Prerouting Timing Prediction Across Different Technology Nodes

Xinyun Zhang, Binwu Zhu, Fangzhou Liu, Jiaxi Jiang, Ziyi Wang, Peng Xu, Hong Xu, *Senior Member, IEEE*, and Bei Yu, *Senior Member, IEEE*

***Abstract*—In the domain of very-large-scale integration (VLSI) design, the accuracy of prerouting timing prediction is of paramount importance for ensuring the performance and reliability of integrated circuits. Traditional methods based on machine learning necessitate the availability of extensive and high-quality datasets. However, this requirement poses significant challenges for advanced technology nodes due to the laborious and time-intensive nature of data preparation. To address this critical issue, we introduce a novel transfer learning framework that leverages data from preceding technology nodes to facilitate learning and prediction on the target node. Our methodology commences with the disentanglement and alignment of timing path features across different nodes, ensuring the preservation and effective translation of intrinsic timing path properties. Subsequently, we employ a Bayesian-based model to predict the arrival times of individual timing paths. This model is particularly adept at managing the high-variability inherent in arrival times and exhibits strong generalization capabilities to novel design scenarios. Moreover, we propose a new algorithm to reweight the preceding node data during training by estimating their transferability through the cell type distribution. We validate the efficacy of our proposed framework through comprehensive experimental evaluations, demonstrating successful transfer learning from 130 or 45 to 7-nm technology nodes. The results underscore the potential of our approach to significantly mitigate the dependency on extensive data preparation while maintaining high accuracy in timing prediction for cutting-edge VLSI designs.**

***Index Terms*—Pre-routing timing prediction, technology node, transfer learning.**

I. Introduction

THE DESIGN of contemporary integrated circuits (ICs) imposes stringent demands on timing constraints to ensure optimal performance and reliability. To adhere to these rigorous timing requirements, the placement and routing (PnR) processes are frequently conducted in an iterative manner, necessitating multiple iterations to meet the specified timing objectives. This iterative nature of PnR can be highly time consuming and resource intensive, often prolonging the design cycle significantly. As a consequence, researchers are actively developing methodologies to predict the timing report prior to the completion of the routing step. This "look-ahead" mechanism provides preliminary feedback that can be utilized for early stage timing optimization, potentially expediting the overall chip design process and reducing the number of required iterations.

The widely used timing prediction approach is the linear RC static timing analysis (STA) model, e.g., Elmore's model [1], which quickly evaluates timing using placement results. However, the efficacy of such models is compromised by the absence of detailed routing information, which is indispensable for precise timing analysis. Recent studies have achieved remarkable achievements in prerouting timing prediction by leveraging machine learning (ML) methodologies. For instance, Barboza et al. [2] introduced a method to predict local net/cell delay and slew using features derived from high-level placement results. Similarly, Guo et al. [3] proposed an end-to-end graph neural network (GNN) framework for predicting prerouting arrival time and slack values at critical timing endpoints. Additionally, Wang et al. [4] developed an endpoint embedding framework that integrates both netlist and layout information to enhance timing optimization.

Despite the considerable progress achieved through these ML-based approaches, a critical challenge persists: training a precise and highly generalizable ML-based timing model necessitates a substantial volume of training data at the target technology node. When trained on limited data, the performance of the timing predictor can deteriorate markedly, as illustrated in Fig. 1. This issue is exacerbated by the rapid advancement of technology nodes, where the collection of comprehensive timing data necessitates the execution of a series of computationally intensive tools, rendering the data collection process exceptionally time consuming and resource demanding. This presents a formidable challenge for training accurate timing predictors for advanced technology nodes. To address this issue, we propose a transfer learning framework to leverage extensive data from preceding technology nodes in conjunction with limited data from the target node. This approach aims to enhance the performance of timing prediction at the target node, as depicted in Fig. 1.

In the context of transfer learning for very-large-scale integration (VLSI) timing prediction, three primary challenges must be addressed to ensure efficacy and reliability.

Received 7 July 2024; revised 17 November 2024; accepted 22 December 2024. Date of publication 27 December 2024; date of current version 20 June 2025. This work was supported in part by The Research Grants Council of Hong Kong, SAR under Grant CUHK14208021, and in part by the MIND Project under Grant MINDXZ202404. This article was recommended by Associate Editor E. Keiter. *(Xinyun Zhang and Binwu Zhu contributed equally to this work.) (Corresponding author: Bei Yu.)*

Xinyun Zhang, Fangzhou Liu, Jiaxi Jiang, Ziyi Wang, Peng Xu, Hong Xu, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Binwu Zhu is with the School of Integrated Circuits, Southeast University, Nanjing 210000, China.

Digital Object Identifier 10.1109/TCAD.2024.3523426

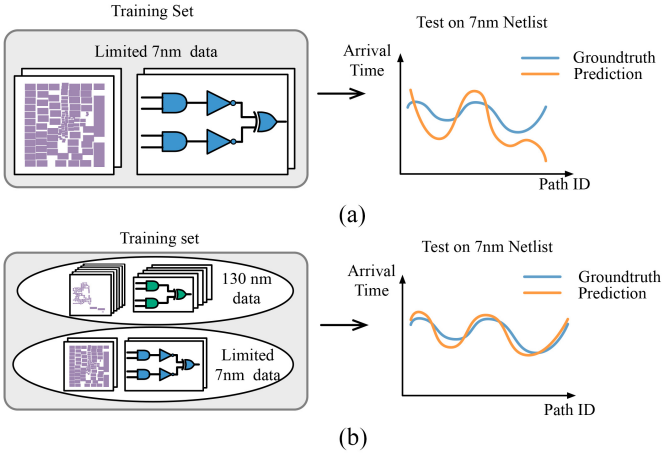


Fig. 1. (a) Trained on limited 7-nm netlist data. (b) Trained on both limited 7-nm netlist data and 130-nm netlist data.

First, transfer learning endeavors to exploit common and transferable knowledge across disparate data distributions. Netlist data intrinsically comprises two distinct types of knowledge: 1) node dependent and 2) design dependent. Node-dependent knowledge pertains to the characteristics of standard cell types, which can vary markedly across different technology nodes, including variations in cell libraries and electrical properties. Conversely, design-dependent knowledge encapsulates the intrinsic functionalities of timing paths, which remain consistent irrespective of the technology node. These two types of knowledge are intricately interwoven within the netlist graph, complicating the isolation and utilization of the transferable components across different nodes. The effective disentanglement of these intertwined features is paramount for successful transfer learning. Second, the arrival time values of different timing paths in VLSI circuits can exhibit substantial variability, sometimes differing by one or even two orders of magnitude. This significant variability poses a formidable challenge for ML-based regression models, which must accurately predict timing across a wide range of values. Third, the limited data availability at the target technology node exacerbates the difficulty of training a reliable timing predictor. With only a sparse dataset available for the target node, there is a heightened risk of overfitting the model to the training designs. Overfitting impairs the model's ability to generalize to new and unseen designs, thereby limiting its practical applicability. This scarcity of data necessitates innovative approaches to maximize the utility of the available information and enhance the model's generalization capabilities.

To mitigate these challenges, we propose a novel framework tailored for timing prediction that first disentangles the features of the timing path into node-dependent and design-dependent parts in the latent feature space, then, we align these two parts separately by minimizing a node-based contrastive loss and a design-based discrepancy loss. Subsequently, with the disentangled features, we propose a new Bayesian ML-based framework that can adapt to highly variable arrival time values and generalize well to new designs. We further provide a new algorithm to estimate data transferability by calculating the discrepancy between the distributions of the cell type. With this estimation, we reweight the data from source technology nodes, leading to better transfer learning efficacy.

To validate the effectiveness of our proposed method, we collect abundant data at the 130 and 45-nm node and limited data at the 7-nm node as our training set, then test its performance using the 7-nm data. The experimental results show that our method outperforms its counterparts by a significant margin. The main contribution can be summarized as follows.

1) To the best of our knowledge, we are the first to investigate transfer learning from different technology nodes in timing prediction.
2) A new feature disentanglement framework is proposed, which first decouples the timing path features into node-dependent and design-dependent parts and aligns them separately.
3) To enable effective generalization to new designs, we propose a novel Bayesian ML-based timing predictor.
4) We also propose a new algorithm to estimate the data transferability by calculating the optimal transport (OT) distance between the cell-type distributions of different technology nodes.
5) With the transferability estimation, we assign different importance to data from source technology nodes during training to further improve the transfer learning efficacy.
6) The experimental results on transfer learning from 130 or 45-nm node to 7-nm node verify the effectiveness of our method.

The remainder of this article is organized as follows. Section II lists some preliminaries about timing prediction and illustrates the problem formulation. Section III gives an overview of our prerouting timing prediction across different technology nodes. Sections IV–VI illustrate the details of each part, including the feature disentanglement and alignment (DA), Bayesian-based prediction method, and transferability estimation. Section VII presents our experimental results, followed by a conclusion in Section VIII.

