ASAP: Accurate Synthesis Analysis and Prediction with Multi-Task Learning

Yikang Ouyang
HKUST(GZ)

Sicheng Li
Alibaba DAMO Academy

Dongsheng Zuo
HKUST(GZ)

Hanwei Fan
HKUST(GZ)

Yuzhe Ma
HKUST(GZ)

2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD) | 979-8-3503-0955-3/23/$31.00 ©2023 IEEE | DOI: 10.1109/MLCAD58807.2023.10299840

***Abstract*—With the ever-growing scale of circuits, the design time also increases dramatically and hinders an efficient chip development process. One way to accelerate the chip design process is to equip designers with early estimations of the circuit metrics without actually running the time-consuming circuit implementation flow. In this paper, we propose a methodology for accurate synthesis results prediction based on deep neural networks. More specifically, reckoning the relevance of circuit metrics during synthesis, we propose to use multi-task learning (MTL) to simultaneously predict circuit delay and area after logic synthesis, given the hardware description language design and the synthesis configuration sequence. A multi-head attention mechanism is developed to allow knowledge sharing between the predictions for delay and area to improve the model performance. Experimental results on 780,000 data points show that the testing mean-absolute-percentage-error (MAPE) on unseen designs can achieve 6%, which is about 3× lower than existing studies. Moreover, we demonstrate that the proposed MTL model can facilitate circuit design space exploration, which can effectively obtain superior designs in terms of area and delay.**

I. Introduction

The design complexity of modern very large integrated circuits keeps soaring, and the major challenge that hinders the expedition of hardware design is the long delay in obtaining feedback on certain metrics, as the electronic design automation (EDA) flow for hardware realization is time-consuming. One way to address the above issue is to develop various models to conduct agile analysis and predict the quality-of-results (QoR) without launching the EDA flow. Hence designers can obtain the QoR (e.g., area, timing, and power) in a shorter time and conduct design space exploration more efficiently.

Many previous works have investigated building these analysis and prediction models. Regarding the design abstraction levels, various analysis and prediction models are proposed for the architectural level [1], [2], register transfer level or hardware description language level [3], [4], and netlist level [5]. Regarding the modeling methodologies, there are analytical models and machine learning-based models.

Analytical models consider functionality, complexity, and other circuit properties, along with the technology used for circuit implementation to predict the QoR [1], [6].

Machine learning-based methods train machine learning models like support vector regression and artificial neural networks with essential circuit features to predict metrics [7], [8]. Due to the powerful modeling capabilities demonstrated by deep learning, methods such as Convolutional Neural Networks (CNN) [9] and Transformer [4] have been explored to further enhance prediction accuracy. Considering the structure of circuits can be naturally represented as graphs, graph neural networks prove to be valuable in learning circuit features and predicting the QoR metrics for circuit implementations [3], [5], [10].

Furthermore, it is observed that circuit metrics can be influenced by the synthesis configuration, which denotes a sequence of logic optimizations in the synthesis process [11]. As a result, synthesis configurations have been leveraged for making predictions using long short-term memory (LSTM) models or CNNs [12], [13].

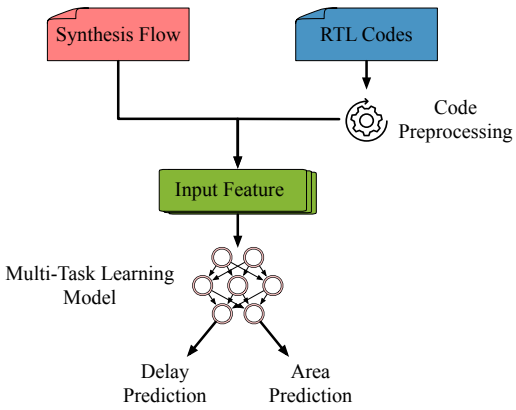


Fig. 1 Overview of our method. It takes RTL design and synthesis flow information to predict area and delay simultaneously.

While previous works have achieved considerable estimation accuracy, they all share a common limitation of considering each metric prediction as an individual task. In this approach, dedicated features are extracted for each metric and used to build separate models. However, in VLSI design and implementation, the relevance among different QoR metrics is ubiquitous. For example, optimizing the area may also have an impact on the final power or timing of a design. Within the logic synthesis, numerous practices can be identified that exemplify these trade-offs. In multi-level logic optimization, techniques like collapsing and substitution reduce delay (resp. area) but come at the cost of increasing area (resp. delay) [14]. Similarly, optimizations like re-substitution on And-Inverter-Graph (AIG) can save area by introducing more delay [15]. In addition, during technology mapping and optimization, the relevance of delay and area also exists in strategies for implementing designs with delay and area trade-offs, which include gate-sizing, buffer insertion, etc. The analysis above indicates that there exist implicit principles in the circuit implementation that are shared among optimization for different metrics, which can be leveraged to facilitate QoR modeling and prediction.

