RAPTA: A Hierarchical Representation Learning Solution For Real-Time Prediction of Path-Based Static Timing Analysis

Tanmoy Chowdhury
Ashkan Vakil
tchowdh6@gmu.edu
Goerge Mason University
Fairfax, Virginia, USA

Banafsheh Saber Latibari
University of California, Davis
Davis, California, USA

Sayed Aresh Beheshti Shirazi
Ali Mirzaeian
Xiaojie Guo
Sai Manoj P D
Goerge Mason University
Fairfax, Virginia, USA

Houman Homayoun
University of California, Davis
Davis, California, USA

Ioannis Savidis
Drexel University
Philadelphia, Pennsylvania, USA

Liang Zhao
Emory University
Atlanta, Georgia, USA

Avesta Sasan
University of California, Davis
Davis, California, USA

ABSTRACT

This paper presents RAPTA, a customized Representation-learning Architecture for automation of feature engineering and predicting the result of Path-based Timing-Analysis early in the physical design cycle. RAPTA offers multiple advantages compared to prior work: 1) It has superior accuracy with errors std ranges 3.9ps~16.05ps in 32nm technology. 2) RAPTA's architecture does not change with feature-set size, 3) RAPTA does not require manual input feature engineering. To the best of our knowledge, this is the first work, in which Bidirectional Long Short-Term Memory (Bi-LSTM) representation learning is used to digest raw information for feature engineering, where generation of latent features and Multilayer Perceptron (MLP) based regression for timing prediction can be trained end-to-end.

CCS CONCEPTS

• **Hardware → Static timing analysis**; • **Computing methodologies → Neural networks**.

KEYWORDS

STA; Timing Slack; LSTM; Machine Learning

ACM Reference Format:
Tanmoy Chowdhury, Ashkan Vakil, Banafsheh Saber Latibari, Sayed Aresh Beheshti Shirazi, Ali Mirzaeian, Xiaojie Guo, Sai Manoj P D, Houman Homayoun, Ioannis Savidis, Liang Zhao, and Avesta Sasan. 2022. RAPTA: A Hierarchical Representation Learning Solution For Real-Time Prediction of Path-Based Static Timing Analysis. In *Proceedings of the Great Lakes Symposium on VLSI 2022 (GLSVLSI '22), June 6–8, 2022, Irvine, CA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3526241.3530831

GLSVLSI'22, June 06–08, 2022, Orange County, CA, USA

ACM ISBN 978-1-4503-9322-5/22/06.
https://doi.org/10.1145/3526241.3530831

1 INTRODUCTION

Physical design and timing closure of Integrated Circuits (IC) becomes increasingly difficult as fabrication technology advances to smaller geometries, packing more transistors to support additional functionality. Support for Dynamic Voltage and Frequency Scaling (DVFS) for a wide range of operating modes further exacerbates the problem by forcing a design team to check and close timing in many processes, voltage, and temperature (PVT) corners.

Typically during the Place and Route (PnR), to make the physical design manageable, the physical designer uses one (or a few) design corner(s) for PnR. Timing checks are performed using the Graph-Based timing Analysis (GBA). Later, the placed and routed design are subject to Static Timing Analysis (STA), checking the timing in different Process-Voltage-Temperature (PVT) corners accuracy of which significantly improved by enabling Path-Based timing Analysis (PBA). The STA for a large SOC with numerous PVT corners would take hours to days. Unsatisfactory STA results force the physical design team to go back to one of the previous design steps to make Engineering Change Orders (ECO) or even roll back to the floorplanning or synthesis stage. Typically, a physical design goes through this iterative flow several rounds until the design passes the STA checks in desired corners. The problem with this flow is twofold: 1) designer sees the accurate STA results very late in the game, when (for using the less accurate yet faster GBA for PPA optimization) the damage is already done. 2) STA runs (especially in PBA mode) significantly increase the design cycle time, and unfortunately, the iterative design process asks for too many STA runs until the timing is evaluated and closed in all desired PVT corners.

At the same time, the early launch of PBA analysis at PnR flow across many corners and many modes significantly increases the EDA licensing cost, slows the design progress, and makes it costly and exhaustively time-consuming to perform design space exploration. For this reason, many researchers have expressed the need for the adoption and use of machine learning solutions for timing analysis as a near-term challenge for IC design [7, 8, 13].