II. Preliminaries

A. ML-Based Timing Prediction

Before the ML era, traditional methods [5], [6], [7] for timing prediction have been used for years. With the rapid development of ML, ML-based timing prediction techniques have been explored and proven to be successful. These approaches leverage large datasets and utilize ML algorithms to model the circuit timing. For routed wire timing estimation, Cheng et al. [8] adopted XGBoost [9] to predict the wire timing based on the tree and nontree RC networks. Later, Ye et al. [10] adopted a tailored GNN to model the complex paths in RC networks. Meanwhile, Kahng et al. [11] and Xing et al. [12] investigated timing prediction across different process-voltage-temperature (PVT) corners. For prerouting timing prediction, the missing routed wire information poses challenges for accurate timing estimation. Early attempts [2], [13] mainly leverage ML algorithms to model local timing delay with physical information from the placement stage. Barboza et al. [2] proposed a random

forest model that first utilizes extracted placement features to predict routed net delay and slew. Then, relying on PERT traversals [14], the overall circuit timing is determined based on the prediction results output by the random forest model. Following [2] and [13] incorporates the FLUTE algorithm [15] to construct a look-ahead RC network that provides a fast estimation for the routed wire information and significantly improves the prerouting timing estimation accuracy. Later, many end-to-end prerouting timing prediction methods [3], [4], [16], [17], [18] have been proposed. To further accelerate the timing prediction, Guo et al. [3] proposed a timing engine-inspired GNN model. By representing netlists as graphs, GNNs can capture the relationships and dependencies between circuit elements. The proposed GNN model predicts global timing metrics at timing endpoints in an end-to-end fashion, eliminating the need for additional feature engineering and invoking STA tools. Meanwhile, Cao et al. [16] incorporated transformers [19] to predict the timing delay for each path. Specifically, by modeling each timing path as a sequence, the attention mechanism in transformers can effectively capture the timing information encoded in various cells and netlist structures. Following [16] and [17] proposes a new automatic feature extractor that leverages a customized heterogeneous GNN as the encoder. Besides, He et al. [17] also considered timing optimization techniques, for example, gate sizing [20], [21], to further boost the performance. Song et al. [18] utilized multilayer perceptrons (MLPs) to encode the prerouting path delay and PVT features to compensate for the sequence-based features.

However, most previous ML-based timing predictors rely solely on the provided prerouting netlist structures, which do not align with the optimized sign-off structures. Consequently, these methods often yield inaccurate timing prediction results. In order to address this issue, Wang et al. [4] introduced a timing optimization-aware predictor capable of handling netlist restructuring. By recognizing that timing endpoints remain unchanged during timing optimization, the authors focus on global endpoint-level prediction instead of the previously utilized local cell-level prediction. They also propose incorporating supplementary information from the layout to model the impact of timing optimization. This study demonstrates that adopting a global endpoint-wise perspective from both the netlist and layout significantly enhances optimization-aware timing estimation. Following [4], our proposed timing model is also timing optimization-aware.

B. Graph Model in EDA

Since the circuit can be intuitively represented by a graph, where gates are depicted as nodes and wires as edges, the graph model is extensively utilized in various applications within the modern design process, offering significant simplification of problem formulation and algorithm analysis. Moreover, it effectively addresses numerous challenges present in typical EDA flow, such as technology mapping [22], [23], testability analysis [24], circuit partitioning [25], [26], placement [27], [28], and more. Besides, different graph algorithms can be employed based on specific application characteristics. For example, Bryant [29] presented a new algorithm for efficiently representing and manipulating Boolean functions using directed acyclic graphs in logic design verification. Constructing a power network with minimal wire length can be modeled as a minimum tree construction (MST) problem [30]. In global routing, Cong and Madden [31] developed a global router for standard cell design, which optimizes interconnect topologies and wire sizes to reduce critical path delays, employing either a channel graph model [31] or grid graph model [32]. In detailed routing, to represent the relative positions of different nets within a channel routing instance, horizontal and vertical constraint graphs are utilized in [33]. Other commonly used graph algorithms in EDA include network flow for placement [34], graph partitioning [35], graph coloring for layout decomposition [36], etc.

C. Multimodal Learning

Multimodal learning [37] is one kind of ML strategy, which offers an innovative pathway to understanding data complexity. It thrives on the very tenet of diverse data sources being interoperable, exploiting meaningful interactions between numerous modalities. Traditional ML approaches often fall short when dealing with multifarious data, typically defaulting to treating different modalities separately. Multimodal learning, alternatively, leverages several data types in an integrative fashion, ushering data analysis toward more comprehensive insight. A multimodal representation entails utilizing information from multiple such entities to represent the data. In this work, we leverage multimodal information, namely, the layout images and the netlist graph, for timing path representation.

D. Transfer Learning

In ML, when we do not have enough data on the target domain, an effective solution is transfer learning, which leverages data from other domains to aid the learning on the target domain. The critical issue in transfer learning is how to design a learning framework for learning the transferrable knowledge between the source and target domains. A widely used technique is pretraining-then-finetuning [38], which first trains the model on the source domain with abundant data to learn a good feature extractor and then finetunes the model with much fewer steps on the target domain to transfer the knowledge. Another effective approach is the parameter-sharing strategy [39] that lets some parameters of the neural networks be shared by data from different domains while the other parameters are learned separately. In EDA, Wang et al. [40] proposed a transfer learning-based framework for transistor sizing that transfers knowledge in different circuits. In the case of timing prediction, collecting the timing data requires running a long, time-consuming toolchain. Consequently, the ability to transfer knowledge from one technology node to another becomes increasingly important. To the best of our knowledge, the application of transfer learning in the field of timing prediction has not been previously discussed.

E. Bayesian-Based Machine Learning

Bayesian-based ML is a paradigm that integrates Bayesian inference with traditional ML techniques to enhance model robustness and interpretability. By representing uncertainty through probability distributions over model parameters, this approach allows for more nuanced and adaptive learning processes. Bayesian methods update beliefs in response to new data, providing a principled framework for incorporating prior knowledge and mitigating overfitting. As a key technique in Bayesian-based ML, Bayesian neural networks (BNNs) [41] integrate Bayesian inference with traditional neural network models to provide a probabilistic framework for learning. Unlike conventional networks that yield point estimates, BNNs maintain distributions over their weights, effectively capturing model uncertainty. This probabilistic approach enhances robustness to overfitting and allows for improved uncertainty quantification in predictions. By leveraging advanced techniques, such as variational inference [42], BNNs approximate the intractable posterior distributions. In this work, we propose a Bayesian-based timing prediction module for robust and adaptive timing delay regression.

F. Optimal Transport

OT [43] is a mathematical framework that determines the most efficient way to transform one probability distribution into another. It quantifies the cost of transporting mass between distributions, providing a powerful tool for comparing them. This framework has broad applications across fields, such as computer vision and ML, where it aids in tasks like domain adaptation, image processing, and generative modeling. By leveraging its ability to compare distributions in a geometrically meaningful manner, OT offers deep insights into data alignment and structural similarities. In this work, we leverage OT to measure the cell type distribution gap between two different netlists.

G. Central Moment Discrepancy

Central moment discrepancy (CMD) [44] quantifies the difference in central tendencies between two probability distributions. It is crucial in contexts where aligning these central characteristics enhances model performance and ensures consistency across datasets. CMD is particularly effective when the central alignment of distributions is prioritized, offering computational efficiency compared to more complex metrics.

H. Problem Formulation

Problem 1 (Transferrable Timing Prediction): Given a large netlist set $\mathcal{N}_S$ at the source preceding technology node and a limited netlist set $\mathcal{N}_T$ at the target advanced technology node, our goal is to learn a model which accurately predicts endpoint arrival time on test netlist data at advanced technology node, achieving high-R^2 score and demanding low-computation cost.

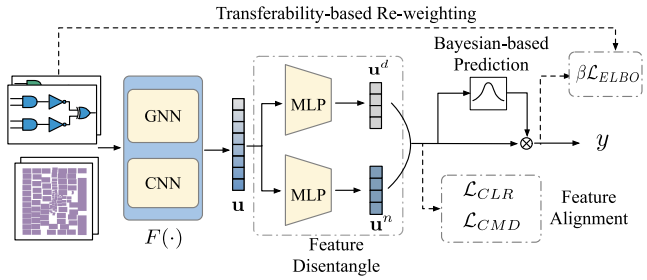


Fig. 2. Overview of the method. Timing path feature alignment and the reweighting are only computed during the training stage.

III. Overview

We build our timing prediction model in an endpoint-based manner. The main idea is first to disentangle the node-dependent and design-dependent features for each timing path. Then, we try to make the node-dependent features consistent in one technology node and distinguishable in different nodes. Meanwhile, since the design-dependent features are node-agnostic, we try to minimize the gap between the design-dependent features in different nodes. These two objectives serve as the target of feature alignment. Later, we propose a Bayesian-based timing predictor to output the highly variable timing delay for each endpoint. Then, we estimate the transferability for each timing path in source technology nodes by calculating the cell type distribution discrepancy. With the estimated transferability, we reweight the samples in the source node to improve the transfer learning efficacy. The overview of our method is shown in Fig. 2. We detail our methods in the following sections.

IV. Feature Disentanglement and Alignment

The feature DA process consists of three parts: 1) feature extraction; 2) node-based contrastive learning; and 3) design-based discrepancy minimization.

A. Timing Path Feature Extractor

Following [4], we build a multimodal timing path feature extractor $F(\cdot)$, that is, capable of handling netlist restructuring, as shown in Fig. 3. The netlist is first transformed into a heterogeneous graph $\mathcal{G}$, where each pin is treated as a node, and each net edge and cell edge is represented as an edge in the graph. Besides, the output of cell edges is referred as cell nodes, and the sink of net edges is regarded as net nodes. We denote the set of all the timing paths as $\text{Path}(\mathcal{G}) = \{\mathcal{G}'_i\}_{i=1}^{M}$, where $\mathcal{G}'$ represents a timing path and M is the total number of timing paths. Here, $\mathcal{G}'$ is the whole fanin cone for the timing endpoint, a subgraph of the netlist $\mathcal{G}$. Each timing path graph $\mathcal{G}'$ consists of pin and net information. Besides, we also collect the corresponding layout image set $\mathcal{X}$ for each timing path.

The heterogeneous graph $\mathcal{G}'$ has two types of edge, the net edge and the cell edge. The nodes in $\mathcal{G}'$ represent the pins in the netlist. The net distance, cell driving strength, gate type, and pin capacitance are used as the node features. Note that we use one-hot representation for the gate type and merge all the gates in different technology nodes as the total gate set.

