iPCL-M: Pre-training of Chip Layout for Metrics Evaluation and Optimization

Xinhua Lai[1], He Liu[2], Weiguo Li[3], Yihang Qiu[1], Miao Liu[1], Simin Tao[3], Xingquan Li[4,*], Jungang Xu[1,*]
[1]University of Chinese Academy of Sciences, [2]Peking University, [3]Peng Cheng Laboratory, [4]Southeast University

Abstract

Physical design is a critical step in the electronic design automation (EDA) process, where the gate-level netlist from logic synthesis is converted into a GDS-II file. Recently, AI methods have been increasingly employed to address challenges in physical design. This paper proposes a new paradigm that integrates routing metric evaluation and optimization on chip layouts, forming a "Generate–Evaluate–Optimize –Generate" (GEOG) closed-loop. In our experiments, the pre-routing generation model achieves an mean relative error (MRE) of only 1%, with a mean absolute error (MAE) below one movement unit. Post-processing reduces both MRE and MAE to zero while ensuring connectivity. The evaluation model performs comparably to iSTA on the test set, with a 336× speedup. Furthermore, the optimization strategy effectively improves wirelength, delay, slew, R, and C by 10.49%, 13.03%, 14.85%, 10.51%, and 6.46%, respectively, and produces routing results superior to commercial tools in key metrics.

CCS Concepts

• **Hardware → Electronic design automation**.

Keywords

Metric evaluation, timing optimization, foundation model for layout, pre-training for routing generation

1 Introduction

Physical design is a critical component of Electronic Design Automation (EDA), tasked with converting a gate-level netlist into a manufacturable GDS-II file. Routing, a key stage in physical design, determines the interconnection paths between components and directly impacts timing, power, and overall chip reliability. Traditionally, designers rely on **pre-routing techniques** during placement to estimate timing performance and guide subsequent optimization. Pre-routing evaluation aims to quickly approximate signal propagation paths before full routing, enabling early identification of potential timing violations and performance bottlenecks [32].

Conventional pre-routing methods, such as HPWL and FLUTE [9], efficiently estimate wirelength, which is then used by a static timing analysis (STA) engine [14, 18] for timing evaluation and optimization. These methods are typically categorized as: (1) **Net-based**, which adjust net weights [5, 6, 10, 11, 20, 26] or net constraints [12, 17, 24]; and (2) **Path-based**, which extract and optimize critical timing paths to improve overall design performance [8, 16, 21, 29]. Recently, **machine learning (ML)-based approaches** have emerged to accelerate timing evaluation. Some construct timing prediction models post-placement or routing for rapid estimation [1, 7, 13, 22], while others model timing as a graph learning problem using graph neural networks (GNNs) or graph transformers [23, 35].

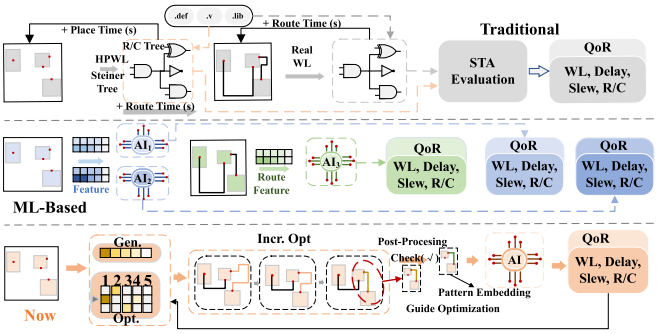


Figure 1: Comparison of traditional, ML-based, and our methods.

As illustrated in Figure 1, existing methods face several challenges. Traditional approaches require R/C Tree construction and depend heavily on accurate pre-routing wirelength estimates. While later-stage estimates are more precise, they are increasingly time-consuming. STA-driven flows also incur significant computational overhead, limiting scalability. ML-based models improve efficiency but are often task-specific and passive—they generalize poorly and cannot directly optimize pre-routing.

These limitations motivate methods that can rapidly evaluate and actively optimize pre-routing paths. Inspired by Transformers [30] and GPT models [4], large models have recently been applied in EDA, including MOSS [31] for RTL-level timing optimization, LLM-assisted error analysis [27], and automatic script generation systems like ChatEDA [34]. Additionally, Ref. [2] introduced a large chip model (LCM) for chip design, demonstrating early potential and providing insights for other EDA tasks. Motivated by these developments [19], we propose a framework that integrates fast metric evaluation with pre-routing optimization, enabling efficient and high-quality path generation. In this paper, we propose **iPCL-M1**, a pretrained pre-routing generation model for chip layout metric evaluation and optimization. iPCL-M1 first defines a **chip symbolic space** representing all routing connections within a layout. Using large-scale routing data generated by commercial EDA tools, we train a pretrained model to learn routing generation logic, which is then applied to the pre-routing generation task. Simultaneously, a dedicated **evaluation model** is trained on the characterized routing data to directly predict key timing metrics from pre-routing information. We further propose an **optimization strategy** that integrates the generation and evaluation models into a **"Generate–Evaluate–Optimize–Generate"** (GEOG) closed-loop framework (as shown in Figure 1). This framework bridges the gap between evaluation and optimization, enabling synergistic improvements in pre-routing generation and timing evaluation.

The main contributions of this paper are summarized as follows:

- **A new AI4EDA paradigm:** We propose a symbolic representation space for chip layout, where routing *Patterns* built from primitive symbols uniformly describe pin-to-pin connections and timing paths, enabling a unified paradigm for pre-routing generation, evaluation, and optimization.
- **A pretrained generative model for pre-routing:** We train a generative model on commercial designs to produce intermediate routing patterns from pin coordinates, achieving