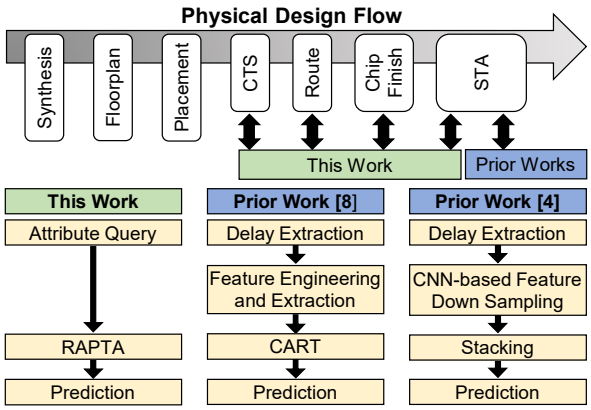

Figure 1: Our proposed solution compared to prior work. (top): The physical design flow and the stage where each learning model can predict the timing impact. (bottom): The flow of learning solutions for proposed and prior art. Note that RAPTA does not require feature engineering.

This paper proposes a novel learning solution that allows obtaining fast yet accurate timing checks simultaneously in all PVTs at early PnR stages. It enables a designer to perform "what if" analysis in real-time and see the timing impact of physical design changes in seconds without waiting for the STA on the routed design. Our learning model is developed explicitly to model timing paths by jointly handling representation learning and slack prediction. The proposed solution is different from prior art solutions to predict the STA results as it removes the need for manual engineering of input features. RAPTA is a hierarchal solution that uses three bidirectional long short-term memory (BiLSTM) arrays in the first layer to digest input features from the launch, capture, and data subpath of a full timing path. RAPTA uses fully connected layers in the second layer to predict subpath delays and path-level slack targets. The model ingests all gate and net properties reported by the EDA tool without the need for feature engineering. Using BiLSTM arrays for subpath feature processing (representation learning) allows the designer to change the number of input features (properties) for each gate and net without changing the architecture. This natural mapping makes the application and adoption of this learning model intuitive and effortless. Our proposed solution is a valuable tool to cut down the rounds of physical design, shorten the time to market, and eliminate the need for STA analysis of doomed designs that would fail the STA.

2 BACKGROUND

The applicability of machine learning for improving various aspects of physical design and its security especially stages relying on heuristic algorithms, has been showcased by researchers [2, 12, 17, 19–22, 25]. Examples include the use of learning to guide the global routing [12], routability prediction [24], shape deformation layout correction [18], and lithography hotspot detection [25]. Several prior publications have also investigated the use of learning for timing analysis. The work in [17] proposes using Multi-Layer Perceptrons (MLP) to approximate statistical SUM and MAX operations in the context of timing analysis to estimate the signal arrival times' distribution. The author in [17] proposes using Deep Learning Neural Network (DNN) as a Static Timing Analysis (SSTA) method to approximate signal arrival times. They consider gates input delay and gates input to output delay but do not consider nets delay. Authors in [9] present a backward elimination strategy to find out a small set of PVT corners for training a linear regression model. This model is then used to predict the timing information of other PVT corners not included in the train set. In [4], the authors investigated a learning solution that could predict the correlation among path delays across different voltages to predict the timing information for missing voltage corners. In [9], authors use a backward elimination strategy to find out the order of the corners, using which they select a subset of corners to predict the STA result in the rest of the PVT corners. Similarly, in [4], the authors investigated a learning solution that could indicate the correlation among path delays across different voltages.

The most relevant prior work to our proposed solution is the learning-based static timing prediction solutions proposed in [8]. This paper [8] presents a machine learning model using bigrams of path stages to predict PBA by selecting model features from GBA analysis. However, this solution faces a few shortcomings that we have addressed in our proposed solution. 1) It [8] relies on the availability of GBA results (either from the PnR tool or the STA) for making the PBA prediction. In contrast, we only rely on the gate and net characteristics reported by the EDA tool in our solution. This difference removes the need to wait for the EDA tool to perform a timing update after a physical change to compute the PBA result. Therefore our solution provides the designer with real-time feedback. 2) the model in [8] relies on manual feature engineering and uses a limited number of (13) engineered features as input. In contrast, our feature engineering is fully automated. Our model takes all the gate and gate properties (reported by EDA) into a representation learning model and automates the generation of useful latent features. Hence, our solution discovers the essential features on its own that can minimize the training error.