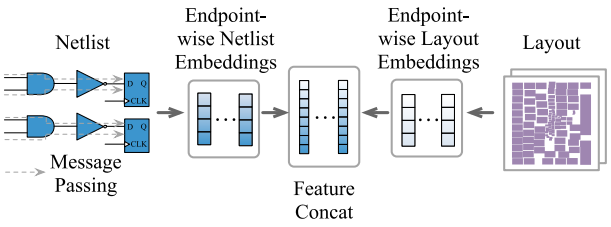


Fig. 3. Timing path feature extractor.

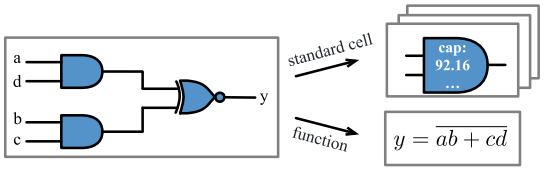


Fig. 4. Netlist can be characterized by two parts of information: The functionality and the standard cell information.

Then, we use a GNN to propagate on $\mathcal{G}'$ from the primary inputs to the endpoints to obtain the feature for each timing path. The GNN model propagates information in topological order, eventually gathering at the timing endpoints. The final netlist embeddings for the endpoint e is denoted as $\vec{u}_n^e$.

The layout image set $\mathcal{X}$ includes the cell density map, rectangular uniform wire density map, and the macro cells region map. Considering that the impact of timing optimization varies for different endpoints, an endpoint-wise masking technique is proposed, which consists of two steps: 1) path-finding and 2) mask generation. The path-finding is to search the longest path P_e for the endpoint e, which can be solved by depth-first-search (DFS). After obtaining the longest path P_e for each endpoint e, the critical region $\mathbf{M}^e$ of endpoint e will be constructed, which is achieved by taking the union region covered by the bounding boxes of the two-pin net edges along P_e. Then, with the region mask $\mathbf{M}^e$ and the layout information map $\mathbf{M}^L$, the endpoint-wise layout embedding $\mathbf{u}_l^e$ can be calculated as follows:

$$\mathbf{u}_l^e = \text{CNN}(\mathbf{M}^e \odot \mathbf{M}^L) \tag{1}$$

where CNN$(\cdot)$ denotes a convolution neural network. The final output of the multimodal feature extractor $\vec{u}$ is the concatenation of $\vec{u}_n^e$ and $\vec{u}_l^e$.

B. Timing Path Feature Disentanglement

As shown in Fig. 4, each netlist contains two parts of information: 1) the functionality information encoded in the design specification and 2) the standard cell information, including its structures and parameters, such as the load and capacitance. For a given design, mapping to different technology nodes may produce two utterly different netlist graphs, but they will share the same logical functionality. Inspired by this, we aim to separate these two kinds of knowledge to facilitate transfer learning. However, these two pieces of knowledge are highly coupled in the netlist graphs, which is infeasible to separate directly from the raw input. Therefore, we propose disentangling the design-dependent knowledge and the node-dependent knowledge from the feature space of the timing path.

Since we aim to learn a performant timing predictor from only limited data in the target advanced technology node and abundant data in the source preceding technology node, we may assume that our training set is composed of two parts $\mathcal{N}_S = \{\mathcal{G}_1^S, \ldots, \mathcal{G}_{L_S}^S\}$ and $\mathcal{N}_T = \{\mathcal{G}_1^T, \ldots, \mathcal{G}_{L_T}^T\}$, where $\mathcal{N}_S$ and $\mathcal{N}_T$ represent the source preceding node and target advanced technology node netlist set, respectively, and we have $L_S \gg L_T$. For any path feature $\vec{u}$, we further adopt two multilayer perceptrons (MLP$(\cdot)$) to disentangle the equal-sized node-dependent features $\vec{u}^n$ and design-dependent features $\vec{u}^d$ by

$$\vec{u}^n = \text{MLP}_n(\vec{u}) \in \mathbb{R}^{m/2}, \vec{u}^d = \text{MLP}_d(\vec{u}) \in \mathbb{R}^{m/2}. \tag{2}$$

Each layer of MLP is parameterized by two learnable weights $\vec{W}$ and $\vec{b}$, which can be characterized by

$$\phi(\vec{u}) = \text{ReLU}(\vec{W}\vec{u} + \vec{b}). \tag{3}$$

Then, the total MLP function can be defined as

$$\text{MLP}(\cdot) = (\phi_L \circ \cdots \circ \phi_1)(\cdot) \tag{4}$$

where L is the number of layers. To save the number of parameters, we set L to 2. Moreover, for MLP$_d$, we append one extra tanh$(\cdot)$ activation after the MLP to limit the range of the design-dependent feature, which will be used for feature alignment detailed in the following section.

C. Timing Path Feature Alignment

Following feature disentanglement, the target of feature alignment is to design a proper loss function to encode our designated disentanglement of the node- and design-dependent knowledge into an end-to-end ML framework. With this in mind, we propose two loss functions, node-based contrastive loss and design-based discrepancy loss, to align the node- and design-dependent features, respectively. The overview of timing path feature alignment is shown in Fig. 5.

Node-Based Contrastive Loss: The intuition to align the node-dependent features is that the netlist on the same node should share the same standard cells, including the gate structures and the configuration parameters for pins and nets. On the contrary, the node-dependent features should be distinguishable for netlists at different nodes. For this purpose, we propose a node-based contrastive loss.

Specifically, in each batch of training data, we first sample some designs $\mathcal{N}_S' \subseteq \mathcal{N}_S$ and $\mathcal{N}_T' \subseteq \mathcal{N}_T$. Then, we sample some paths from both nodes to construct the batch-wise training data. We denote the paths from the source node as $\mathcal{B}_S = \{\mathcal{G}_i'^S\}_{i=1}^P$ and the paths from the target node as $\mathcal{B}_T = \{\mathcal{G}_i'^T\}_{i=1}^P$, where P is the batch size for each node. Each batch of data consists of data from both the source and target nodes, which can be defined as

$$\mathcal{B} = \mathcal{B}_S + \mathcal{B}_T. \tag{5}$$

Then, we can obtain the disentangled path feature sets, i.e., node- and design-dependent features, for the source preceding

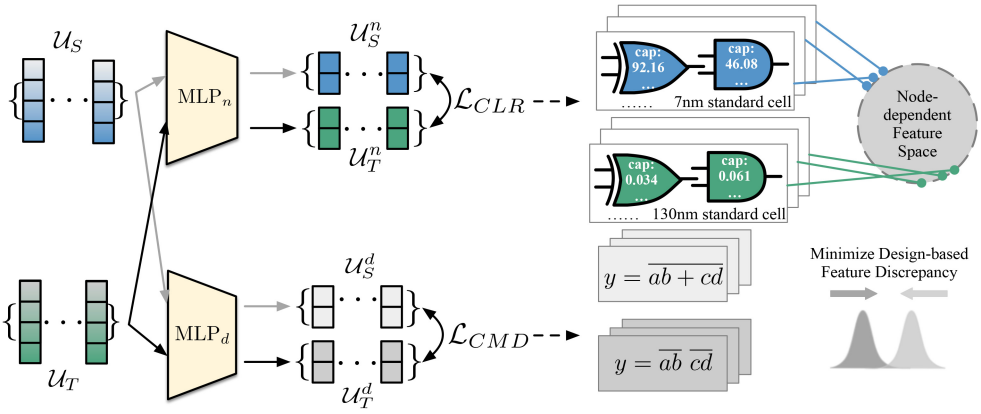


Fig. 5. Timing path feature alignment.

node, denoted as $\mathcal{U}_S^n$ and $\mathcal{U}_S^d$. The feature extraction process for the source technology node can be formulated as

$$\mathcal{U}_S^n = \{\text{MLP}_n(F(\mathcal{G}'))|\mathcal{G}' \in \mathcal{B}_S\} \tag{6}$$

and

$$\mathcal{U}_S^d = \{\text{MLP}_d(F(\mathcal{G}'))|\mathcal{G}' \in \mathcal{B}_S\}. \tag{7}$$

Similarly, we can obtain the node-dependent path features and design-dependent path features of the target advanced node, denoted as $\mathcal{U}_T^n$ and $\mathcal{U}_T^d$. The feature extraction process can be characterized by

$$\mathcal{U}_T^n = \{\text{MLP}_n(F(\mathcal{G}'))|\mathcal{G}' \in \mathcal{B}_T\} \tag{8}$$

and

$$\mathcal{U}_T^d = \{\text{MLP}_d(F(\mathcal{G}'))|\mathcal{G}' \in \mathcal{B}_T\}. \tag{9}$$

Denote the set of all the node-dependent features as $\mathcal{A} = \mathcal{U}_S^n \cup \mathcal{U}_T^n$. Given any node-dependent feature set $\mathcal{U}^n$ ($\mathcal{U}_S^n$ or $\mathcal{U}_T^n$), the contrastive loss for the feature set can be defined as

$$\mathcal{L}_{\text{Set}}(\mathcal{U}^n) = \sum_{\vec{u}\in\mathcal{U}^n} \frac{-1}{|\mathcal{U}^n| - 1} \sum_{\vec{m}\in\mathcal{U}^n\setminus\{\vec{u}\}} \frac{\exp(\vec{u}\cdot\vec{m}/\tau)}{\sum_{\vec{a}\in\mathcal{A}\setminus\{\vec{u}\}}\exp(\vec{u}\cdot\vec{a}/\tau)} \tag{10}$$

and the total contrastive loss can be formulated as

$$\mathcal{L}_{CLR} = \frac{1}{|\mathcal{U}_S^n|}\mathcal{L}_{\text{Set}}(\mathcal{U}_S^n) + \frac{1}{|\mathcal{U}_T^n|}\mathcal{L}_{\text{Set}}(\mathcal{U}_T^n). \tag{11}$$

Node-based contrastive loss aims to pull together the features from the same node while pushing apart those from different nodes. With this loss, the node-dependent features on the same node will be approximately consistent and differ in different nodes.