DAC '26, Long Beach, CA, USA

ACM ISBN 979-8-4007-2254-7/2026/07
https://doi.org/10.1145/3770743.3804232

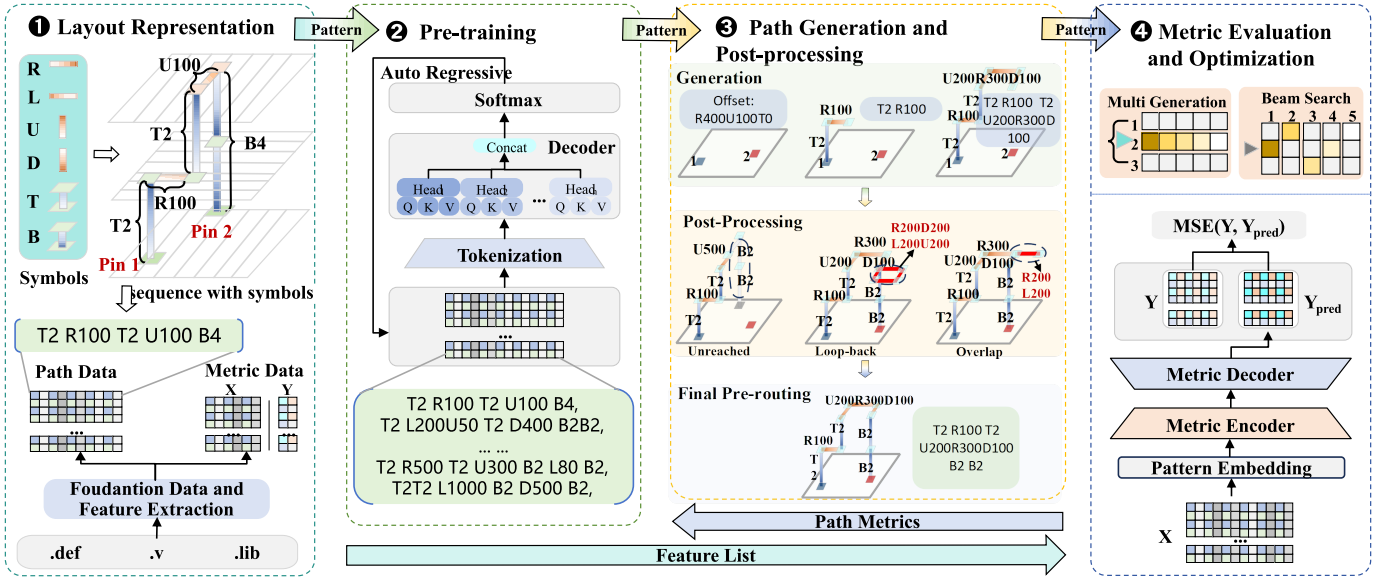


Figure 2: Overall framework of iPCL-M.

pre-routing with an average endpoint error below 1% and within 1 movement unit.

- **An efficient and generalizable evaluation model:** We develop a lightweight model predicting timing metrics from compact features without full-chip information, outperforming traditional STA on key metrics with nearly 100× speedup.
- **A pre-routing optimization model:** By integrating the evaluation model into generation, we realize real-time route optimization, improving timing, power, and parasitics. The optimized model improves performance by 10% over the baseline and exceeds commercial tools by about 5%.

2 Problem and Framework

2.1 Problem Formulation

Because the quality of timing paths critically impacts overall chip performance, we focus on the evaluation and optimization of net- and path-related metrics. As discussed in the introduction, traditional timing optimization relies heavily on accurate R/C-tree construction, which requires completing placement and routing (PR) to achieve high precision but leads to substantial inefficiency (see Figure 1). Meanwhile, existing ML-based approaches are decoupled from the routing process and thus cannot support pre-routing optimization. Motivated by these limitations, we formally define the following problem:

Problem 1 (Chip Layout 3D Pre-routing Generation and Optimization). *Given a chip layout and a set of pin pairs, the goal is to efficiently generate, evaluate, and optimize pre-routing solutions that approximate real routing behavior while achieving desirable timing and signal integrity metrics. We decompose this problem into three sub-tasks:*

(1) ***Pre-routing Generation:*** *Train a model on commercial routing data to generate valid 3D pre-routing paths between two pins.*
(2) ***Metric Evaluation:*** *Build a real-time evaluation model that predicts key metrics—such as wirelength, delay, slew, resistance, capacitance, and power—from pre-routing paths without invoking full STA.*
(3) ***Pre-routing Optimization:*** *Integrate generation and evaluation into a closed-loop process where the evaluation model provides immediate feedback to refine routing quality.*

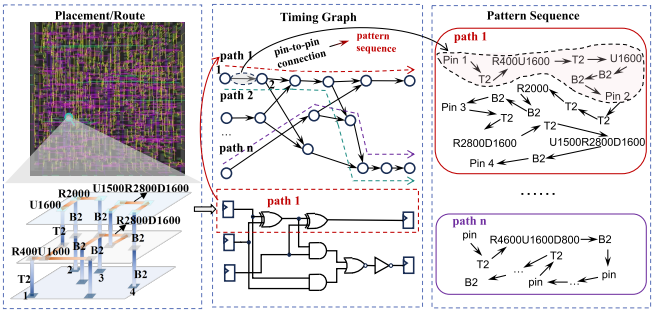


Figure 3: Pattern sequence as a unified representation for routing and timing analysis.

To efficiently solve the above three sub-tasks—which are inherently interdependent—we require a unified framework rather than isolated models. In particular, pre-routing generation provides structural patterns for metric evaluation, while metric feedback must guide subsequent generation and refinement. Therefore, we develop an integrated framework that tightly couples these components into a coherent pipeline. We introduce this framework below.

2.2 Framework of iPCL-M

Building on the problem formulation above, our framework is designed to jointly address path generation, metric evaluation, and optimization within a single closed-loop system. The overall structure is illustrated in Figure 2 and consists of four components: ❶ Layout Representation, ❷ Pretraining, ❸ Path Generation and Post-processing, and ❹ Metric Evaluation and Optimization. In the layout representation stage, we construct a **symbolic space** to encode routing connections, with the concrete encoding and serialization scheme shown in Figure 3. Using this unified representation, we extract path and metric data from design files such as `design.def`, `netlist.v`, and `library.lib` to construct standardized datasets. The pretrained model learns transferable pre-routing capability from the symbolic connection corpus. During the component ❸, the model generates pre-routing paths, followed by post-processing to correct loops, overlaps, and unreached endpoints. A metric evaluation model then provides real-time feedback to guide refinement. Together, these components form a **GEOG** closed loop that unifies generation and evaluation, enabling efficient and performance-driven pre-routing.