The recent surge in multi-task learning (MTL) techniques [16] has demonstrated promising potential in simultaneously addressing several relevant tasks using a unified model. By allowing knowledge-sharing among tasks, there is an expectation of improved performance across all tasks, as widely observed in other deep learning applications [17], [18]. Considering the relevance of delay and area during the circuit synthesis, we aim to develop a unified model to simultaneously predict the delay and area of a design, given the RTL design and the synthesis configuration. The predictions enable agile design iteration and guide further design space exploration effectively. The overview of our work is shown in Fig. 1.

979-8-3503-0955-3/23/$31.00 ©2023 IEEE

Our multi-task learning model first contains a primary module consisting of a GNN and an LSTM to extract synthesis-related features. Then feature extraction and feature-sharing modules based on the attention mechanism [19] are integrated to obtain task-specific features for the accurate delay and area predictions, which will be elaborated in Section III.

We construct a large dataset that contains 780,000 data points to validate the proposed MTL model and compare it with previous methods. The results show that the proposed MTL model can generalize well to other unseen designs. Moreover, we extend the application of our MTL model to perform design space exploration to validate the effectiveness of the proposed model in real front-end design scenarios.

Our contributions are summarized as follows:

- We are the first to solve circuit QoR prediction tasks by multi-task learning, which improves prediction accuracy.
- We leverage attention-based feature extraction and feature-sharing modules to obtain specific features to predict delay and area.
- Experimental results show that the proposed MTL model can obtain more accurate predictions than previous works, achieving a mean-absolute-percentage-error (MAPE) of less than 6%.
- We further validated our proposed MTL model for RTL design space exploration to predict Pareto-optimal points. Compared with all baseline methods, we achieved over 15% improvements in terms of hypervolume.

II. Preliminary

A. Graph Neural Network

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents nodes and $\mathcal{E}$ represents edges, respectively. Let $\mathbf{X} : \{\boldsymbol{x}_v, \forall v \in \mathcal{V}\}$ be the node feature matrix for a graph with N nodes, we have $\boldsymbol{x}_v \in \mathbb{R}^{d_x}$ and $\mathbf{X} \in \mathbb{R}^{d_x \times N}$. Graph neural networks (GNNs) aim to learn the graph embedding through stacking k layers, which yield node embeddings $\boldsymbol{e}_v^{(1)}, \boldsymbol{e}_v^{(2)}, ..., \boldsymbol{e}_v^{(k)}$ for each node v. Specifically, the embedding at the first layer is the node feature, i.e., $\boldsymbol{e}_v^{(1)} = \boldsymbol{x}_v$.

There have been many powerful GNNs raised to learn from graph-structured data. Graph Isomorphism Network (GIN) [20] is one of the most powerful ones, which adds up all neighbors' embedding to represent the neighboring structures injectively.

B. Multi-Task Learning

Multi-task learning [16] is a new learning paradigm in machine learning which differs from typical methods that train one model for each task. It usually uses a unified model for multiple tasks with task-specific branches and shares features among them to improve the performance of all tasks. Researchers in the MTL field are constantly working on inventing advanced mechanisms for feature sharing to improve model performance, like linear combinations of activations [17], attention-based feature sharing [18], etc.

III. Methodologies

A. Overview

This section will describe the workflow of our method in predicting delay and area. Some notations are shown in TABLE I for a clearer illustration of our model. It contains the following phases:

1) *Circuit Pre-processing*: To make our work adaptable for different hardware design language descriptions, we first transform the circuit to AIG, which only comprises *and* gates and *not* gates. The generated AIG is a directed acyclic graph (DAG). This transformation is done by the open-source synthesis tool Yosys [21].
2) *Synthesis-Related Feature Extraction*: The framework of our multi-task learning model is shown in Fig. 2. It first uses a primary module to extract features related to synthesis.

TABLE I Notations in Multi-task Learning Model

Notations	Explanations
$\mathbf{X}$	The input node feature matrix.
$\boldsymbol{e}_v^{(l)}$	Node embedding of node v at layer l.
$\boldsymbol{s}$	The input synthesis sequence.
$\mathbf{T}_c$	The synthesis-related feature from the primary module.
$\mathbf{T}_i^{(j)}$	Feature from layer j in feature extraction module for task i.
$\mathbf{F}_i^{(j)}$	Feature from layer j in feature-sharing module for task i.
$\mathbf{Q}, \mathbf{K}, \mathbf{V}$	The input matrices for multi-head attention.
$\mathbf{W}_g^{(Q)}, \mathbf{W}_g^{(K)}, \mathbf{W}_g^{(V)}$	The projection matrices in multi-head attention for $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ at head g, respectively.
$\mathbf{H}_g$	The computation at head g in multi-head attention.
$\mathbf{W}^{(O)}$	The matrix for fusing outputs from G heads in multi-head attention.
$\boldsymbol{c}_i$	The classifier's output in ensemble prediction for task i.
$\boldsymbol{m}_i$	The MLPs' output in ensemble prediction for task i.