Design-Based Discrepancy Loss: The design-dependent features represent the abstract logical functionality of each netlist. For each design, we can opt for different technology nodes to synthesize various netlists but with the same functionality. Therefore, the overall distribution of the design-dependent features in different nodes should be consistent.

To align the design-dependent features, we optimize the CMD between the feature sets from different nodes, which can be formulated as

$$\mathcal{L}_{CMD}(\mathcal{U}_S^d, \mathcal{U}_T^d) = \frac{1}{b-a}\left\|\mathbb{E}(\mathcal{U}_S^d) - \mathbb{E}(\mathcal{U}_T^d)\right\| + \sum_{k=2}^{\infty}\frac{1}{|b-a|^k}\left\|c_k(\mathcal{U}_S^d) - c_k(\mathcal{U}_T^d)\right\| \tag{12}$$

where $[a, b]$ is the interval that bounds $\mathcal{U}_S^d$ and $\mathcal{U}_T^d$, $\mathbb{E}(\cdot)$ denotes the expectation, and $c_k(\cdot)$ is the kth order moment. Since we apply a tanh activation function after MLP$_d$, we limit the value of the node-dependent features in $(-1, 1)$. Therefore, we can set a and b to -1 and 1, respectively. In practice, we set the maximum moment order to 5. In essence, minimizing (12) equals minimizing the gap between the statistics in different orders of the design-dependent features from different nodes. According to [44], we have the following properties.

Theorem 1: Let $\mathcal{U}_S^d$ and $\mathcal{U}_T^d$ be two probability distributions on a compact interval, then we have

$$CMD(\mathcal{U}_S^d, \mathcal{U}_T^d) \to 0 \implies \mathcal{U}_S^d \to \mathcal{U}_T^d. \tag{13}$$

Such a property can guarantee that minimizing the loss function formulated in (12) can align the overall distribution of the design-dependent features from netlists at different technology nodes.

V. Bayesian-Based Timing Prediction

Given a timing path feature $\vec{u}$, the conventional approach is to feed the timing path feature $\vec{u}$ into a linear layer with weight $\vec{W} \in \mathbb{R}^{1\times m}$ to output the arrival time. However, the value of the arrival time for different endpoints can be significantly different even in one netlist, as shown in Fig. 6, which poses a huge challenge for accurate timing prediction with only a fixed $\vec{W}$. Besides, since we only have very limited data on our target technology node, the ML-based model is prone to overfitting the training design. This can dramatically limit the timing prediction performance on the unknown test netlists, which may possess a large distribution gap with the training set as shown in Fig. 6. To address these issues, we propose a Bayesian-based timing prediction model, as shown in Fig. 7.

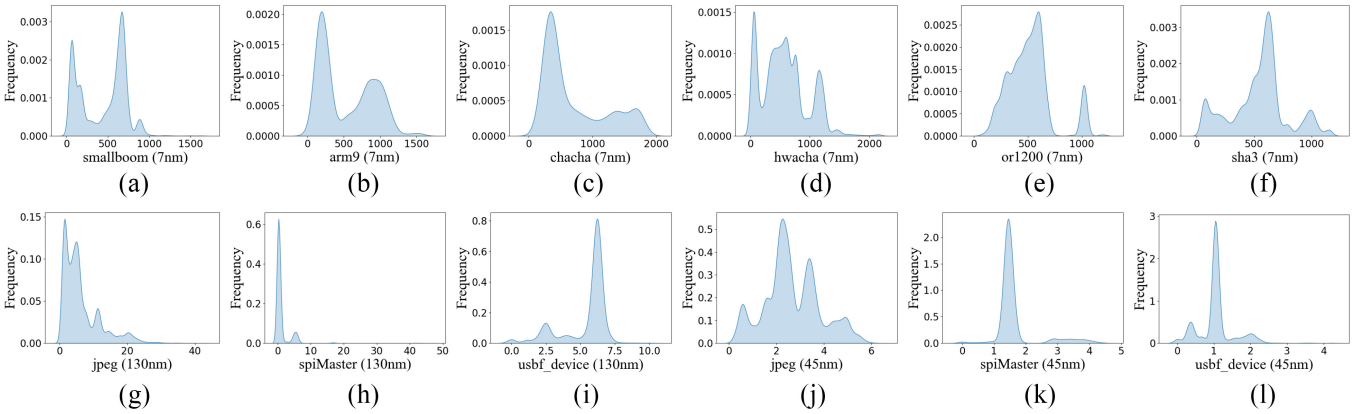


Fig. 6. Kernel density estimation of the different designs' arrival time distribution. (a) smallboom (7nm); (b) arm9 (7nm); (c) chacha (7nm); (d) hwacha (7nm); (e) or1200 (7nm); (f) sha3 (7nm); (g) jpeg (130nm); (h) spiMaster (130nm); (i) usbf_device (130nm); (j) jpeg (45nm); (k) spiMaster (45nm); (l) usbf_device (45nm).

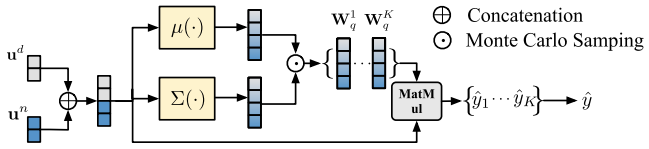


Fig. 7. Bayesian ML-based timing prediction.

Under the Bayesian ML framework, the model parameters are considered as a distribution instead of fixed parameters. This feature allows us to condition the model parameters on any inputs, allowing the model to adapt to different inputs easily. In our timing prediction task, we model the final readout linear layer $\vec{W}$ as a distribution to obtain better flexibility. Moreover, an ideal and well-generalizable timing predictor should make predictions based on the input timing path feature and the distribution of all the timing paths on the target node. However, common ML-based timing prediction methods [3], [4] only consider the input timing path while ignoring the global distributions.

Therefore, following common practice in Bayesian ML [45], [46], we can formulate our learning objective as:

$$\log p(y|\mathcal{G}', \mathcal{N}) = \log \int p(y|\mathcal{G}', \vec{W}) p(\vec{W}|\mathcal{N}) d\vec{W} \tag{14}$$

where y denotes the ground truth arrival time, $\mathcal{N}$ represents the overall distribution for all the timing paths at the technology node, and we condition $\vec{W}$ on this true global timing path distribution with a probability density function $p(\vec{W}|\mathcal{N})$, also known as the prior distribution. However, computing this probability in real practice is infeasible since the real timing path distribution of the whole target node $\mathcal{N}$ is intractable when we only have limited timing paths from the netlists on the target advanced technology node. Therefore, we adopt variational inference [45], [46], [47] to approximate the prior distribution.

Specifically, we introduce a variational posterior distribution $q(\vec{W}|\mathcal{G}')$ which only conditions on single timing path input and is easy to compute to simulate the prior distribution. Then, we can derive the evidence lower bound (ELBO) of the objective function by

$$\begin{aligned}
\log p(y|\mathcal{G}', \mathcal{N}) &= \log \int p(y|\mathcal{G}', \vec{W}) p(\vec{W}|\mathcal{N}) d\vec{W} \\
&= \log \int p(y|\mathcal{G}', \vec{W}) \frac{p(\vec{W}|\mathcal{N})}{q(\vec{W}|\mathcal{G}')} q(\vec{W}|\mathcal{G}') d\vec{W} \\
&= \log \mathbb{E}_q \left[p(y|\mathcal{G}', \vec{W}) \frac{p(\vec{W}|\mathcal{N}))}{q(\vec{W}|\mathcal{G}')} \right] \\
&\geq \mathbb{E}_q \left[\log p(y|\mathcal{G}', \vec{W}) \frac{p(\vec{W}|\mathcal{N}))}{q(\vec{W}|\mathcal{G}')} \right] \\
&= \mathbb{E}_q \big[\log p(y|\mathcal{G}', \vec{W}) \big] \\
&\quad - \mathrm{KL}(q(\vec{W}|\mathcal{G}')||p(\vec{W}|\mathcal{N}))).
\end{aligned} \tag{15}$$

The first term in (15) is the log-likelihood with the variational posterior. Minimizing the first term equals narrowing down the gap between the timing prediction conditioned on the input path features and the ground truth, which is the same as previous ML-based timing prediciton methods [3], [4]. The second term in (15) is the KL divergence between the variational posterior and the prior distribution, which regularizes the prediction conditioned on single paths and the prior knowledge from the global distribution. This regularizer can help alleviate the overfitting issue and benefit the model's generalization ability [45]. In all, with (15), we can learn a network with parameters only conditioned on the input timing path $\mathcal{G}'$ but with good generalization ability. Besides, this design also fits the feature of high variability of the arrival time of different timing paths.