Table 1: Symbols used for chip layout representation.

Symbol	Meaning
R	Move Right in layout plane
L	Move Left in layout plane
U	Move Up in layout plane
D	Move Down in layout plane
T	Move Up in vertical direction (out-of-plane)
B	Move Down in vertical direction (out-of-plane)
0–9	Numerical values representing movement distances; digits can be combined to form multi-digit distances

3 Pre-training of Path Generation

Traditional timing analysis typically extracts a post-placement or post-routing netlist, reconstructs the logical circuit, and builds a timing graph for evaluation. Unlike prior methods, our framework further encodes the timing graph as a *Pattern Sequence* based on the layout representation in Figure 2. This unified sequence-level abstraction harmonizes the input modality of generation, evaluation, and optimization, enabling seamless interaction among components and transforming previously disjoint stages into an integrated pipeline. In this section, we detail the **chip layout representation**, the **pre-trained generation model**, the **pattern formulation rules**, and the **post-processing procedures**.

3.1 Chip Layout Representation

3.1.1 Symbolic Space for Chip Layout. In natural language, letters and punctuation compose words, which further form sentences; NLP models leverage this compositional structure for prediction and generation. Analogously, we construct a symbolic representation for chip layouts by encoding their fundamental elements. In a layout, each inter-cell connection is defined by the coordinates of the source pin, the target pin, and the intermediate routing path. We serialize this path as a sequence of directional movements and distances. The six movement types correspond to in-plane directions—R/L (Right/Left) and U/D (Up/Down)—and out-of-plane directions—T/B (Top/Bottom). Assuming each pin is located at its cell center, a connection can be expressed as a combination of these directional steps, referred to as a *Pattern*. These symbols constitute the complete symbolic space for chip layouts, as summarized in Table 1 and illustrated in Figure 3.

3.1.2 Path Representation and Serialization. In Section 3.1.1, we introduced the basic chip layout symbols as well as the more complex concept of *Patterns*. The next question to consider is how to represent the entire routing path between two pins. Intuitively, a routing path is generated segment by segment, similar to how text is composed character by character. Following this analogy, we represent a pin-to-pin routing path as a sequence of "words", where each "word" corresponds to a *Pattern*. In this way, the complete routing path can be serialized into a *Pattern sequence*, enabling a natural and structured representation of the connection between pins (as shown in Figure 3).

3.1.3 Tokenization Technology. Using the previous path representation method, across chip designs, we observe that the dataset contains over 135,000 unique *Patterns*. Directly using this vocabulary would cause the output dimension, model parameters, and memory/bandwidth cost to scale linearly, leading to substantial computational overhead (Table 2). To mitigate this, we apply Byte-Pair Encoding (BPE) [4], which merges frequently co-occurring symbol sequences into compact tokens. Owing to the strong repetitiveness of routing Patterns, BPE reduces the vocabulary from 135,000 Patterns to roughly 1,100 tokens—a $\sim$ 120× compression—improving output-layer efficiency. As shown in Figure 4, the compressed tokens exhibit a more balanced frequency distribution and a steeper power-law exponent compared to raw Patterns. Combined with Table 2, these results indicate that BPE-based tokens form a significantly more compact and efficient representation space, accelerating all computations and data transfers that scale with vocabulary size.

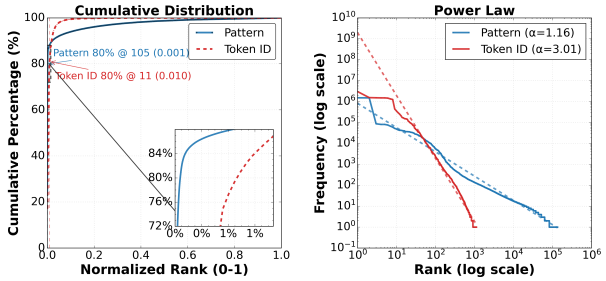


Figure 4: Comparison of Pattern and Token distributions.

Table 2: Comparison between Pattern and Token representations.

Metric	Pattern	Token
Vocabulary size (V)	134,649	1,118
Reduction ratio (V_p/V_t)	120.44× smaller with tokens	
Output compute ($O(B \cdot T \cdot V)$)	High	≈120.44× less
Output layer parameters ($H \cdot V + V$)	Large	≈120.44× less
Logits memory / bandwidth ($B \cdot T \cdot V$)	High	≈120.44× less

B = batch size, T = sequence length, V = vocabulary size, H = hidden size.

3.2 Pre-training Model

In the previous subsection, we described how a routing path is serialized into a sequence of Patterns, analogous to how words form a sentence. Following the formulation of language sequence models [3], we model pre-routing generation using the conditional probability:

$$\text{pat}_i = \arg\max_{\text{pat}_i} p(\text{pat}_i \mid \text{offset}, \text{pat}_1, \ldots, \text{pat}_{i-1}). \tag{1}$$

Following this, reasoning mechanisms such as chain of thought (CoT) have been shown to improve long-horizon sequence generation [33]. Inspired by this insight, we introduce a *Trace token* into the pattern sequence to enhance the model's ability to track and maintain global routing progress. The trace token explicitly indicates whether the current partial route has reached the target pin, thereby mitigating errors in long-sequence generation caused by uncertainty about reaching the destination. Given the offset (i.e., the target pin position R100U500T2), the trace token is initialized as R0U0T0 at the start of generation. The trace token shares the same format as the offset. It is updated with each newly generated Pattern, and once the trace token matches the offset, the model has successfully completed a routing path to the target pin. With the trace token incorporated, the generation process is defined over an interleaved sequence of patterns and trace tokens, resulting in the conditional probability:

$$p(s_i \mid s_{1:i-1}, \text{offset}), \tag{2}$$

where s_i denotes the i-th sequence item, $s_i = (\text{pat}_i, \text{trace}_i)$. To train the model, we minimize the negative log-likelihood of the predicted sequence items, yielding the standard cross-entropy loss:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta) = \arg\min_\theta [-\sum_{i=1}^{n} \log p_\theta(s_i \mid s_{1:i-1}, \text{offset})], \tag{3}$$

where $\mathcal{L}(\theta)$ denotes the cross-entropy between the predicted and ground-truth sequence items.