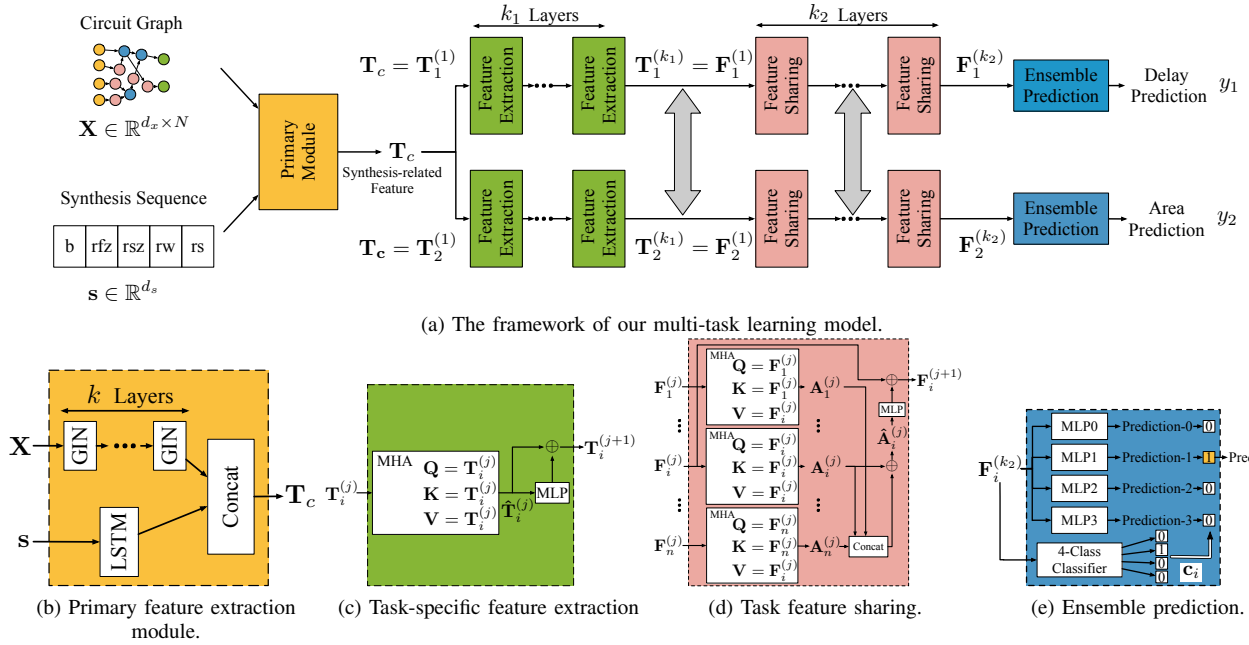
3) *Task-Specific Feature Extraction*: Our model extracts specific delay and area prediction features through two task-specific feature extraction modules.
4) *Task Feature Sharing*: After task-specific features are extracted, feature-sharing modules use an attention mechanism to relate delay and area features and allow sharing between each other.
5) *Ensemble Prediction*: The ensemble prediction modules take the features after sharing and use a set of multi-layer perceptrons (MLPs) and a classifier synergistically to predict delay or area.

B. Synthesis-Related Feature Extraction

First, we use a primary module containing GIN and LSTM to extract crucial features in logic synthesis, i.e., circuit and synthesis flow features. It is shown in Fig. 2(b). The GIN learns graph embedding through aggregating transformed node features. We employ four types of input node features, i.e., $\boldsymbol{x}_v, \forall v \in \mathcal{V}$: node type, logic level of nodes, number of fan-in of nodes, and number of fan-out of nodes. The node features are inspired by the cut-based synthesis [11]. Cuts are clusters of AIG nodes such that each cluster consists of a set of connected nodes rooted at one node producing its output [14].

The node type is either *and* or *not*, representing the node's functionality. During synthesis, cuts in AIG will be optimized by rewriting or refactoring with cuts with the same function but less delay or area [22]. The functionality of nodes is essential as it determines what kind of cuts will be used to optimize the original one, consequently affecting the delay and area of the circuit. The logic level is the number of nodes along the longest paths from any primary input to the concerned node. Logic optimization often involves cuts with reconvergence paths to reduce delay [23]. Therefore, the difference in logic level between two nodes in a cut indicates the possible reconvergence structure and the potential for optimization. The number of fan-in and fan-out are features that indicate the connectivity of the nodes. Besides, fan-out is also an indicator of circuit QoR, as gates with large fan-out will have high capacitive loads and will be optimized during synthesis by methods like buffer insertion and gate sizing.

The computation of GIN in the primary module is shown in Equation (1). For a GIN with k layers, $\boldsymbol{e}_v^{(l)}$ is the embedding for the l-th layer, $\boldsymbol{e}_v^{(l+1)}$is the updated embedding for layer $l+1$. We



(a) The framework of our multi-task learning model.

(b) Primary feature extraction module.

(c) Task-specific feature extraction

(d) Task feature sharing.

(e) Ensemble prediction.

