris Chu and Yiu-Chung Wong. 2007. 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 (2007), 70–83.

[6] Ke-Ren Dai, Wen-Hao Liu, and Yih-Lang Li. 2011. NCTU-GR: Efficient simulated evolution-based rerouting and congestion-relaxed layer assignment on 3-D global routing. *IEEE Transactions on very large scale integration (VLSI) systems* 20, 3 (2011), 459–472.

[7] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings, 315–323.

[8] Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z. Pan, and Yibo Lin. 2022. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 1207–1212. https://doi.org/10.1145/3489517.3530597

[9] Guoqing He, Wenjie Ding, Yuyang Ye, Xu Cheng, Qianqian Song, and Peng Cao. 2024. An Optimization-Aware Pre-Routing Timing Prediction Framework Based on Heterogeneous Graph Learning. In *Proceedings of the 29th Asia*

and South Pacific Design Automation Conference (Incheon, Republic of Korea) *(ASPDAC '24)*. IEEE Press, 177–182. https://doi.org/10.1109/ASP-DAC58780.2024.10473937

[10] Xu He, Zhiyong Fu, Yao Wang, Chang Liu, and Yang Guo. 2022. Accurate timing prediction at placement stage with look-ahead RC network. In *Proceedings of the 59th ACM/IEEE Design Automation Conference* (San Francisco, California) *(DAC '22)*. Association for Computing Machinery, New York, NY, USA, 1213–1218. https://doi.org/10.1145/3489517.3530598

[11] Jiang Hu and Sachin S Sapatnekar. 2001. A survey on multi-net global routing for integrated circuits. *Integration* 31, 1 (2001), 1–49.

[12] Zhipeng Huang, Zengrong Huang, Simin Tao, Shijian Chen, Zhisheng Zeng, Liwei Ni, Chunan Zhuang, Weiguo Li, Xueyan Zhao, He Liu, et al. 2024. AiEDA: an open-source AI-native EDA library. In *2024 2nd International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 794–795.

[13] Andrew B Kahng, Lutong Wang, and Bangqi Xu. 2021. TritonRoute-WXL: The open-source router with integrated DRC engine. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 4 (2021), 1076–1089.

[14] Xingquan Li, Zengrong Huang, Simin Tao, Zhipeng Huang, Chunan Zhuang, Hao Wang, Yifan Li, Yihang Qiu, Guojie Luo, Huawei Li, et al. 2024. iEDA: An Open-source infrastructure of EDA. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 77–82.

[15] Rongjian Liang, Anthony Agnesina, Geraldo Pradipta, Vidya A Chhabria, and Haoxing Ren. 2023. Circuitops: An ml infrastructure enabling generative ai for vlsi circuit optimization. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–6.

[16] He Liu, Shengkun Wu, Simin Tao, Biwei Xie, Xingquan Li, and Ge Li. 2023. Accurate Timing Path Delay Learning Using Feature Enhancer with Effective Capacitance. In *2023 International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 280–285.

[17] 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.

[18] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. 2018. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in neural information processing systems* 31 (2018).

[19] Siting Liu, Ziyi Wang, Fangzhou Liu, Yibo Lin, Bei Yu, and Martin Wong. 2023. Concurrent sign-off timing optimization via deep steiner points refinement. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC56929.2023.10247794

[20] Ziming Liu, Yixuan Wang, Sachin Vaidya, Fabian Ruehle, James Halverson, Marin Soljačić, Thomas Y Hou, and Max Tegmark. 2024. Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756* (2024).

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*. Springer, 234–241.

[22] Prashant Saxena, Rupesh S Shelar, and Sachin Sapatnekar. 2007. *Routing congestion in VLSI circuits: estimation and optimization*. Springer Science & Business Media.

[23] Ziyi Wang, Siting Liu, Yuan Pu, Song Chen, Tsung-Yi Ho, and Bei Yu. 2023. Restructure-tolerant timing prediction via multimodal fusion. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC56929.2023.10247802

[24] Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, Jiang Hu, Chen-Chia Chang, Jingyu Pan, and Yiran Chen. 2022. Preplacement net length and timing estimation by customized graph neural network. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 41, 11 (2022), 4667–4680.

[25] Zhiyao Xie, Rongjian Liang, Xiaoqing Xu, Jiang Hu, Yixiao Duan, and Yiran Chen. 2021. Net2: A graph attention network method customized for pre-placement net length estimation. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 671–677.

[26] Yue Xu and Chris Chu. 2011. MGR: Multi-level global router. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 250–255.

[27] Yue Xu, Yanheng Zhang, and Chris Chu. 2009. FastRoute 4.0: Global router with efficient via minimization. In *2009 Asia and South Pacific Design Automation Conference*. IEEE, 576–581.

[28] Tai Yang, Guoqing He, and Peng Cao. 2022. Pre-routing path delay estimation based on Transformer and residual framework. In *Proceedings of the 27th Asia and South Pacific Design Automation Conference* (Taipei, Taiwan) *(ASPDAC '22)*. IEEE Press, 184–189. https://doi.org/10.1109/ASP-DAC52403.2022.9712484

[29] Zhisheng Zeng, Jikang Liu, Zhipeng Huang, Ye Cai, Biwei Xie, Yungang Bao, and Xingquan Li. 2024. Net resource allocation: a desirable initial routing step. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.