3.3 Generation, Criterion and Post-processing

3.3.1 Path Generation. Building on the previous sections, we train a pre-trained model on a generated dataset to predict the most probable *Pattern* sequence given an input offset. This model captures routing logic knowledge similar to that of commercial tools. We address two routing generation tasks: (i) generating paths between two instance pins to produce interconnection wiring (Figure 5-a) , and (ii) timing path generation, where the routing data from task (i) is used to

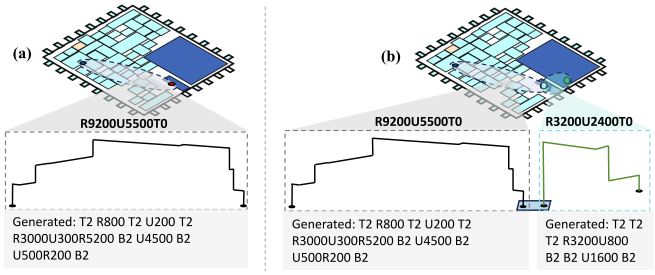


Figure 5: Generation results for pin-to-pin and path.

generate a timing path by decomposing it into two-pin routing tasks (Figure 5-b).

3.3.2 Connectivity Criterion for Path Generation. While visualization helps illustrate the generated paths, quantitative metrics are needed for comprehensive evaluation. Endpoint error is a suitable measure for assessing the accuracy of generated paths. A larger endpoint error indicates a deviation from the target, reflecting weaker generative capability, while a smaller error suggests better accuracy.

Mean Relative Error (MRE). The MRE compares the final coordinates of the predicted path with the true coordinates, reflecting the overall quality of the path. Relative errors along the x, y, z axes are:

$$e_x = |p_x - t_x|/D, \quad e_y = |p_y - t_y|/D, \quad e_z = |p_z - t_z|/D_z, \quad (4)$$

where $\varepsilon > 0$, $D = \max(\|\mathbf{t}\|_2, \ \|\mathbf{p}\|_2, \ \varepsilon)$, $D_z = \max(|t_z|, \ |p_z|, \ \varepsilon)$, $\|\mathbf{t}\|_2 = \sqrt{t_x^2 + t_y^2}$, $\|\mathbf{p}\|_2 = \sqrt{p_x^2 + p_y^2}$, $\mathbf{t} = (t_x, t_y, t_z)$, $\mathbf{p} = (p_x, p_y, p_z)$. The overall MRE is the average of these errors, which are $\bar{e}_x$,$\bar{e}_y$,$\bar{e}_z$.

Mean Absolute Error (MAE). The MAE measures the absolute distance between predicted and true endpoint coordinates. It helps evaluate discrepancies in path generation, and is defined as:

$$\text{MAE} = (1/N) \sum_{i=1}^{N} |p_i - t_i|, \quad (5)$$

where N is the total number of samples, and p_i, t_i are the predicted and ground-truth coordinates of the i-th sample.

3.3.3 Quality Criterion for Path Generation.

Routed Ratio (RR). The RR quantifies routing efficiency on planar layers and is defined as:

$$\text{routed\_ratio} = (L_x + L_y)/\max(|\Delta x| + |\Delta y|, \epsilon), \quad (6)$$

where L_x and L_y are the actual routed lengths along the X and Y axes, and Δx, Δy are the net offsets.

Via Count (VC). VC reflects the number of vias required to make inter-layer connections. In the Pattern Sequence, the total number of T and B operations indicates the VC:

$$\text{ViaCount} = N_{\text{T}} + N_{\text{B}}, \quad (7)$$

where N_{T} and N_{B} are the counts of T and B operations, respectively. A smaller VC is preferred for reduced cost and better signal integrity.

3.3.4 Model Generation Post Process. In the previous discussions, we introduced two error analysis methods to evaluate the model's generation accuracy and completeness, along with a quality criterion to assess routing performance. To ensure that each generated routing path correctly reaches its target pin, we apply a post-processing procedure after path generation (shown in Figure 6). The procedure addresses three types of abnormal cases: (1) *Look-back*, (2) *Overlap*, and (3) *Unreached*. Cases (1) and (2) correspond to short-circuit behaviors and must be removed, while case (3) is corrected by inserting minimal additional patterns to complete the path with the shortest feasible extension.

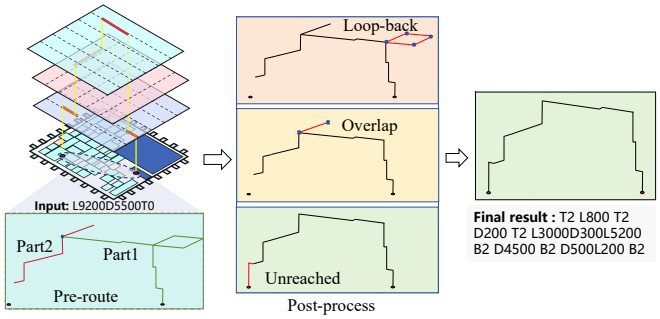


Figure 6: pre-routing generation with post process.

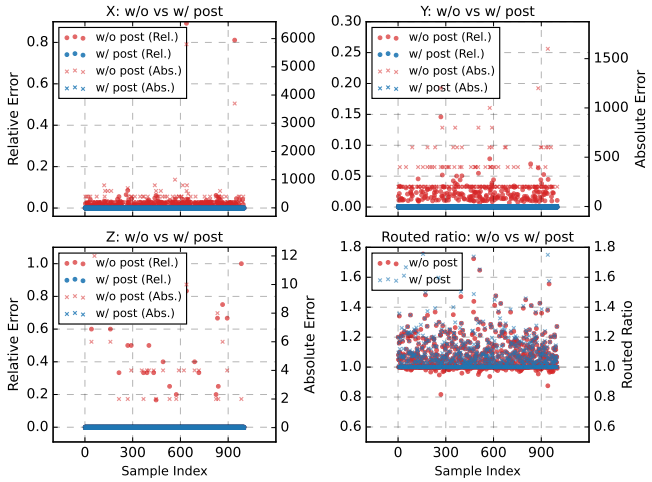


Figure 7: MRE, MAE and RR comparison of the generative model.