Fig. 2 (a) The whole framework of our multi-task learning model. It takes input as a circuit graph and synthesis sequence and outputs delay and area prediction. (b) A primary module containing GIN and LSTM that extracts primary features from the circuit structure and synthesis flow. (c) This module uses the attention mechanism to extract specific features of the circuit metrics from the synthesis-related feature. (d) This module shares specific features for circuit metrics across tasks. (e) Ensemble prediction module uses a set of MLPs and a classifier together to give an accurate prediction.

have $\boldsymbol{e}_v^{(1)} = \boldsymbol{x}_v$ as the initial node embedding.

$$\boldsymbol{e}_v^{(l+1)} = \mathrm{MLP}^{(l)}\left(\left(1+\epsilon^{(l)}\right)\cdot \boldsymbol{e}_v^{(l)} + \sum_{u\in\mathcal{N}(v)} \boldsymbol{e}_u^{(l)}\right), \tag{1}$$

where MLP is multi-layer perceptron, and $\epsilon^{(l)}$ is a learnable parameter.

The updated node embedding contains embeddings from the concerned node and its neighbors, which are also transitive fan-in nodes in the DAG. Consequently, the node embedding reflects the structure and functionality of the cut rooted at the concerned node. We further add up all nodes' embeddings to get the embedding of the entire circuit.

Apart from the circuit itself, the logic synthesis flow also affects the circuit QoR after implementation and can be represented as a sequence [10], [12]. We use LSTM to learn the embedding of synthesis flows. For each circuit and synthesis flow, we concatenate the graph embedding provided by the GIN with the synthesis sequence embedding generated by LSTM. The concatenation resembles the process of applying logic optimizations on the circuit. The concatenated embedding is our primary module's output and encompasses crucial features of circuit and synthesis flows.

C. Task-Specific Feature Extraction and Sharing

After we get the features related to logic synthesis and the circuit structure from a high level, we proceed to derive task-specific features for the delay and area prediction tasks. Inspired by [18], we use the multi-head attention mechanism [19] in our multi-task learning model. For feature extraction, self-task attention is used to extract distinct feature representations for delay prediction and area prediction tasks. Then, feature-sharing modules use cross-task attention to help each task get relevant and beneficial features from the other task.

The multi-head attention (MHA) allows a model to attend to parts of the information globally. The $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are the inputs of MHA and are called *query*, *key*, and *value*, respectively. Suppose there are G heads and each head is with projection matrices $\mathbf{W}_g^{(Q)}, \mathbf{W}_g^{(K)}, \mathbf{W}_g^{(V)}$ to project the *query*, *key*, *value* so that different heads can attend to information from different representation subspaces of the input [19]. The computation is as follows:

$$\mathrm{MHA}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \mathrm{Concat}\left(\mathbf{H}_1,\ldots,\mathbf{H}_\mathrm{G}\right)\mathbf{W}^{(O)}, \tag{2}$$

where $\mathbf{H}_g, g \in \{1,2,\ldots,G\}$, is the output of a single scaled dot-product attention and G is the number of heads. For $\mathbf{H}_g$, it computes the similarity between projected $\mathbf{Q}\mathbf{W}_g^{(Q)}$ and $\mathbf{K}\mathbf{W}_g^{(K)}$ and applies it to the projected $\mathbf{V}\mathbf{W}_g^{(V)}$ to get the attention between $query$ and $value$ through looking into key, which is computed by:

$$\begin{aligned}\mathbf{H}_g &= \mathrm{Attention}\left(\mathbf{Q}\mathbf{W}_g^{(Q)}, \mathbf{K}\mathbf{W}_g^{(K)}, \mathbf{V}\mathbf{W}_g^{(V)}\right)\\ &= \mathrm{softmax}\left(\frac{(\mathbf{Q}\mathbf{W}_g^{(Q)})(\mathbf{K}\mathbf{W}_g^{(K)})^T}{\sqrt{d_f}}\right)\mathbf{V}\mathbf{W}_g^{(V)},\end{aligned} \tag{3}$$

where d_f is the feature dimension for input $\mathbf{K}$. Finally, the projection matrix $\mathbf{W}^{(O)}$ is used to fuse the attention from each head and get a feature with richer information.

The feature extraction modules will extract specific features for delay and area. The input $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are the same as the synthesis-related feature $\mathbf{T}_c$ from the primary module. This module is shown in Fig. 2(c), and we call the output of MHA *attention map*. We first compute the attention map for task i by feature from layer j in feature extraction modules and add it again as Equation (4) shows.

$$\hat{\mathbf{T}}_i^{(j)} = \mathrm{MHA}(\mathbf{T}_i^{(j)}, \mathbf{T}_i^{(j)}, \mathbf{T}_i^{(j)}) + \mathbf{T}_i^{(j)}. \tag{4}$$

Then we use MLP to project the attention map further and add it back to get the specific feature for the next layer, as Equation (5) shows.

$$\mathbf{T}_i^{(j+1)} = \mathrm{MLP}(\hat{\mathbf{T}}_i^{(j)}) + \hat{\mathbf{T}}_i^{(j)}. \tag{5}$$

The MHA in this module can attend to delay or area information from the input synthesis-related feature. The original feature $\mathbf{T}_i^{(j)}$ and the attention map $\hat{\mathbf{T}}_i^{(j)}$ is added back in this module MHA, so multiple layers can be stacked to extract more specific features for each task from the synthesis-related feature in a repetitive manner.

After extracting task-specific features, feature-sharing modules aim to help the prediction for one circuit metric obtain beneficial features from the prediction for the other metric.