Now, we introduce the construction of p and q for optimization. Following common practice in variational inference [45], [47], we can model the variational posterior distribution as a Gaussian formulated by

$$\vec{W}_q \sim N(\mu([\vec{u}^n, \vec{u}^d]), \Sigma([\vec{u}^n, \vec{u}^d])) \tag{16}$$

where $\vec{W}_q$ is the linear layer generated by distribution q, $\vec{u}^n$ and $\vec{u}^d$ are the disentangled node-dependent and design-dependent features for the timing path $\mathcal{G}'$, and $\mu(\cdot)$ and $\Sigma(\cdot)$ are two MLPs. In other words, we use the two learnable small networks, $\mu(\cdot)$ and $\Sigma(\cdot)$, to output the mean and variance of the distribution of $\vec{W}_q$, as shown in Fig. 7. Similarly, using the amortization trick [47], we can model the prior as

$$\vec{W}_p \sim N(\mu(\tilde{\vec{u}}(\mathcal{N})), \Sigma(\tilde{\vec{u}}(\mathcal{N}))) \tag{17}$$

where $\vec{W}_p$ is the linear layer generated by the prior distribution p and $\tilde{\vec{u}}(\mathcal{N}) \in \mathbb{R}^m$ is a dummy timing path feature that represents the true distribution of all the timing paths within the whole technology node $\mathcal{N}$. To construct a representative $\tilde{\vec{u}}$ for the target technology node with only limited data, we leverage the disentangled node- and design-dependent features. Specifically, since the node-dependent features are approximately consistent within one technology node, we can simply use the mean of all the node-dependent features to represent the node information. As for the design-related information, we can collect all the design-dependent features in both the source preceding technology node and the target advanced technology node since the design-based discrepancy loss has already brought them to the same distribution, and we can take the mean of them as the representative feature for all the designs. Since we adopt batch-wise stochastic gradient descent (SGD) for optimization, we update $\tilde{\vec{u}}$ via moving average. Specifically, for each batch of features, we first calculate the batch means of the node-dependant features of the source technology node as follows:

$$\vec{u}'^n_S = \frac{1}{|\mathcal{U}^n_S|} \sum_{\vec{u} \in \mathcal{U}^n_S} \vec{u}. \tag{18}$$

Similarly, the batch mean of the node-dependant features of the target technology node can be calculated by

$$\vec{u}'^n_T = \frac{1}{|\mathcal{U}^n_T|} \sum_{\vec{u} \in \mathcal{U}^n_T} \vec{u}. \tag{19}$$

For the design dependent features, we calculate the overall batch mean from both the source and target nodes, which can be formulated as follows:

$$\vec{u}'^d = \frac{1}{|\mathcal{U}^d_T| + |\mathcal{U}^d_S|} \sum_{\vec{u} \in \mathcal{U}^d_T \cup \mathcal{U}^d_T} \vec{u}. \tag{20}$$

Then, the total batch mean for the source target node can be constructed as

$$\vec{u}'_S = \left[\vec{u}'^n_S, \vec{u}'^d\right] \tag{21}$$

and the batch mean for the target node can be constructed as

$$\vec{u}'_T = \left[\vec{u}'^n_T, \vec{u}'^d\right] \tag{22}$$

where $[\cdot]$ denotes feature concatenation. With these batch means, the moving average update of the dummy feature $\tilde{\vec{u}}$ can be denoted as

$$\tilde{\vec{u}}(\mathcal{N}_S) = (1 - \mathrm{m}) \cdot \tilde{\vec{u}}(\mathcal{N}_S) + \mathrm{m} \cdot \vec{u}'_S \tag{23}$$

and

$$\tilde{\vec{u}}(\mathcal{N}_T) = (1 - \mathrm{m}) \cdot \tilde{\vec{u}}(\mathcal{N}_T) + \mathrm{m} \cdot \vec{u}'_T \tag{24}$$

where m is the momentum factor. We set m to 0.001 in our experiment, and $\tilde{\vec{u}}$ is initialized to the first batch mean.

Combining (16) and (17) and the construction of $\tilde{\vec{u}}$, we can explicitly calculate $\mathrm{KL}(q(\vec{W}|\mathcal{G}')||p(\vec{W}|\mathcal{N})))$. To calculate the first term in (15), we use Monte Carlo sampling to sample K samples from $q(\vec{W}|\mathcal{G}')$, denoted as $\{\vec{W}^i_q\}^K_{i=1}$, and with each sampled $\vec{W}^i_q$, we can obtain a timing prediction $\hat{y}_i$, which are then averaged to obtain the final timing prediction $\hat{y} = (1/K)\sum^K_1 \hat{y}_i$. The ELBO objective can be formulated as

$$\begin{aligned} &\mathcal{L}_{ELBO}(y, \mathcal{G}', \mathcal{N}) \\ &= \frac{1}{K}\sum_{i=1}^{K} \log p(y|\mathcal{G}', \vec{W}^i_q) - \mathrm{KL}(q(\vec{W}|\mathcal{G}')||p(\vec{W}|\mathcal{N})). \end{aligned}$$

Overall, the first term in (25) represents the timing prediction loss, and the second term describes the distribution discrepancy between the variational posterior and prior distribution.

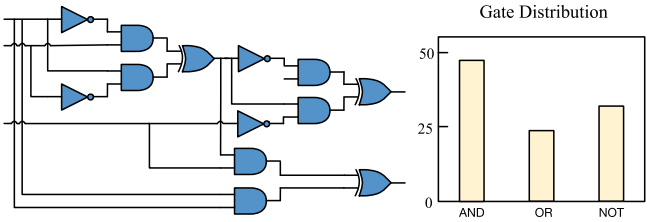


Fig. 8. Example of cell type distribution construction.

VI. Transferability-Based Reweighting

The efficacy of transfer learning is not universally guaranteed. When the source and target domains lack sufficient similarity, transferring knowledge from a weakly related source can adversely impact performance in the target domain, a phenomenon commonly referred to as negative transfer [48]. Therefore, we propose a standard cell similarity-aware transfer learning that reweights the samples from previous technology nodes by estimating their transferability. Specifically, we first construct the standard cell type distribution for each timing path. Then, we estimate the transferability by calculating the discrepancy between the data from the source nodes and that from the target nodes using OT distance. Finally, we reweight the source data based on the estimation. Note that other factors, such as the netlist graph structure, may affect the similarity between the source and target data. However, these are hard to quantify and calculate. We adopt cell type distribution as it is an important and easily calculable indicator to measure the similarity between the source and target data.

Standard Cell Type Distribution Construction: For each timing path, we first calculate the distribution of different types of cells. Specifically, during GNN propagating from the primary inputs to the timing endpoints, we count the frequency for each cell type and normalize them to construct the cell type distribution, as shown in Fig. 8.

Transferability Estimation: The key idea for transferability estimation is to measure the cell type distribution discrepancy. Considering that the support of the cell type distribution of different nodes may be different due to the different cell libraries, we leverage OT to calculate the discrepancy. OT distance is a commonly utilized metric for comparing distributions that may have different support. In this work, we only focus on the discrete case because of the nature of the cell type distribution. Suppose we have two sets of features;

the discrete distributions can be represented as follows:

$$k = \sum_{i=1}^{M} a_i \delta_{\vec{f}_i} \text{ and } l = \sum_{j=1}^{N} b_j \delta_{\vec{g}_j} \tag{25}$$

where k and l are two M and N-dimensional discrete probability distributions, respectively. $\vec{a}$ and $\vec{b}$ are discrete probability vectors summing to 1, and δ is a Dirac delta function. In our case, $\vec{f}$ and $\vec{g}$ are the representations of cell types from different technology nodes, and $\vec{a}$ and $\vec{b}$ denote the probability density. The transport distance is then expressed as

$$\langle \vec{T}, \vec{C} \rangle = \sum_{i=1}^{M} \sum_{j=1}^{N} \vec{T}_{m,n} \vec{C}_{m,n} \tag{26}$$

where $\vec{C}$ represents the cost between points $\vec{f}_i$ and $\vec{g}_j$, and $\vec{T}$ represents the transport plan. Given a fixed cost matrix $\vec{C}$, different plan $\vec{T}$ will lead to different transport distances. The OT distance is defined as the minimal transport distance, which can be defined as solving the following optimization problem:

$$\begin{aligned} \text{OT}(k, l) &= \min_{\vec{T}} \langle \vec{T}, \vec{C} \rangle \\ s.t. \quad & \vec{T}\vec{1}_N = \vec{a}, \vec{T}^\top \vec{1}_M = \vec{b}, \vec{T} \in \mathbb{R}_+^{M \times N} \end{aligned} \tag{27}$$

where $\text{OT}(\cdot, \cdot)$ denotes the optimal distance. Since directly optimizing this objective is often computationally expensive, we apply the Sinkhorn distance [49] which incorporates an entropic constraint for faster optimization. The optimization problem with an entropic regularization term is defined as

$$\begin{aligned} \text{OT}_\gamma(k, l) &= \min_{\vec{T}} \langle \vec{T}, \vec{C} \rangle - \gamma h(\vec{T}) \\ s.t. \quad & \vec{T}\vec{1}_N = \vec{a}, \vec{T}^\top \vec{1}_M = \vec{b}, \vec{T} \in \mathbb{R}_+^{M \times N} \end{aligned} \tag{28}$$

where $h(\cdot)$ denotes the entropy function and $\gamma \geq 0$ is a regularization parameter. This leads to a more efficient solution through iterative updates

$$\vec{T}^* = \text{diag}\left(\vec{a}^{(t)}\right) \exp\left(\frac{-\vec{C}}{\gamma}\right) \text{diag}\left(\vec{b}^{(t)}\right) \tag{29}$$

where t represents the iteration step. At each iteration, the updates are

$$\vec{a}^{(t)} = \frac{\vec{a}}{\left(\exp\left(\frac{-\vec{C}}{\gamma}\right)\vec{b}^{(t-1)}\right)} \tag{30}$$

and

$$\vec{b}^{(t)} = \frac{\vec{b}}{\left(\exp\left(\frac{-\vec{C}}{\gamma}\right)^\top \vec{a}^{(t)}\right)} \tag{31}$$

starting with the initialization $\vec{b}^{(0)} = 1$.