3.3.5 Analysis of Model Generative Capability. Based on the above error and quality analyses, we conduct experiments on design s5378, where 1000 randomly selected pin-to-pin pairs are used for path generation. As illustrated in Figure 7, we visualize the comparison of the MRE, MAE, and RR before and after post-processing using scatter plots. From the figure, it can be observed that before post-processing, the relative and absolute errors along the x- and y-axes are mostly concentrated around 0, with a considerable number of scattered points distributed between 0.0 and 0.2, and a few even exceeding 0.2. Along the z-axis, most points also cluster near 0, while the scattered points range from 0.0 to 1.0. After post-processing, however, all points of MRE and MAE across the x, y, and z axes converge to 0, with no remaining outliers. These results indicate that although the pre-routing generation model may fail on a few paths before post-processing, it generally achieves satisfactory performance. After post-processing, all pre-routes can successfully connect while maintaining the same connection quality.

4 Path Metric Evaluation and Optimization

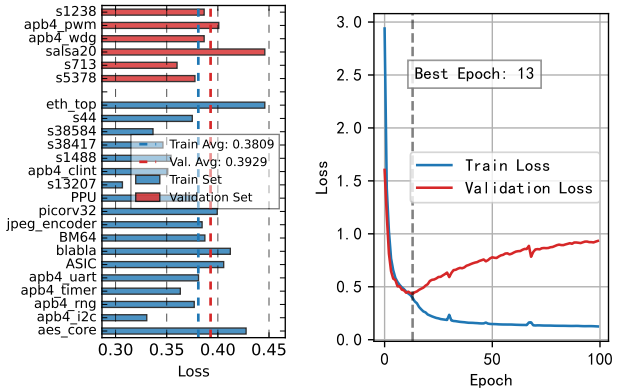
4.1 Evaluation

As shown in Figure 2, the evaluation module consists of four components: pattern embedding, encoder-decoder, PPA prediction, and objective function, providing feedback on PPA metrics and bridging pre-training and optimization. Pattern embedding represents patterns as dense vectors to reduce computational complexity. Inspired by SkipGram [25], a neural network (NN) with softmax models the conditional probability:

$$p(pat \mid pat_t; \theta) = \exp(h_t^\top u_{pat}) / \left(\sum_{s=1}^{P} \exp(h_t^\top u_{pat_s}) \right), \quad (8)$$

Table 3: Comparison of iEDA, iPCL-M(Trans.), and iPCL-M(LSTM) across different designs with all metrics.

Design	iEDA							iPCL-M(Trans.)							iPCL-M(LSTM)						
	WL	Delay	Slew	R	C	Power	Inf.(s)	WL	Delay	Slew	R	C	Power	Inf.(s)	WL	Delay	Slew	R	C	Power	Inf.(s)
aes	-	1.02	0.71	**0.04**	0.08	0.37	624.64	**0.32**	**0.25**	**0.37**	0.32	**0.04**	**0.04**	**1.81**	0.72	0.31	0.68	0.71	0.07	0.22	3.34
apb4_ps2	-	0.36	0.26	**0.01**	0.07	**0.01**	3.67	**0.32**	**0.09**	**0.15**	0.33	**0.03**	0.03	**0.04**	0.51	0.28	0.40	0.50	0.06	0.44	0.04
gcd	-	0.28	0.18	**0.01**	0.09	**0.00**	1.07	**0.15**	**0.08**	**0.15**	0.15	**0.05**	0.05	**0.02**	0.52	0.29	0.43	0.51	0.07	0.44	0.02
apb4_archinfo	-	**0.01**	**0.01**	**0.00**	**0.01**	**0.01**	1.34	**0.15**	0.10	0.12	0.15	0.03	0.03	**0.02**	0.62	0.29	0.49	0.60	0.09	0.44	0.02
s15850	-	0.30	0.20	**0.02**	0.05	0.03	72.51	**0.19**	**0.09**	**0.17**	0.19	**0.03**	**0.03**	**0.39**	0.55	0.29	0.45	0.53	0.07	0.43	0.17
s35932	-	0.23	0.19	0.12	0.04	0.09	204.54	**0.24**	**0.16**	**0.19**	**0.24**	**0.04**	**0.04**	**0.48**	0.54	0.26	0.44	0.53	0.07	0.37	0.50
s9234	-	0.10	0.07	**0.01**	0.04	**0.01**	35.40	**0.31**	**0.11**	**0.19**	0.31	0.04	0.04	**0.05**	0.56	0.29	0.47	0.54	0.08	0.44	0.05
avg.	–	0.33	0.23	**0.03**	0.05	0.07	134.74	**0.24**	**0.13**	**0.19**	0.24	**0.04**	**0.04**	**0.40**	0.57	0.29	0.48	0.56	0.07	0.40	0.59



(a) Training and validation loss. (b) Learning curve.

Figure 8: Training results of the 0.5B pre-trained path generation model.

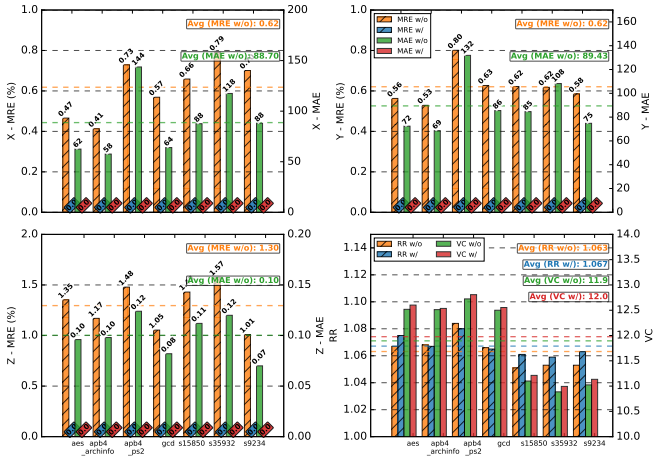


Figure 9: Comparison of MRE, MAE, RR, and VC w./w.o. Post-process.

where $h_t = W_{\text{in}}^\top x_t, u_{pat}$ is the output vector, and W_{in} is the learned embedding. The encoded pattern sequence is processed by a Transformer-based encoder-decoder model (alternatively LSTM). The encoder outputs serve as queries and keys for multi-head attention in the decoder, which generates outputs autoregressively. A final NN layer predicts the target features, and the model is trained using MSE loss.