Without the loss of generality, the computation for the feature of task i is shown in Fig. 2(d). It takes the specific features from all tasks as the input $\mathbf{Q}, \mathbf{K}$ and the features from task i as $\mathbf{V}$. For the feature of task i at layer j, $\mathbf{F}_i^{(j)}$, we can get n attention maps for n tasks.

$$\begin{aligned} \mathbf{A}_1^{(j)} &= \mathrm{MHA}\left(\mathbf{F}_1^{(j)}, \mathbf{F}_1^{(j)}, \mathbf{F}_i^{(j)}\right); \\ &\vdots \\ \mathbf{A}_n^{(j)} &= \mathrm{MHA}\left(\mathbf{F}_n^{(j)}, \mathbf{F}_n^{(j)}, \mathbf{F}_i^{(j)}\right). \end{aligned} \tag{6}$$

The initial input features for feature-sharing modules are specific features for circuit metrics extracted previously. After we get n attention maps for each task, we concatenate all of them and use an MLP for projection. Then, the attention map is added back from task i to get a fused attention map $\hat{\mathbf{A}}_i^{(j)}$.

$$\hat{\mathbf{A}}_i^{(j)} = \mathrm{MLP}(\mathrm{Concat}(\mathbf{A}_1^{(j)}, \ldots, \mathbf{A}_n^{(j)})) + \mathbf{A}_i^{(j)}. \tag{7}$$

Finally, $\hat{\mathbf{A}}_i^{(j)}$ is further projected and the original feature of task i, $\mathbf{F}_i^{(j)}$, is added back to generate the new feature $\mathbf{F}_i^{(j+1)}$ for task i:

$$\mathbf{F}_i^{(j+1)} = \mathrm{MLP}(\hat{\mathbf{A}}_i^{(j)}) + \mathbf{F}_i^{(j)}. \tag{8}$$

MHAs will relate features for each circuit metric, i.e., $\mathbf{F}_i^{(j)}, i \in \{1, 2, \ldots, n\}$, with the features for the concerned feature at task i, i.e., $\mathbf{F}_i^{(j)}$. This is achieved by computing attention maps with $\mathbf{Q}, \mathbf{K}, \mathbf{V}$, where $\mathbf{Q}$ and $\mathbf{K}$ are from all circuit metrics being predicted, and $\mathbf{V}$ is from the concerned one. The features are then shared to the task i through concatenation. For each task self-task attention map $\mathbf{A}_i^{(j)}$ and the self-task feature $\mathbf{F}_i^{(j)}$ are added back during the computation. It aligns with the intuition that when predicting delay (resp. area), delay (resp. area) features should take the principal role while area (resp. delay) features take the auxiliary role.

D. Ensemble Prediction Module

Conventionally, regression models like MLPs are used to conduct the final prediction for delay and area of a design. However, the sizes of circuits with different functions vary significantly, and the ground-truth values also span across a large range. For instance, the delay of a decoder is $26ps$, whereas a square root circuit can have a delay of $21500ps$ [24], leading to a tough training process.

To handle this problem, we use an ensemble prediction module to get the final prediction for each task. We set 4 data ranges for our data points and use 4 MLP branches for each task, each for one data range. They are fed with the same feature generated by the feature-sharing module. We also feed the feature to a classifier to decide which output range should be used among the four MLP branches.

For each data point, the outputs of four MLPs for task i can be viewed as a vector, denoted as $\boldsymbol{m}_i \in \mathbb{R}^4$. The output of the classifier is a one-hot vector denoted by $\boldsymbol{c}_i \in \mathbb{R}^4$ that indicates which output of the MLPs should be used as the final prediction y_i. To get y_i, we can take the product of $\boldsymbol{m}_i$ and $\boldsymbol{c}_i$, as shown in Equation (9) and Fig. 2(e).

$$y_i = \boldsymbol{m}_i^T \boldsymbol{c}_i. \tag{9}$$

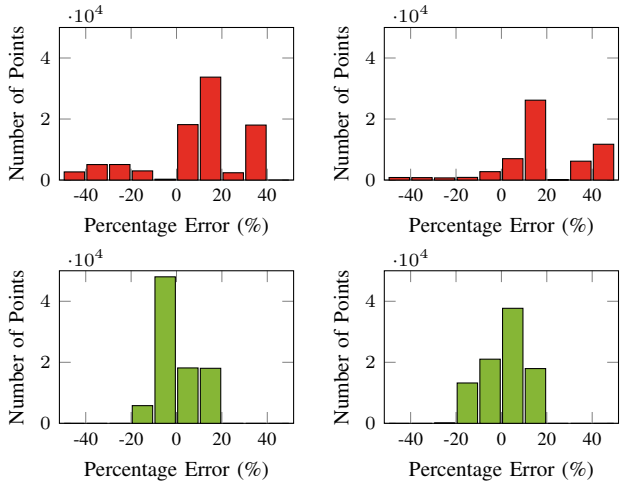


Fig. 3 Distribution of the prediction errors. Each row, from left to right, shows the number of predictions based on the percentage error of inductive testing results on area and delay, respectively. The first row is for LOSTIN [10], and the second is for the proposed method.