The OT distance is conditioned on the definition of the cost matrix $\vec{C}$. In the context of the cell type distribution, we split the cell libraries for each node into two parts: 1) the cells with common logical functions in both nodes denoted as $\mathcal{Q}$ and 2) the cells with specific logical function on the given node. For example, there may be cells with INV_1x1 functionality in both nodes. Suppose the source and target nodes have M and N cells, respectively. We denote $\{\vec{f}_i\}_{i=1}^M$ and $\{\vec{g}_i\}_{i=1}^N$ as the cell representation for the source and target nodes, respectively. By further letting logical$(\cdot)$ represent the logical function of a given cell, we can define the cost matrix as

$$\vec{C}_{i,j} = \begin{cases} 0.1, & \text{if logical}(\vec{f}_i) = \text{logical}(\vec{g}_j) \\ 0.5, & \text{if logical}(\vec{f}_i) \neq \text{logical}(\vec{g}_j) \text{ and } \vec{f}_i, \vec{g}_j \in \mathcal{Q} \\ 1, & \text{o/w.} \end{cases} \tag{32}$$

The intuition behind (32) is that we assign small cost to cells with identical logical functions in both nodes. We assign larger weights for cells with different logical functions but appearing in the common cell set $\mathcal{Q}$. For cells not in $\mathcal{Q}$, which means they only appear in one node, we assign the highest cost.

Sample Reweighting: We reassign different weights to samples in the source technology node based on the transferability estimation, as illustrated above. Specifically, for each batch, we first calculate the average cell type distribution for the target data, denoted as k^T. Then, for each timing path in the source node, we calculate the OT distance between its cell distribution l_i^S and k^T, as formulated by

$$\alpha_i = \text{OT}_\gamma(k^T, l_i^S) \ \forall i \in \{1, \ldots, |\mathcal{B}_S|\}. \tag{33}$$

Considering the construction of the cost matrix, we can see that $0.1 \leq \alpha_i \leq 1$. Therefore, we define the weight as

$$\beta_i = 1 - \alpha_i \ \forall i \in \{1, \ldots, |\mathcal{B}_S|\}. \tag{34}$$

TABLE I
STATISTICS OF THE DATASET (EDP STANDS FOR ENDPOINT, AND e_n AND e_c DENOTE NET EDGE AND CELL EDGE, RESPECTIVELY)

Benchmark		Input information				
		tech node	#pin	#edp	#e_n	#e_c
train	smallboom	$7nm$	694441	61764	488052	423344
	jpeg	$130nm$	1527166	39783	1150173	749294
	linkruncca	$130nm$	186546	17796	151617	84393
	spiMaster	$130nm$	99507	4739	75718	44917
	usbf_device	$130nm$	48104	4777	37557	21706
	jpeg	$45nm$	606729	39801	400746	440820
	linkruncca	$45nm$	118687	17916	86043	67125
	spiMaster	$45nm$	71616	4752	51853	41142
	usbf_device	$45nm$	48104	4777	37557	21706
test	arm9	$7nm$	44469	2500	33065	29287
	chacha	$7nm$	35687	1986	25117	23083
	hwacha	$7nm$	1357798	61313	985057	922085
	or1200	$7nm$	1165114	172401	844443	658961
	sha3	$7nm$	794720	60323	552021	485596
Avg	train	$7nm$&$130nm$	511153	25772	380623	264731
	test	$7nm$	679558	59705	487941	423802

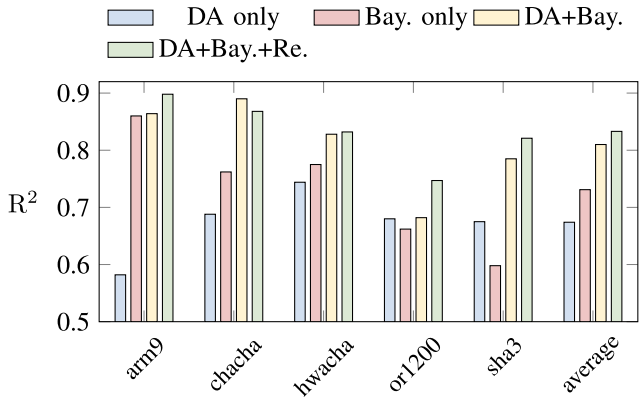


Fig. 9. Ablation study on the effectiveness of each module.

TABLE II
EVALUATION RESULTS ON TRANSFER LEARNING IN THE 130-TO-7-NM SETTING. HERE, RUNTIME DENOTES THE MODEL INFERENCE TIME

Methods	DAC23 [4]-AdvOnly		DAC23 [4]-SimpleMerge		DAC23 [4]-ParamShare [39]		DAC23 [4]-PT-FT [38]		DAC24 [50]		Ours	
	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime
arm9	0.603	2.546	-2.069	2.546	0.567	2.546	0.837	2.546	0.864	2.621	**0.898**	2.621
chacha	0.624	1.188	-1.983	1.188	0.568	1.188	0.726	1.188	**0.890**	1.234	0.868	1.234
hwacha	0.170	5.229	-2.203	5.229	0.499	5.229	0.818	5.229	0.828	5.400	**0.832**	5.400
or1200	0.156	14.257	-6.037	14.257	0.240	14.257	0.209	14.257	0.682	14.793	**0.747**	14.793
sha3	0.425	1.690	-4.741	1.690	0.195	14.257	0.284	1.690	0.785	1.725	**0.821**	1.725
average	0.396	4.982	-3.407	4.982	0.414	4.982	0.575	4.982	0.810	5.154	**0.833**	5.154

TABLE III
EVALUATION RESULTS ON TRANSFER LEARNING IN THE 45-TO-7-NM SETTING. HERE, RUNTIME DENOTES THE MODEL INFERENCE TIME

Methods	DAC23 [4]-AdvOnly		DAC23 [4]-SimpleMerge		DAC23 [4]-ParamShare [39]		DAC23 [4]-PT-FT [38]		DAC24 [50]		Ours	
	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime	R^2 score	runtime
arm9	0.603	2.546	-1.774	2.546	0.530	2.546	0.854	2.546	0.857	2.621	**0.862**	2.621
chacha	0.624	1.188	-1.364	1.188	0.619	1.188	0.656	1.188	0.838	1.234	**0.883**	1.234
hwacha	0.170	5.229	-1.335	5.229	0.549	5.229	0.699	5.229	0.820	5.400	**0.863**	5.400
or1200	0.156	14.257	-4.132	14.257	0.297	14.257	0.606	14.257	**0.782**	14.793	0.775	14.793
sha3	0.425	1.690	-2.709	1.690	0.369	1.690	-0.056	1.690	0.766	1.725	**0.810**	1.725
average	0.396	4.982	-2.263	4.982	0.473	4.982	0.552	4.982	0.813	5.154	**0.839**	5.154

TABLE IV
RUNTIME (S) COMPARISON WITH COMMERCIAL TOOLS

Design	Commercial Tool (20 threads)				Ours			
	Timing Optimization	Routing	STA	Total	Data Processing	Inference	Total	Speedup
arm9	305	1825	16	2146	0.88	2.62	3.50	**613**×
chacha	1621	1794	23	3438	0.82	1.23	2.05	**1674**×
hwacha	43883	136946	241	181070	23.89	5.40	29.29	**6182**×
or1200	28641	40291	339	69271	112.20	14.79	118.72	**583**×
sha3	18785	16870	185	35840	24.95	1.73	126.99	**545**×
Average	18647	39545.20	160.8	58353	32.50	5.15	37.70	**2072**×

Then, the total loss for each batch can be defined as

$$\mathcal{L} = \sum_{i=1}^{|\mathcal{B}_T|} \mathcal{L}_{ELBO}(y_i^T, \mathcal{G}_i'^T, \mathcal{N}_T) + \sum_{i=1}^{|\mathcal{B}_S|} \beta_i \mathcal{L}_{ELBO}(y_i^S, \mathcal{G}_i'^S, \mathcal{N}_S) + \gamma_1 \mathcal{L}_{CLR} + \gamma_2 \mathcal{L}_{CMD} \quad (35)$$

where γ_1 and γ_2 are two hyperparameters.

TABLE V
TRAINING TIME (HRS) COMPARISON

Setting	DAC23 [4]-SimpleMerge	DAC24 [50]	Ours
130nm-to-7nm	32	35	39
45nm-to-7nm	30	33	36

VII. EXPERIMENTAL RESULTS

A. Implementation Details

We implement our entire framework with the widely used deep learning library, DGL [51] and Pytorch [52]. The model is trained and tested on a Linux system with a 2.3GHz Intel Xeon CPU and a single NVIDIA GeForce RTX 3090 GPU. Regarding the hyperparameters of our model in the experiments, we implement all the MLPs with a hidden dimension of 256 for the GNN. As for the CNN, we set the input size to $3\times512\times512$. Besides, the dimension of endpoint-wise netlist and layout embedding is set to 128. The weights for the node-based contrastive loss γ_1 and the designed-based discrepancy loss γ_2 are set to 10 and 100, respectively. Our model is trained with a learning rate of 0.0001 and batch size of 2048 for 200 epochs.

We use 7-nm node as our target node. As for preceding node data, we collect netlists from two nodes, 130 and 45 nm. Therefore, our experiments include two settings, 130-to-7 and 45-to-7 nm. For each setting, we use four 130 or 45-nm netlists and one 7-nm netlist as the training set, and five 7-nm netlists as the test set. The statistics of all the designs are shown in Table I. Specifically, we collect open-source designs from Freecores [53] and Chipyard [54]. In the dataset generation workflow, we employ Cadence Genus with 130-nm SkyWater [55], 45-nm Nangate [56], and 7-nm ASAP7 PDK [57] for synthesis. We further adopt Cadence Innovus for placement, timing optimization, routing and STA. Note that timing constraints for each design in the PnR phase are derived from estimated values provided by Cadence Genus during synthesis to ensure effective timing optimization. DEF files and netlists are acquired at each phase of the flow, enabling the retrieval of pin and cell locations, as well as other available features, in accordance with PDK rules. The prediction is performed at pre-CTS stage for all the methods. Besides, we employ the `routeDesign -globalDetail` command with default settings as the final step in our flow, without further power/timing optimization.