4.2 Optimization

We now describe strategies to optimize the generated timing paths.

4.2.1 Multiple Sampling. For a given pin pair, we generate N candidate pre-routings using the generative model G_θ and select the one with the best overall metrics according to the evaluation function $\mathcal{R}$:

$$\text{pre-routing}^* = \arg \max_{r \in \{G_\theta(\text{pin pair})\}_{i=1}^N} \mathcal{R}(r), \tag{9}$$

where r denotes a generated routing candidate and pre-routing* is the selected routing with the highest evaluation score.

4.2.2 Beam Search. We formulate pattern generation as a conditional sequence modeling problem:

$$\text{pat}_i = \arg \max_{\text{pat}_i} p(\text{pat}_i \mid \text{offset}, \text{pat}_1, \ldots, \text{pat}_{i-1}). \tag{10}$$

A greedy strategy selects the highest-probability pattern at each step, producing a single sequence. However, exhaustive search for the globally optimal sequence is computationally intractable due to combinatorial explosion. To balance efficiency and optimality, we adopt **Beam Search** with Top-K selection, where each candidate sequence is scored by combining its likelihood, EDA-metric penalties, and length normalization:

$$\text{TopK} = \arg \max_{y_{1:T}} \frac{1}{(T+\alpha)^\gamma} \Big(\log P(y_{1:T}) - \boldsymbol{\beta}^\top \widetilde{\mathbf{t}}(y_{1:T}) \Big), \tag{11}$$

where $y_{1:T}$ denotes the generated sequence of length T, $P(y_{1:T})$ its probability, α is a small constant to avoid division by zero, γ is the length penalty exponent, and $\widetilde{\mathbf{t}}(y_{1:T}) = [\text{WL}, \text{Delay}, \text{Power}, R, C, \text{Slew}]^\top$ the normalized EDA metrics with corresponding weights $\boldsymbol{\beta}$. The Top-K operation retains the sequences with the highest scores for further expansion.

5 Experimental Results

In this section, we first describe the experimental environment setup, followed by an introduction to the pre-trained model results, path evaluation model, and finally the path optimization model experiments. We conclude by ablation studies to analyze the impact of certain variables. The experiments are conducted on a system equipped with 96 Intel(R) Xeon(R) Platinum 8168 CPUs @ 2.70GHz, 8 Tesla V100-SXM3-32GB GPUs, and 1.5TB of memory, running Ubuntu 22.04.5 LTS. All experiments are executed using Python (torch 2.4.1, CUDA 12.2). Data generation is performed with the help of AiEDA [15, 28].

5.1 Pretraining Results

This subsection presents an experimental analysis of the pre-training process, focusing on the model's training dynamics and performance across datasets. The model is based on the architecture described in Section 3.2, with a total of 0.5B learnable parameters. The dataset for each design is randomly split into training, validation, and test sets. The training set is used to optimize the model, the validation set is used to select the best checkpoint, and the test set is used to evaluate real-world performance. As shown in Figure 8-(b), the validation loss reaches its minimum at the 13th iteration, which is saved as the best model for subsequent path generation and evaluation. Beyond this point, the training loss continues to decrease while the validation loss shows overfitting. Figure 8-(a) illustrates the loss distribution of the selected model on the training and validation sets, showing that the validation loss distribution closely matches that of the test set.

5.2 Path Generation Results

The path generation model adopted for evaluation is the best-performing checkpoint from previous experiments. As detailed in Section 3.3.2, we use MRE and MAE to assess the accuracy and completeness of the

Table 4: Experimental Results across Methods: Baseline, Multiple Generation (M. Gen), Beam Search (BS), and commercial Innovus.

Design	Baseline					M. Gen					BS					Innovus				
	WL	Delay	Slew	R	C	WL	Delay	Slew	R	C	WL	Delay	Slew	R	C	WL	Delay	Slew	R	C
apb4_ps2	630.72	333.30	29.38	270.08	18.83	**566.32**	**285.19**	**23.97**	**242.49**	**17.82**	587.43	302.41	26.44	251.60	17.98	605.67	326.08	29.83	258.97	18.45
apb4_archinfo	650.71	347.18	30.58	278.56	20.62	**593.69**	**303.30**	**24.81**	**254.27**	**18.74**	593.75	308.18	26.13	254.27	18.66	585.13	308.22	25.44	250.38	19.37
gcd	571.16	295.72	24.41	244.22	17.82	**470.60**	**240.35**	**19.26**	**201.36**	**16.59**	473.78	241.30	19.20	202.75	16.65	561.13	292.99	24.08	239.96	19.31
s15850	562.44	294.25	24.95	240.33	17.84	**507.73**	**261.72**	**22.14**	**217.08**	**16.61**	507.73	261.72	22.14	217.08	16.61	564.26	310.33	28.83	240.91	19.54
s35932	768.85	418.67	37.04	328.87	21.18	695.69	372.52	32.85	297.54	20.05	695.69	372.52	32.85	297.54	20.05	**636.56**	**339.31**	**31.17**	**271.85**	**18.51**
s9234	501.56	257.01	22.13	214.61	16.69	**456.02**	**218.31**	**17.54**	**194.56**	16.34	467.86	228.36	18.69	199.64	16.44	516.24	260.25	20.84	220.60	18.43
aes	854.29	460.43	40.40	365.72	21.86	773.33	411.49	37.31	331.03	19.98	777.22	414.08	37.43	332.62	20.12	**808.74**	**442.80**	**42.35**	**346.01**	**21.53**
Avg.	648.53	343.79	29.84	277.48	19.26	**580.48**	**298.98**	**25.41**	**248.33**	**18.02**	586.21	304.08	26.13	250.78	18.07	611.11	325.71	28.93	261.24	19.31
Impr. (%)			–			10.49	13.03	14.85	10.51	6.46	9.61	11.55	12.45	9.62	6.18	5.77	5.26	3.05	5.85	-0.22