3 PROPOSED SOLUTIONS

Problem Statement: Given the raw properties of a timing path reported by the EDA tool, we seek to formulate a learning solution that accurately predicts its PBA timing slacks (for setup or hold analysis) across all desired PVT corners without manual feature engineering.

Solution: Figure 2 illustrates the hierarchical formulation of our proposed and customized learning solution, RAPTA, for accurate prediction of the path-based timing results in early physical design stages. The model is designed to receive features from one timing path, P (including data (P_D), launch (P_L), and capture (P_C) and predict the slack T reported from STA analysis in PBA mode.

We first briefly describe the RAPTA's overall architecture and the existential philosophy for each sub-model for building an accurate regression model. Then in the following sub-sections, each sub-model and the theory and formulation used for modeling and training the sub-models are explained in detail.

The hierarchical RAPTA is a customized learning solution that works as follows: 1) Each data path, P information is collected separately for each of its sub-paths (data (P_D), launch (P_L), and capture (P_C)), where the raw gate and net properties (as reported by the EDA tool without any feature engineering) are combined into Bigrams, g. Each bigram is composed of one gate and its proceeding net that

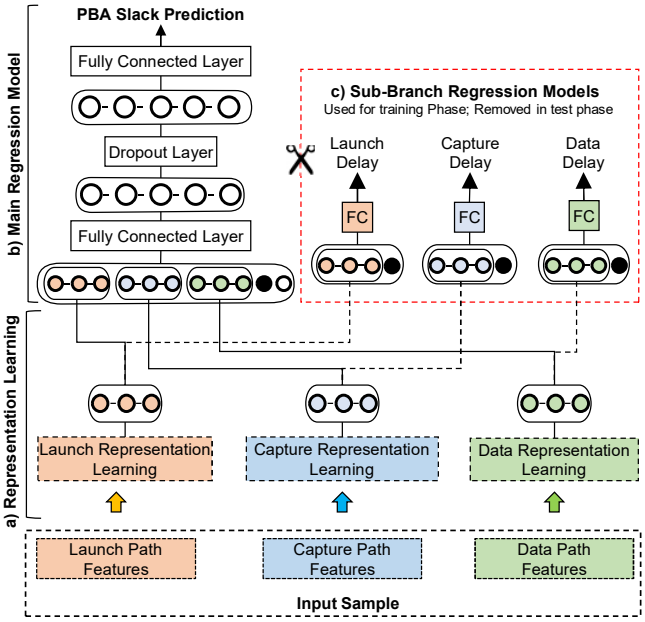

Figure 2: Proposed Model Architecture: The RAPTA model comprises three learning modules; a) Representation Learning, b) Main Regression Model, and c) Branched Sub-label Regression Models. Processing a timing path as bigram features as part of data processing has been discussed in 3.1. Then the bigram information of launch, capture, and data path is first fed to the Representation learning module (composed of a BiLSTM network) as input samples. The architecture of the Representation Learning is described in 3.2. The extracted features are fed to a regression model (composed of a sequence of a fully connected network, a dropout layer, and a final fully connected network with one output) for predicting the regression result (3.3). The representation learning models' outputs are also separately fed to three parallel branches (composed of a fully connected network) for sub-label predictions. The sub-label prediction branches, the architecture of which is discussed in section 3.4, are only used at training time and are removed during the test.