To verify the effectiveness of our proposed learning strategy, we set the following baselines for comparison. By default, all the baseline models are the previous state-of-the-art (SOTA)

TABLE VI
ABLATION STUDY ON THE NUMBER OF 130-NM DESIGNS. WE REPORT THE R^2 SCORE

jpeg	linkruncca	spiMaster	usbf_device	arm9	chacha	hwacha	or1200	sha3	average
✓				0.675	0.601	0.684	0.372	0.631	0.593
✓	✓			0.770	0.653	0.440	0.544	0.673	0.616
✓	✓	✓		0.861	0.718	0.794	0.637	0.733	0.749
✓	✓	✓	✓	**0.898**	**0.868**	**0.832**	**0.747**	**0.821**	**0.833**

model published in DAC23 [4], but trained with different strategies.

1) The first baseline is only trained with limited advanced 7-nm node netlist data, denoted as *DAC23-AdvOnly*.
2) The second baseline is simply merging the 7 nm and the 130 or 45-nm netlist data as the training set, denoted as *DAC23-SimpleMerge*.
3) The third baseline is parameter sharing [39], which is a common practice in transfer learning and multitask learning, denoted as *DAC23-ParamShare*. For the source node and target node data, they share the same feature extractor, but adopt a node-specific linear layer for the final prediction.
4) The fourth baseline is pretraining-then-finetuning [38], denoted as *DAC23-PT-FT*. As a widely used technique in transfer learning, this strategy first trains the timing prediction model with 130 or 45-nm netlist data and then fine-tunes it with 7-nm netlist data.

B. Main Results

The main results for the 130-to-7 and 45-to-7-nm settings are shown in Tables II and III, respectively. First of all, we can observe that only training with limited 7-nm data (DAC23-AdvOnly) achieves poor performance, indicating the necessity of massive training data belonging to the same distribution as the test data. The rest of the baselines investigate different approaches to leveraging the data from the 130 or 45-nm node to help the learning on the 7-nm node. The most intuitive one is to merge the limited data on the target node and the abundant data on the source node. However, since the arrival time on target and source nodes suffer from a large distribution discrepancy, as shown in Fig. 6, this strategy is infeasible to handle them simultaneously, leading to negative R^2 scores in both 130-to-7 and 45-to-7-nm settings. The next baseline parameter sharing (DAC23-ParamShare). As a common practice in transfer learning, it obtains some improvements compared with DAC23-AdvOnly, specifically 0.018 in the 130-to-7-nm setting and 0.077 in the 45-to-7-nm setting. This demonstrates that it somehow leverages the knowledge from the source node data to help the learning for target node data. Another common practice in transfer learning, pretraining-then-finetuning (DAC23-PT-FT), achieves superior regression performance compared with DAC23-ParamShare with a substantial margin (0.161 in the 130-to-7-nm setting and 0.079 in the 45-to-7-nm setting), indicating its effective knowledge transfer capability

Our method outperforms all the baselines by a significant margin, with improvements of about 41% and 47% in the R^2 score compared to its best counterpart, DAC23-PT-FT, in the 130-to-7 and 45-to-7-nm settings, respectively. This validates that our method is effective in handling the distribution shifts between the source and target node data and is capable of transferring knowledge to different technology nodes. Meanwhile, our method requires only about 4% additional runtime to achieve this remarkable improvement due to the added parameters. Note that all the baseline models share the same runtime since they only differ in training strategy, and they are the same during inference. Moreover, compared to [50], we observe further improvements on most benchmarks, with the average R^2 score increasing by around 2.8% and 3.2% in the 130-to-7 and 45-to-7-nm settings, respectively. This indicates the effectiveness of the proposed transferability-based reweighting algorithm.

C. Runtime Analysis

The runtime analysis is shown in Tables IV and V. First of all, since our method is used for prerouting timing predictions, we do not have to run time-consuming timing optimization and routing to get detailed wire information for the timing information. Therefore, we compare our method with an advanced commercial tool, which requires running complex timing optimization and routing. As shown in Table IV, given the results from the placement stage, the following timing optimization and routing stage will take up much time for commercial tool usage. Moreover, the post-routing STA stage also consumes some extra time. As for our method, with the netlist and layout as the input, we only need two steps to get the final timing prediction. First, we process the data to the required multimodal format, and then we only need a single model inference to output the timing delay. As we can see in Table IV, for smaller designs the data processing time is neglectable, while it will increase as the design gets larger and will dominate the whole test-time runtime for large designs, for example, or1200. The inference time also increases with the design size but with a much smaller speed compared with the data processing time. In all, the total test-time runtime for our method is much smaller than that required by the commercial tools to run full timing optimization and routing process. We can get on average 2000 times acceleration for timing prediction, which validates the efficiency and necessity of ML-based prerouting timing prediction methods.

We also show the training time comparison in Table V. Compared with our baseline model [4], in the 130-to-7-nm setting, the proposed feature disentanglement and Bayesian prediction framework introduces around 9% extra training time. Compared with [50], our proposed transferability estimation algorithm requires solving an OT problem for each iteration, which further introduces another 11% training time.

In the 45-to-7-nm setting, we can observe similar time consumption difference. The transferability estimation algorithm takes around 9% more training time. The overall training time for 45-to-7-nm setting is slightly shorter than that of 130-to-7 nm because the 45-nm netlist sizes are smaller than that of the 130-nm netlists, as shown in Table I. Generally, the training on one 7-nm design and four 130 or 45-nm designs ends in less than two days with only one Nvidia 3090 GPU.

D. Ablation Study

To further validate the effectiveness of our method, we conduct two ablation studies to show 1) the effectiveness of each module in our method and 2) the effect of the number of preceding node netlist data. We conduct the ablation study experiments in the 130-to-7-nm setting.

Effectiveness of Different Modules: To verify the effectiveness of the feature DA, the Bayesian-based timing prediction module (Bay.), and the transferability-based reweighting algorithm (Re.), we conduct ablation studies as shown in Fig. 9. For the two core modules, DA and Bayesian-based timing prediction module, the performance drops by a substantial margin without any of them, demonstrating the effectiveness of both of them. In addition, these two modules lead to different improvements on different designs. For instance, the model with DA only outperforms the model with Bayesian only on or1200 and sha3, but on arm9 and chacha model with Bayesian only owns substantial advantages. Moreover, the transferability-based reweighting algorithm further boosts the performance of our proposed transfer learning framework.

Number of Source Node Netlist Data: We also investigate how the number of source node data used in transfer learning affects the performance of the timing predictor. As shown in Table VI, when we increase the number of source node data, the timing prediction performance improves consistently. This indicates that, on the one hand, our method is effective in transferring the knowledge in different nodes to enhance the performance on the target node; on the other hand, the involvement of more source node data can improve the timing predictor's generalization ability on various 7-nm designs.

VIII. Conclusion

ML-based methods [3], [4] have achieved remarkable success in prerouting timing prediction. To ensure precise prediction, they demand extensive data from the designated technology node. However, the data collection process is time consuming, posing a challenge in acquiring adequate data for advanced technology nodes. To mitigate this issue, we propose a novel transfer learning framework that leverages abundant data from preceding technology nodes to enhance learning on the target technology node. Specifically, our method first disentangles the timing path features into node- and design-dependent parts and aligns them separately. Then, we use a Bayesian ML-based model to predict the arrival time of each timing path, which can handle its high variability and generalize to new designs in the test set. Moreover, we reweight the samples from preceding technology nodes based on their transferability to further enhance the transfer learning efficacy. Experimental results on transfer learning from 130 or 45-nm node to 7-nm node validate the effectiveness of our method.

References

[1] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in RC tree networks," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 2, no. 3, pp. 202–211, Jul. 1983.

[2] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu, "Machine learning-based pre-routing timing prediction with reduced pessimism," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, 2019, pp. 1–6.

[3] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin, "A timing engine inspired graph neural network model for pre-routing slack prediction," in *Proc. ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1207–1212.

[4] Z. Wang, S. Liu, Y. Pu, S. Chen, T.-Y. Ho, and B. Yu, "Restructure-tolerant timing prediction via multimodal fusion," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[5] J. Qiu, S. Reda, and S. Hassoun, "Fast, accurate a priori routing delay estimation," in *Proc. 12th ACM/IEEE Workshop Syst. Level Interconnect Predict. (SLIP)*, 2010, pp. 77–82.

[6] Q. Liu and M. Marek-Sadowska, "Pre-layout wire length and congestion estimation," in *Proc. 41st ACM/IEEE Design Autom. Conf. (DAC)*, 2004, pp. 582–587.

[7] S. Bodapati and F. N. Najm, "Pre-layout estimation of individual wire lengths," in *Proc. ACM Workshop Syst. Level Interconnect Predict. (SLIP)*, 2000, pp. 93–98.

[8] H.-H. Cheng, I. H.-R. Jiang, and O. Ou, "Fast and accurate wire timing estimation on tree and non-tree net structures," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.

[9] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM Int. Conf. Knowl. Discov. Data Min. (KDD)*, 2016, pp. 785–794.

[10] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi, "Fast and accurate wire timing estimation based on graph learning," in *Proc. IEEE/ACM Design, Autom. Test Eurpoe (DATE)*, 2023, pp. 1–6.

[11] A. B. Kahng, U. Mallappa, L. Saul, and S. Tong, "'Unobserved corner' prediction: Reducing timing analysis effort for faster design convergence in advanced-node design," in *Proc. IEEE/ACM Design, Autom. Test Eurpoe (DATE)*, 2019, pp. 168–173.

[12] W. W. Xing et al., "TOTAL: Multi-corners timing optimization based on transfer and active learning," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[13] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo, "Accurate timing prediction at placement stage with look-ahead RC network," in *Proc. 59th ACM/IEEE Design Autom. Conf. (DAC)*, 2022, pp. 1213–1218.

[14] H. Chang and S. S. Sapatnekar, "Statistical timing analysis considering spatial correlations using a single PERT-like traversal," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2003, pp. 621–625.