We use Mean-Squared-Error (MSE) as the final prediction loss function, as computed by

$$L_i^{(pred)} = (y_i - \hat{y}_i)^2, \tag{10}$$

where $\hat{y}_i$ is the ground truth for prediction y_i. In addition, we can also get the label of the classification result $\hat{\boldsymbol{c}}_i \in \mathbb{R}^4$ and utilize an auxiliary loss function for the classifier, in which the cross-entropy (CE) loss is applied and can be formulated as

$$L_i^{(aux)} = \mathrm{CE}(\boldsymbol{c}_i, \hat{\boldsymbol{c}}_i). \tag{11}$$

Therefore, for a single data point, the loss can be calculated by combining the losses from all the tasks, as computed by

$$L = \sum_{i=1}^{n} (L_i^{(pred)} + L_i^{(aux)}). \tag{12}$$

IV. Experimental Results

In this section, we first describe the dataset details and the experiment setups. Then we compare the performance of circuit QoR prediction of our MTL model against a set of baseline models, including conventional single-task prediction models and other multi-task learning models. We also show that the proposed method can facilitate design space exploration by predicting Pareto-optimal points.

A. Dataset Generation and Evaluation

The input to the synthesis results prediction model contains two parts: a design and a synthesis sequence. We collect the designs from EPFL15 benchmark [24], ISCAS85 benchmark [25], an online adder generator [26], and an in-house multiplier generator, which contains 39 combinational circuits in total. Then we synthesize each circuit with 20000 different synthesis flows and map them using ASAP 7nm low-voltage technologies [27] with open-source synthesis tool ABC [11] to get the ground truth of delay and area.

To assess the models' ability to generalize, we devise two testing scenarios. The first scenario, termed *transductive testing*, comprises seen designs with unseen synthesis sequences during training. This scenario corresponds to exploring synthesis flows that yield favorable QoR during circuit implementation. The second scenario, referred to as *inductive testing*, consists of unseen circuits during training with all

TABLE II MAPE Results Comparison with Baseline Methods.

		LR	CNN [13]	LSTM [12]	LOSTIN [10]	2E[a] [16]	SA[b] [18]	CS[c] [17]	Ours
Transductive Testing	Delay Prediction	23.37%	42.21%	80.27%	11.52%	1.32%	0.90%	1.34%	**0.79%**
	Area Prediction	98.07%	63.58%	16.30%	10.85%	1.23%	1.06%	1.22%	**0.83%**
Inductive Testing	Delay Prediction	26.33%	38.04%	-	22.51%	7.78%	7.27%	12.35%	**5.80%**
	Area Prediction	146.05%	74.57%	-	25.78%	11.11%	7.63%	9.34%	**5.84%**

[a] **2-Ensemble** baseline. [b] **Self-attention** baseline. [c] **Cross-stitch** baseline.

synthesis sequences. It corresponds to the exploration of entirely new designs. For the design points used for training, we split the training, validation, and test set with a ratio of 20%, 5%, and 75%, respectively.

B. Model Configuration

For our MTL model, we choose the node embedding dimension to be 128 (32 for each node feature) in the primary module. The number of layers of the GIN and LSTM are both 2. The dimension for the hidden layers in MLPs in the ensemble prediction modules is 200. Both task-specific feature extraction and feature-sharing modules have 2 layers with 4 heads in the multi-head attention. We use an Adam optimizer with a learning rate of 0.005.

C. Comparison with Baseline Methods

We use Mean Absolute Percentage Error (MAPE) to evaluate the performance of our work. It computes the percentage error between prediction value y_m and ground-truth value $\hat{y}_m$ for M prediction values, as shown in Equation (13).

$$\text{MAPE} = \frac{100\%}{M} \sum_{m=1}^{M} \left| \frac{\hat{y}_m - y_m}{\hat{y}_m} \right| \tag{13}$$

We conduct a comprehensive comparison of our model with both single-task and multi-task methods. The single-task methods treat predicting delay and area as independent tasks, including Linear Regression (LR), CNN [13], LSTM [12], and a Spacial-Temporal model, LOSTIN [10].

To ensure the CNN and LR methods are capable of predicting unseen designs, we encode specific circuit properties (e.g., number of inputs/outputs, number of nodes, logic level, etc.) into their inputs. The LSTM method can only predict seen designs during training, but it is tested on unseen synthesis flows, i.e., transductive testing.

Regarding multi-task learning baselines, since no existing work is available, we construct them based on prior MTL research. The first baseline, denoted as **2-Ensemble**, simply employs two ensemble prediction modules to directly predict delay and area from the same synthesis-related features obtained from a primary module [16]. The second named **Self-attention**, involves using task-specific feature extraction after the primary module, incorporating self-task attention to derive specific features for delay and area [18]. The third one is a cross-stitch-based model [17], which linearly combines delay and area features extracted by two primary modules (namely **Cross-stitch**).