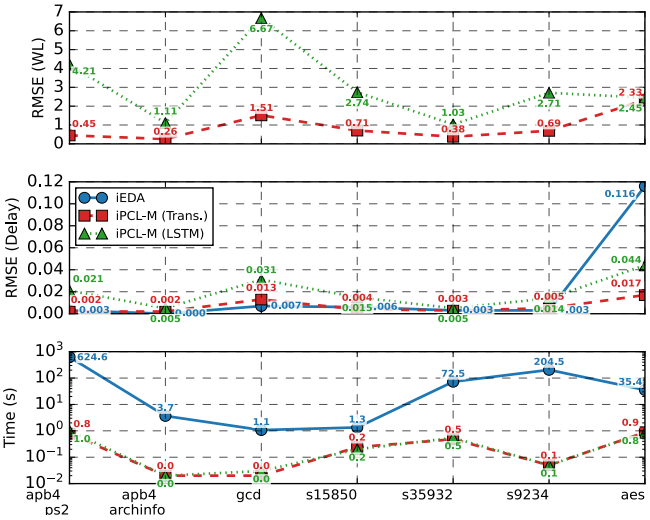


Figure 10: Path metrics comparison of iEDA, Transformer-based, and LSTM-based models on RMSE and evaluation time.

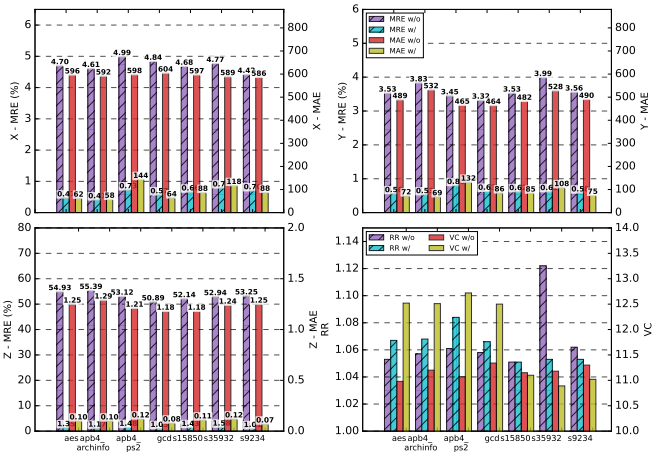


Figure 11: Comparison of MRE (%), MAE, RR and VC w./w.o. Trace.

generated paths. This subsection presents the MRE and MAE results across all test designs. Figure 9 shows the performance of the model with the best validation loss. Before post-processing, MRE remains below 1% in the X and Y directions, with averages of 0.62 and 0.62, respectively, and 1.30 in the Z direction. For MAE, the average values in X and Y are below 100, while in Z it is around 0.1. The routing quality metrics—RR and VC—have averages of 1.063 and 11.9, respectively. An RR value is close to 1, which indicates that the model already achieves high routing completion with satisfactory quality prior to post-processing. After post-processing, substantial improvements are observed in both MRE and MAE across all directions, with each metric reduced to zero. In contrast, RR and VC remain largely unchanged, confirming that routing quality and completion are preserved while MRE and MAE are fully minimized.

5.3 Metric Evaluation Results

For pin-to-pin and timing paths, Table 3 and Figure 10 report RMSE for predicted metrics (wirelength, slew, delay, R, C, and power) against Innovus ground truth. Wirelength is excluded for iEDA. While iEDA achieves the best results on resistance (R), the Transformer-based model (Trans.) generally outperforms other models across remaining metrics, except for `apb4_archinfo`, and also surpasses LSTM in wirelength prediction. For path delay, LSTM lags behind both iEDA and Trans., which achieve comparable results. Inference is significantly faster for Trans.: for large designs like `aes` and `s35932`, iEDA requires 624.6 s and 204.54 s, whereas Trans. completes in 1.81 s and 0.48 s, an average 336× speedup. These results demonstrate that Transformer-based iPCL-M provides accurate and efficient predictions for key routing metrics.

5.4 Optimization

In this subsection, we evaluate the performance of our generation optimization models. Table 4 compares multiple generation, Beam Search, the baseline model, and the commercial tool **Innovus**. The baseline represents results without optimization. Average improvements show that multiple generation enhances Wirelength, Delay, Slew, R, and C by 10.49%, 13.03%, 14.85%, 10.51%, and 6.46%, respectively, while Beam Search improves them by 9.61%, 11.55%, 12.45%, 9.62%, and 6.18%. Both methods significantly improve pre-routing metrics. Innovus outperforms the baseline on Wirelength, Delay, Slew, and R, with minimal change in C, but it is generally worse than our optimized methods, likely due to the lack of DRC considerations in our optimization.

5.5 Ablation Study

In Section 3.2, we introduced inference tokens. To evaluate their impact on path generation, we compared endpoint MRE, MAE, RR, and VC on the test set with and without inference tokens. As shown in Figure 11, incorporating inference tokens substantially reduces MRE and MAE by over 80% on the XY-plane and over 90% on the Z-axis, while RR and VC remain largely unchanged. These results demonstrate that inference tokens significantly enhance the model's path generation accuracy.

6 Conclusion

In this paper, we presented **iPCL-M1**, a unified framework that introduces a chip layout symbolic space and integrates pretrained pre-routing generation, lightweight metric evaluation, and closed-loop optimization. By linking generation and evaluation through the **GEOG** paradigm, iPCL-M1 enables efficient, high-quality pre-routing and timing estimation. Experiments demonstrate consistent improvements over baseline models and noticeable gains over commercial tools. Future work will extend this paradigm to broader routing tasks and richer metric objectives.

Acknowledgments

This work was supported by the Major Key Project of PCL (No. PCL2025A04), and the NSF of Fujian Province (No. 2024J09045).

References

[1] Jaehoon Ahn, Kyungjoon Chang, Kyu-Myung Choi, Taewhan Kim, and Heechun Park. 2024. DTOC-P: Deep-learning-driven Timing Optimization Using Commercial EDA Tool with Practicality Enhancement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 43, 8 (2024), 2493–2506.