belongs to the timing path under test. The features of the bigrams are the concatenated gate and net features with no modification. 2) The Bigrams for each subpath is fed to custom representation learning models that exploit the relationship between Bigram properties to generate complex latent features, Ψ that could be used in the next stage (as input) for regression. These representation learning models are means of automated feature engineering. 3) latent features generated by representation learning modules are fed to a Multi-Layer Perceptron (MLP) for regression. The MLP is used for regression and is trained using a labeled dataset to predict the PBA timing results. 4) to assist with label prediction, the output of each representation learning model is also fed to a shallow MLP for sub-path label regression to predict the delay of the Data, Launch and Capture portions of a timing path using a labeled training set. The sub-path prediction path only exists during the training stage to improve the model's accuracy and speed up the training and is removed at the inference stage. The whole RAPTA model is trained end-to-end using a labeled training set. The training set is generated by running the PnR flow once (without caring for timing violations) and generating PBA timing analysis results in Process, Voltage, and Temperature (PVT) corners of interest. The STA engine reported PBA slacks are used as the label, combined with raw EDA properties as input (bigram feature) streams for building a training set. Note that to generate training data, alternatively, the designer can collect the training data from a previously timing-closed design

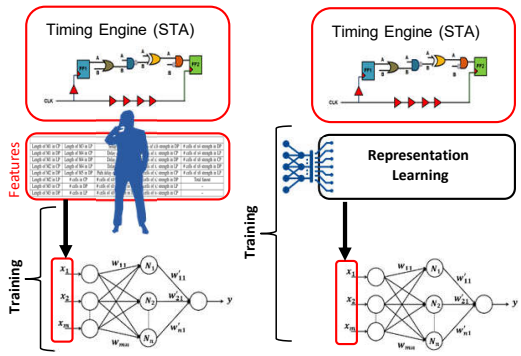

Figure 3: (left): Solution explored in previous work [8][4], (right): proposed representation Learning Solution. The proposed solution eliminates the need for the human in the loop for feature engineering and allows us to build and extract complex features unknown to the expert user. Additionally, using the representation learning solution removes the limitation on the input size. Unlike previous work, the representation learning solutions can take EDA features reported for timing path components as an input stream with no limitation. The novel feature of this solution is that the model size does not increase if the size of the input feature stream increases, addressing one of the big challenges in previous work.

that is signed off in the same technology with the same EDA and the same libraries.

The unique feature of RAPTA is its independence from input size. The model size is fixed, and if the number of Bigram (net and get) properties increases, the model size does not change. The bigram information (reported properties) is processed sequentially (similar to natural language processing). This eliminates one of the significant shortcomings of the previous work. The model does not need to change when the EDA version, number of gate or net properties, the library, or the process changes allowing us to have a feature-independent model size. The second unique feature of the RAPTA, as shown in Fig. 3, is the elimination of the need for feature engineering. RAPTA takes all raw bigram properties, and there is no need for a human expert to down-select the features or build/combine features as input to the model. The representation learning module automatically generates complex and meaningful latent features at its output and is fed to the subsequent regression layer. With this introduction, next we describe each of these steps. We illustrate the Bi-LSTM representation in detail in Section 3.2. Label prediction and Sub-Label Prediction are explained in Sections 3.3 and 3.4.

3.1 Modeling and Training set Preparation

Each data point in our training or test sets represents a timing path. We break each timing path into launch, data, and capture subpaths. The launch and capture subpaths include all gates and nets in the clock network from the clock source to the launch and capture registers, respectively. The data subpath consists of the launch and capture register and all gates and nets in between. Each gate and net are characterized by I and J features, respectively. These features are the raw properties reported by the EDA tool. We also collect timing-path level features, including the capture register's setup/hold time, launch register's clk2Q, Voltage, Temperature, and Process corner.

Let's assume P_D, P_L, and P_C represent the data, lunch, and capture subpaths in a timing path, respectively. We combine a gate and its proceeding net into a bigram cell. Hence, each subpath is

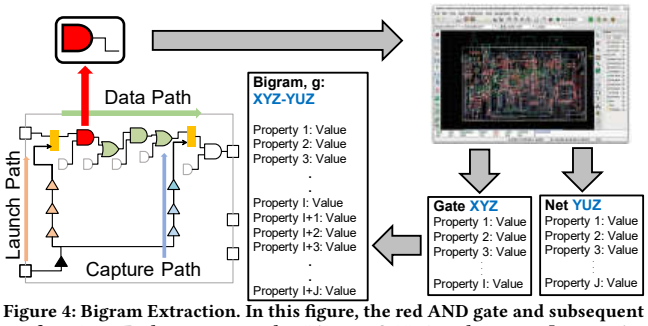

Figure 4: Bigram Extraction. In this figure, the red AND gate and subsequent net from Data Path are converted to Bigram, G. EDA tools extract I properties from that AND gate and J features from its succeeding net. All properties ($I + J$) together are used as properties for Bigram (G) for that specific AND gate of Data Path. The same procedure has been followed for the rest of the gates, registers for Data, Launch and Capture path.