Our MTL model excels in both transductive and inductive testing. In transductive testing, we achieve an impressive MAPE of 0.79% and 0.83% for delay and area, respectively. Even in the more challenging inductive testing, our method performs well, with a MAPE of 5.80% and 5.84% for delay and area, respectively.

Compared with single-task baselines, our MTL model outperforms the best single-task baseline, LOSTIN [10], by more than 3 times.

In inductive testing, all predictions made by our model have errors below 20%, with most falling under 10%. In contrast, the errors in LOSTIN [10] are more widely spread, as shown in Fig. 3. This highlights our model's strong generalizability to unseen designs, avoiding predictions with large errors.

In TABLE II, we further compare our model with multi-task learning baselines (2E, SA, and CS). The 2-Ensemble, a simple MTL model directly using common synthesis-related features for both delay and area, outperforms all single-task baselines. This shows the effectiveness of multi-task learning in capturing the relevance among QoR metrics during circuit implementation, resulting in higher accuracy and generalizability. The self-attention utilizes task-specific feature extraction, providing unique representations for delay and area features, leading to higher accuracy compared to 2-Ensemble. Moreover, the cross-stitch model performs similarly to 2-Ensemble but worse than both self-attention and our model. This suggests that more complex mechanisms like attention, are better suited for exploiting the relevance of delay and area features rather than linear combinations.

The proposed MTL model utilizes self-task attention to extract specific features for delay and area from the synthesis-related features and cross-task attention to share the features with each other. The resulting feature for predicting delay (resp. area) combines task-specific features from both delay and area but is still based on delay (resp. area). Therefore it outperforms all baseline methods in both transductive and inductive scenarios on delay and area.

Judging from the MAPE value, our work reaches an error of less than 1% in the transductive testing scenario. This indicates that our model facilitates designers to explore new synthesis flows for better QoR without the need for time-consuming synthesis. Our work is also generalizable to predict designs never seen before, thus equipping designers with the ability to rapidly assess the QoR of new designs.

D. Enabling Design Space Exploration

In this section, we demonstrate that the proposed MTL model can facilitate the design space exploration workflow by alleviating the time overhead in running countless times of design implementations.

We generate a datapath circuit dataset by the open-source adder generator [26] and the in-house compressor tree multiplier generator. It contains 36 adders and 35 multipliers. Each design is synthesized by ABC [11] with the aforementioned 20000 different synthesis flows to form 20000 points to explore. There are 6 types of prefix adders, each with bit-widths of 4, 8, 16, 32, 64, and 128. The multipliers are generated with bit-widths of 4, 8, 16, and 64, each of which consists of 7 designs with different random compressor structures. For adders and multipliers of each bit-width, we randomly select one design for the inductive testing, leaving the others in the transductive testing scenario.

The design space exploration aims to identify Pareto-optimal designs concerning multiple trade-off metrics, such as delay and area. In this experiment, we utilize our MTL models and baseline models to predict designs on the datapath dataset and select those predicted to be Pareto-optimal. These selected designs are then mapped in the real design space to determine if they form a desired Pareto-frontier.

To assess the quality of the Pareto-frontiers, we use the widely-used metric, hypervolume, which represents the bounded region between the Pareto-frontier and a reference point (as shown in Fig. 4). The

TABLE III Hypervolume Ratio of Delay and Area Comparison.

Design	ADD32	ADD64	ADD128	MUL16	MUL32	MUL64	Averaged
LOSTIN [10]	0.705	0.704	0.826	0.873	0.693	0.352	0.692
2-Ensemble [16]	0.889	0.831	0.927	0.931	0.593	0.861	0.838
Self-attention [18]	0.814	0.859	0.835	0.802	0.870	0.939	0.853
Cross-stitch [17]	0.724	0.861	0.964	0.558	0.910	0.804	0.803
Ours	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**

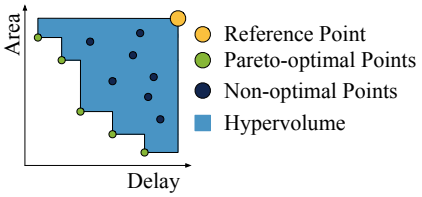


Fig. 4 An illustration of hypervolume.

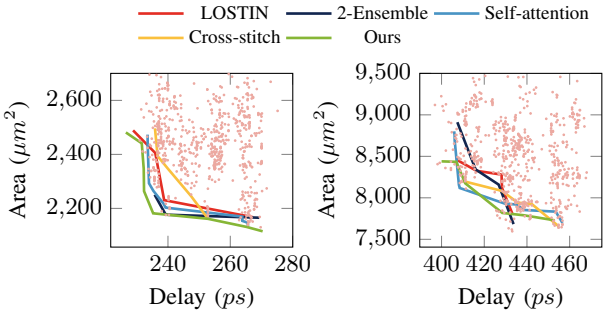


Fig. 5 Pareto-frontiers comparison of baseline methods. From left to right are results for 16-bit and 32-bit multipliers, respectively.

hypervolume value is influenced by the choice of the reference point. Thus, we compare the normalized hypervolume ratio of different Pareto-frontiers generated by various models.