[2] Lei and Chen, Yiqi Chen, Zhufei Chu, et al. 2024. Large circuit models: opportunities and challenges. *Science China Information Sciences* 67, 10 (2024), 200402.

[3] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research (JMLR)* 3, Feb (2003), 1137–1155.

[4] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Proceedings of 34th Advances in Neural Information Processing Systems (NeurIPS)*. MIT Press, 1877–1901.

[5] Michael Burstein and Mary N Youssef. 1985. Timing influenced layout design. In *Proceedings of 22nd Design Automation Conference (DAC)*. IEEE, ACM, 124–130.

[6] H Chang, Eugene Shragowitz, Jian Liu, Habib Youssef, Bing Lu, and Suphachai Sutanthavibul. 2002. Net criticality revisited: An effective method to improve timing in physical design. In *Proceedings of 11th International Symposium on Physical Design (ISPD)*. 155–160.

[7] 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 Transactions on Design Automation of Electronic Systems (TODAES)* 29, 1 (2023), 18:1–18:25.

[8] Amit Chowdhary, Karthik Rajagopal, Satish Venkatesan, Tung Cao, Vladimir Tiourin, Yegna Parasuram, and Bill Halpin. 2005. How accurately can we model timing in a placement engine?. In *Proceedings of 42nd Design Automation Conference (DAC)*. ACM, 801–806.

[9] Chris Chu. 2004. FLUTE: Fast lookup table based wirelength estimation technique. In *Proceedings of 23rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, ACM/IEEE, 696–701.

[10] Alfred E et al. Dunlop. 1988. Chip layout optimization using critical path weighting. In *Papers on Twenty-five years of electronic design automation*. 278–281.

[11] Hans Eisenmann and Frank M Johannes. 1998. Generic global placement and floorplanning. In *Proceedings of 35th Design Automation Conference (DAC)*. ACM, 269–274.

[12] Tong Gao, Pravin M Vaidya, and Chung Laung Liu. 1992. A Performance Driven Macro-Cell Placement Algorithm.. In *Proceedings of 29th Design Automation Conference (DAC)*. IEEE, 147–152.

[13] 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 59th Design Automation Conference (DAC)*. ACM, ACM, 1213–1218.

[14] Tsung-Wei Huang and Martin D. F. Wong. 2015. OpenTimer: A High-Performance Timing Analysis Tool. In *Proceedings of 34th IEEE/ACM International Conference on Computer-Aided Design (ICCAD)* (2015-11). IEEE, 895–902.

[15] Zhipeng Huang, Zengrong Huang, Simin Tao, et al. 2024. AiEDA: An Open-Source AI-Native EDA Library. In *2024 International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 794–795.

[16] Michael AB Jackson and Ernest S Kuh. 1989. Performance-driven placement of cell based IC's. In *Proceedings of 26th Design Automation Conference (DAC)*. ACM, 370–375.

[17] Andrew B Kahng, Jens Lienig, Igor L Markov, and Jin Hu. 2011. *VLSI physical design: from graph partitioning to timing closure*. Vol. 312. Springer.

[18] Xingquan Li, Zengrong Huang, Simin Tao, 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.

[19] Xingquan Li, Weiguo Li, Xinhua Lai, Junfeng Liu, and Rui Wang. 2026. iPCL: Pre-training for Chip Layout (invited). In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 1–7.

[20] Peiyu Liao, Dawei Guo, Zizheng Guo, Siting Liu, Yibo Lin, and Bei Yu. 2023. DREAMPlace 4.0: Timing-driven Placement with Momentum-based Net Weighting and Lagrangian-based Refinement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2023), 3374–3387.

[21] 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, 1–6.

[22] He Liu, Zhisheng Zeng, Simin Tao, et al. 2025. AiTPO: KAN-UNet Heterogeneous Network for Timing Prediction and Optimization at Global Routing. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 31, 3 (2025), 1–28.

[23] Lihao Liu, Beisi Lu, Yunhui Li, Li Shang, and Fan Yang. 2025. GTN-Path: Efficient Path Timing Prediction through Waveform Propagation with Graph Transformer. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.

[24] Wing K Luk. 1991. A fast physical constraint generator for timing driven layout. In *Proceedings of 28th Design Automation Conference (DAC)*. ACM, 626–631.

[25] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Proceedings of 27th Advances in Neural Information Processing Systems (NeurIPS)*. MIT Press, 3111–3119.

[26] Bernd Obermeier and Frank M Johannes. 2004. Quadratic placement using an improved timing model. In *Proceedings of 41st Design Automation Conference (DAC)*. ACM, 705–710.

[27] Siyu Qiu, Benjamin Tan, and Hammond Pearce. 2024. LLM-aided Explanations of EDA Synthesis Errors. In *Proceedings of 1st IEEE LLM Aided Design Workshop (LAD)* (2024-06). IEEE, 1–6.

[28] Yihang Qiu, Zengrong Huang, Simin Tao, et al. 2025. AiEDA: An Open-source AI-Aided Design Library for Design-to-Vector. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* (2025).

[29] William Swartz and Carl Sechen. 1995. Timing driven placement for large standard cell circuits. In *Proceedings of 32nd Design Automation Conference (DAC)*. ACM, 211–215.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2025. Attention is all you need. *Neural Computing and Applications* 37, 3 (2025), 1509–1522.

[31] Mingjun Wang, Bin Sun, Jianan Mu, et al. 2025. MOSS: Multi-Modal Representation Learning on Sequential Circuits. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–7.

[32] Ziyi Wang, Siting Liu, Yuan Pu, Song Chen, Tsung-Yi Ho, and Bei Yu. 2023. Restructure-Tolerant Timing Prediction via Multimodal Fusion. In *Proceedings of 60th Design Automation Conference (DAC)* (2023-07-09). IEEE, 1–6.

[33] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of 36th Advances in Neural Information Processing Systems (NeurIPS)*. MIT Press, 24824–24837.

[34] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. 2024. ChatEDA: A Large Language Model Powered Autonomous Agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 43, 10 (2024), 3184–3197.

[35] 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 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*. ACM, 671–677.