modeled as a sequence of bigrams g containing $I + J$ features. The capture register has no proceeding net, and value zero is assigned for its net features. Different timing paths have different numbers of bigrams in their P_D, P_L and P_C subpaths. We denote the maximum size of bigrams sequence in each of P_D, P_L and P_C using the notation X_D, X_L, X_C, respectively. That is the size of the design's largest data, launch, and capture path.

Algorithm 1 Data Preparation (see Figure. 4)

$X_L \leftarrow$ size of largest Launch sub-path in the design
$X_C \leftarrow$ size of largest Capture sub-path in the design
$X_D \leftarrow$ size of largest Data sub-path in the design
for each timing path y in the set of timing paths selected for training **do**
  $P_D(y) \leftarrow$ Get_Bigram_Features $(y, Data, X_D)$ ▷ Algorithm 2
  $P_L(y) \leftarrow$ Get_Bigram_Features $(y, Launch, X_L)$ ▷ Algorithm 2
  $P_C(y) \leftarrow$ Get_Bigram_Features $(y, Capture, X_C)$ ▷ Algorithm 2
  $P_{reg}(y) = concat[c2q(y), T_{setup}(y)]$ ▷ path level timing info
  $P_{PVT}(y) = concat[V(y), P_{proc}, P_{temp}(y)]$ ▷ PVT info
  $F(y) = concat[P_D(y), P_L(y), P_C(y), P_{reg}(y), P_{PVT}(y)]$
  $T(y) \leftarrow$ PBA Slack of path y reported from STA ▷ label
  $T_{sub}(y) \leftarrow concat$[Delay(L), Delay(C), Delay(D)] ▷ sub-labels
end for

Algorithm 1 describes how the bigram information is collected for each timing path. The bigram stream of each of the data, launch, and capture paths has the size of X_D, X_L, and X_C, respectively. During training, for $batch\_size > 1$; if a subpath in a timing path has fewer gates (bigrams) than the maximum capacity of the model (e.g., size(data path) < X_D, or size(capture path) < X_C, or size(Launch path) < X_L), the unused bigrams features set to 0. For the launch and capture subpath, the bigrams are filled from the last cell/bigram to the first cell/bigram, and the zero padding (if needed) is applied to the first bigrams. In the data path, the first and last bigrams are assigned to the launch and capture register, then bigrams are assigned from the last combinational cell to the first combinational cell in the data path, and zero-padding is applied if the number of cells is smaller than the bigram size. Note that in this case, the zero-padding happens after the first bigram (which is assigned to the launch register). The reason for this type of padding is to expect the same bigrams to hold the data for launch and capture register across all timing paths, while zero padding is used to have the same data structure (of the same size) for all timing paths. The slack and delay information (main and sub-labels) are collected from STA simulation of the timing path for the selected timing paths in the training set. The design does not need to be optimized, and the first PnR attempt (out of the box without having to fix any timing violations) could be subjected to STA analysis in PBA mode to generate the training data. Alternatively, as described previously, the training set could be collected from a previously timing-closed design. The process of preparing the data set and generation of bigram data structure (to be used in our representation learning sub-module) is illustrated in Algorithm 1.