For brevity, we only show the results for high bit-width designs in TABLE III. Our proposed method reaches the largest hypervolume for all bit widths. The average hypervolume ratio is 30% higher than the best single-task baseline (LOSTIN) and 15% higher than the best MTL baseline (Self-attention). For simplicity, we only show the Pareto frontiers for multipliers of 16 and 32 bit in Fig. 5, which shows that the Pareto-frontiers given by our model have better coverage of different scenarios, including small area, small delay, and area-delay trade-off.

V. Conclusion

In this paper, a multi-task learning model is proposed to predict the delay and area of RTL designs after logic synthesis. It considers the effect on circuit metrics from both the circuit structure and synthesis flows. More importantly, it leverages the relevance of circuit metrics in synthesis with attention-based feature extraction and feature-sharing mechanisms to make more accurate predictions. Experiments on a dataset of various circuits show that our model is more accurate compared with previous methods that take metric prediction as individual tasks. We also show that our MTL model facilitates design space exploration by predicting Pareto-optimal design points. Future work will focus on extending the multi-task learning to more EDA stages.

VI. Acknowledgement

This work is supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No. 2023A03J0013).

References

[1] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proc. MICRO*, 2009, pp. 469–480.

[2] C. Bai, Q. Sun, J. Zhai, Y. Ma, B. Yu, and M. D. Wong, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *Proc. ICCAD*, 2021, pp. 1–9.

[3] P. Sengupta, A. Tyagi, Y. Chen, and J. Hu, "How good is your verilog rtl code? a quick answer from machine learning," in *Proc. ICCAD*, 2022, pp. 1–9.

[4] C. Xu, C. Kjellqvist, and L. W. Wills, "Sns's not a synthesizer: a deep-learning-based synthesis predictor," in *Proc. ISCA*, 2022, pp. 847–859.

[5] Y. Zhang, H. Ren, and B. Khailany, "Grannite: Graph neural network inference for transferable power estimation," in *Proc. DAC*, 2020, pp. 1–6.

[6] A. Srinivasan, G. D. Huber, and D. P. LaPotin, "Accurate area and delay estimation from rtl descriptions," *IEEE TVLSI*, vol. 6, no. 1, pp. 168–172, 1998.

[7] S. Roy, Y. Ma, J. Miao, and B. Yu, "A learning bridge from architectural synthesis to physical design for exploring power efficient high-performance adders," in *Proc. ISLPED*, 2017, pp. 1–6.

[8] D. S. Lopera, L. Servadei, S. Prebeck, and W. Ecker, "Early rtl delay prediction using neural networks," *Microprocessors and Microsystems*, vol. 94, p. 104671, 2022.

[9] Y. Zhou, H. Ren, Y. Zhang, B. Keller, B. Khailany, and Z. Zhang, "Primal: Power inference using machine learning," in *Proc. DAC*, 2019, pp. 1–6.

[10] N. Wu, J. Lee, Y. Xie, and C. Hao, "Lostin: Logic optimization via spatio-temporal information with hybrid graph models," in *Proc. ASAP*, 2022, pp. 11–18.

[11] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Proc. CAV*. Springer, 2010, pp. 24–40.

[12] C. Yu and W. Zhou, "Decision making in synthesis cross technologies using lstms and transfer learning," in *Proc. MLCAD*, 2020, pp. 55–60.

[13] C. Yu, H. Xiao, and G. De Micheli, "Developing synthesis flows without human knowledge," in *Proc. DAC*, 2018, pp. 1–6.

[14] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[15] A. M. R. Brayton, "Scalable logic synthesis using a simple circuit structure," in *Proc. IWLS*, vol. 6, 2006, pp. 15–22.

[16] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, pp. 41–75, 1997.

[17] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *Proc. CVPR*, 2016, pp. 3994–4003.

[18] X. Xu, H. Zhao, V. Vineet, S.-N. Lim, and A. Torralba, "Mtformer: Multi-task learning via transformer and cross-task reasoning," in *Proc. ECCV*. Springer, 2022, pp. 304–321.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Proc. NIPS*, vol. 30, 2017.

[20] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[21] C. Wolf, "Yosys open synthesis suite," https://yosyshq.net/yosys/.

[22] A. Mishchenko, S. Chatterjee, and R. Brayton, "Dag-aware aig rewriting a fresh look at combinational logic synthesis," in *Proc. DAC*, 2006, pp. 532–535.

[23] R. Brayton, J.-H. R. Jiang, and S. Jang, "Sat-based logic optimization and resynthesis," in *Proc. IWLS*, 2007, pp. 358–364.

[24] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The epfl combinational benchmark suite," in *Proc. IWLS*, 2015.

[25] M. C. Hansen, H. Yalcin, and J. P. Hayes, "Unveiling the iscas-85 benchmarks: A case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[26] L. Flea, "Iamflea/addercircuitgenerator: This script generates and analyzes prefix tree adders." 2021. [Online]. Available: https://github.com/IamFlea/AdderCircuitGenerator

[27] X. Xu, N. Shah, A. Evans, S. Sinha, B. Cline, and G. Yeric, "Standard cell library design and optimization methodology for asap7 pdk," in *Proc. ICCAD*, 2017, pp. 999–1004.