[15] C. Chu and Y.-C. Wong, "FLUTE: Fast lookup table based rectilinear Steiner minimal tree algorithm for VLSI design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 27, no. 1, pp. 70–83, Jan. 2008.

[16] P. Cao, G. He, and T. Yang, "TF-predictor: Transformer-based prerouting path delay prediction framework," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 7, pp. 2227–2237, Jul. 2023.

[17] G. He, W. Ding, Y. Ye, X. Cheng, Q. Song, and P. Cao, "An optimization-aware pre-routing timing prediction framework based on heterogeneous graph learning," in *Proc. 29th IEEE/ACM Asia South Pac. Design Autom. Conf. (ASPDAC)*, 2024, pp. 177–182.

[18] Q. Song, X. Cheng, and P. Cao, "Critical paths prediction under multiple corners based on BiLSTM network," in *Proc. 60th ACM/IEEE Design Autom. Conf. (DAC)*, 2023, pp. 1–6.

[19] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2017, pp. 1–11.

[20] X. Zhou et al., "Heterogeneous graph neural network-based imitation learning for gate sizing acceleration," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, pp. 1–9.

[21] S. Nath, G. Pradipta, C. Hu, T. Yang, B. Khailany, and H. Ren, "TransSizer: A novel transformer-based fast gate sizer," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2022, pp. 1–9.

[22] K.-C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, "DAG-Map: Graph-based FPGA technology mapping for delay optimization," *IEEE Design Test Comput.*, vol. 9, no. 3, pp. 7–20, Sep. 1992.

[23] R. J. Francis, J. Rose, and K. Chung, "Chortle: A technology mapping program for lookup table-based field programmable gate arrays," in *Proc. 27th ACM/IEEE Design Autom. Conf. (DAC)*, 1991, pp. 613–619.

[24] K.-T. Cheng and C.-J. Lin, "Timing-driven test point insertion for full-scan and partial-scan BIST," in *Proc. IEEE Int. Test Conf. (ITC)*, 1995, pp. 506–514.
[25] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hypergraph partitioning with fixed vertices," in *Proc. 36th ACM/IEEE Design Autom. Conf. (DAC)*, 1999, pp. 355–359.
[26] N. Selvakkumaran and G. Karypis, "Multiobjective hypergraph-partitioning algorithms for cut and maximum subdomain-degree minimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 25, no. 3, pp. 504–517, Mar. 2006.
[27] B. Hu and M. Marek-Sadowska, "Fine granularity clustering-based placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 4, pp. 527–536, Apr. 2004.
[28] B. Yu et al., "Methodology for standard cell compliance and detailed placement for triple patterning lithography," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 5, pp. 726–739, May 2015.
[29] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. Comput.*, vol. C-35, no. 8, pp. 677–691, Aug. 1986.
[30] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic Design Automation: Synthesis, Verification, and Test*. Burlington, MA, USA: Morgan Kaufmann, 2009.
[31] J. Cong and P. H. Madden, "Performance driven global routing for standard cell design," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 1997, pp. 73–80.
[32] C. Albrecht, "Provably good global routing by a new approximation algorithm for multicommodity flow," in *Proc. ACM Int. Symp. Phys. Design (ISPD)*, 2000, pp. 19–25.
[33] T. Yoshimura and E. S. Kuh, "Efficient algorithms for channel routing," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 1, no. 1, pp. 25–35, Jan. 1982.
[34] Y. Lin et al., "MrDP: Multiple-row detailed placement of heterogeneous-sized cells for advanced nodes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 6, pp. 1237–1250, Jun. 2018.
[35] C. J. Alpert, A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Hypergraph partitioning with fixed vertices [VLSI CAD]," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 19, no. 2, pp. 267–272, Feb. 2000.
[36] B. Yu and D. Z. Pan, "Layout decomposition for quadruple patterning lithography and beyond," in *Proc. 51st ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, 2014, pp. 1–6.
[37] D. Ramachandram and G. W. Taylor, "Deep multimodal learning: A survey on recent advances and trends," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 96–108, Nov. 2017.
[38] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Annu. Conf. North Amer. Chapter Assoc. Comput. Linguist. (NAACL)*, 2019, pp. 4171–4186.
[39] Z. Zhang, P. Luo, C. C. Loy, and X. Tang, "Facial landmark detection by deep multi-task learning," in *Proc. 13th Eur. Conf. Comput. Vis. (ECCV)*, 2014, pp. 94–108.
[40] H. Wang et al., "GCN-RL circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, 2020, pp. 1–6.
[41] E. Goan and C. Fookes, "Bayesian neural networks: An introduction and survey," in *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair*. Cham, Switzerland: Springer, 2020, pp. 45–87.
[42] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, "Variational inference: A review for statisticians," *J. Amer. Stat. Assoc.*, vol. 112, no. 518, pp. 859–877, 2017.
[43] C. Villani, *Optimal Transport: Old and New*, vol. 338, Berlin, Germany: Springer, 2009.
[44] W. Zellinger, T. Grubinger, E. Lughofer, T. Natschläger, and S. Saminger-Platz, "Central moment discrepancy (CMD) for domain-invariant representation learning," 2019, *arXiv:1702.08811*.
[45] Z. Xiao, X. Zhen, L. Shao, and C. G. Snoek, "Learning to generalize across domains on single test samples," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2022, pp. 1–20.
[46] J. Zhang, C. Zhao, B. Ni, M. Xu, and X. Yang, "Variational few-shot learning," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2019, pp. 1685–1694.
[47] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2014, pp. 1–14.
[48] Z. Wang, Z. Dai, B. Póczos, and J. Carbonell, "Characterizing and avoiding negative transfer," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2019, pp. 11293–11302.
[49] M. Cuturi, "Sinkhorn distances: Lightspeed computation of optimal transport," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2013, pp. 1–9.
[50] X. Zhang et al., "Disentangle, align and Generalize: Learning a timing predictor from different technology nodes," in *Proc. 61st ACM/IEEE Design Autom. Conf. (DAC)*, 2024, pp. 1–6.
[51] M. Y. Wang, "Deep graph library: Towards efficient and scalable deep learning on graphs," in *Proc. ICLR Workshop Represent. Learn. Graphs Manifolds*, 2019, pp. 1–18.
[52] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Proc. 33rd Annu. Conf. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 8026–8037.
[53] "Freecores." Accessed: Jan. 1, 2023. [Online]. Available: https://github.com/freecores
[54] "Chipyard." Accessed: Jan. 1, 2023. [Online]. Available: https://github.com/ucb-bar/chipyard
[55] "Skywater open source PDK." Accessed: Jan. 1, 2023. [Online]. Available: https://github.com/google/skywater-pdk
[56] "Nangate45 open source PDK." Accessed: May 1, 2023. [Online]. Available: https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45
[57] L. T. Clark et al., "ASAP7: A 7-nm finFET predictive process design kit," *Microelectron. J.*, vol. 53, pp. 105–115, Jul. 2016.

Xinyun Zhang received the B.Eng. degree from the School of Electrical Engineering, Xi'an Jiaotong University, Xi'an, China, in 2018, and the M.S. degree from the School of Engineering, The Hong Kong University of Science and Technology, Hong Kong, in 2020. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His current research interests include machine learning in EDA and computer vision.

Binwu Zhu received the B.E. degree from the Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China, in 2020, and the Ph.D. degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2024.

He is currently an Associate Professor with the School of Integrated Circuit, Southeast University, Nanjing, China. His research focuses on the combination of AI and EDA, especially in the DFM domain.

Fangzhou Liu received the B.Eng. degree from the School of Electronic Science and Engineering, Nanjing University, Nanjing, China, in 2023. She is currently pursuing the Ph.D. degree with The Chinese University of Hong Kong, Hong Kong, under the supervision of Prof. Bei Yu.

Her research focuses on applying machine learning techniques to EDA and logic synthesis optimization.

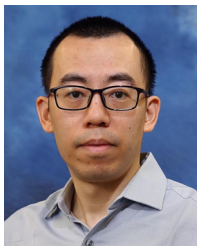
Jiaxi Jiang received the B.E. degree in computer science from Northwestern Polytechnical University, Xi'an, China, in 2023. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

His current research interests include physical design and machine learning applications in EDA.

Ziyi Wang received the B.S. degree from the Department of Computer Science and Technology, Fudan University, Shanghai, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. His research interests include graph learning applications in electronic design automation and logic synthesis.

Peng Xu received the B.S. degree from Central South University, Changsha, China, in 2019, and the M.S. degree from the Harbin Institute of Technology (Shenzhen), Shenzhen, China, in 2021. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, under the supervision of Prof. Bei Yu.

His research interests include machine learning for analog physical design and optimization in EDA problems.



Hong Xu (Senior Member, IEEE) received the B.Eng. degree from The Chinese University of Hong Kong, Hong Kong, in 2007, and the M.A.Sc. and Ph.D. degrees from the University of Toronto, Toronto, ON, Canada, in 2009 and 2013, respectively.

He is an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong. From 2013 to 2020, he was with The City University of Hong Kong, Hong Kong. His research area is computer networking and systems, particularly big data systems and data center networks.

Dr. Xu was the recipient of an Early Career Scheme Grant from the Hong Kong Research Grants Council in 2014. He received three best paper awards, including the IEEE ICNP 2015 Best Paper Award. He is a Senior Member of ACM.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Austin, TX, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received 11 Best Paper Awards from ICCAD 2024 and 2021 and 2013, IEEE TSM 2022, DATE 2022, ASPDAC 2021 and 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD contest awards, the IEEE CEDA Ernest S. Kuh Early Career Award in 2021, the DAC Under-40 Innovator Award in 2024, and the Hong Kong RGC Research Fellowship Scheme Award in 2024. He has served as a TPC Chair for ACM/IEEE Workshop on Machine Learning for CAD in 2019, International Symposium of EDA in 2025, and in many journal editorial boards and conference committees.