Algorithm 2 Collecting Bigram Features

function Get_Bigram_Features(timingPath y, subpath s, MaxSize X)
  $g \leftarrow$ empty bigram array of size X initialized to zero ▷ $g \in g$
  if $(subpath == Data)$ **then** ▷ Data sub-path
    $g_1 \leftarrow$ concat features of first register and proceeding net
    $g_X \leftarrow$ concat features of the last register
    for $(i = 1; i \leq size(s) - 2; i = i - 1)$ **do**
      $g_i \leftarrow$ concat feature of gate (i) and proceeding net
    end for
  else ▷ Lauch or Capture sub-path
    for $(i = 0; i \leq size(s); i = i - 1)$ **do**
      $g_i \leftarrow$ concat feature of gate (i) and proceeding net
    end for
  end if
  return g;
end function

3.2 Representation Learning

The sequential information evaluation over bigrams along subpaths (P_D, P_L, P_C) conveys useful information when being processed in both directions. This is because the drive strength of a cell affects several gates' slew rate down the path. Besides, a change in the capacitive load of a wire affects the delay of its preceding gate (to drive a larger CAP) and several stages before. Hence, in RAPTA, we deploy a BiLSTM network to learn from 1) feature information of each bigram and 2) feedback and feedforward information propagated from proceeding and preceding bigrams. Besides, the use of BiLSTM allows us to change the feature count in the bigram input stream of each LSTM cell, without affecting the network architecture as features are sequentially processed by LSTM cells.

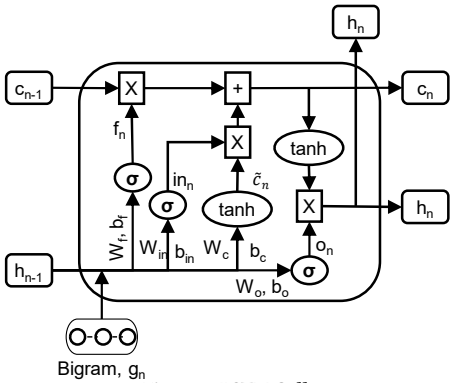

Figure 5: LSTM Cell

The basic building block of BiLSTM layer is LSTM cell and this cell has an input gate, a output gate and a forget gate. We considered a LSTM cell (Figure 5) for processing a bigram, g_n at n time steps. This cell also considers the previous cell's hidden state h_{n-1}, and cell state c_{n-1}. Each gate uses its own weight and bias shown in the figure. The input gate (in_n), forget gate (f_n), output gate (o_n) values are calculated by using a sigmoid layer (σ), previous cell's hidden state (h_{n-1}) and bigram features (g_n) in Equations (1) to (3). Hyperbolic tangent ($tanh$) is used instead of sigmoid layer (σ) to calculate the new candidate ($\tilde{c}$) for replacing the memory cell in

Equation (4),

$$in_n = \sigma(W_{in} \cdot [h_{n-1}, g_n] + b_{in}) \quad (1)$$

$$f_n = \sigma(W_f \cdot [h_{n-1}, g_n] + b_f) \quad (2)$$

$$o_n = \sigma(W_o \cdot [h_{n-1}, g_n] + b_o) \quad (3)$$

$$\widetilde{c}_n = tanh(W_c \cdot [h_{n-1}, g_n] + b_c) \quad (4)$$

With updated gates and candidate values, the LSTM cell uses previous cell state c_{n-1} to update the current cell state c_n and hidden state h_n in Equations (5) and (6). For each time step in an LSTM layer h_n becomes the output of that step.

$$c_n = f_n * c_{n-1} + in_n * \widetilde{c}_n \quad (5)$$

$$h_n = o_n * tanh(c_n) \quad (6)$$

If n becomes the last cell of the sequence ($n = X$ in our case), then h_n (or h_X) can be presented as a context vector for the whole sequence. In the case of the BiLSTM layer, there are two outputs for each time step; one is forward output, h_n^f and another is backward output, h_n^b. So, for each time step, the output would be the concatenation of these two outputs. The sequence processing of BiLSTM has been depicted in Figure 6.

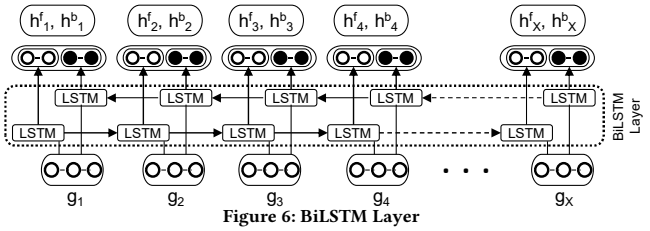

Figure 6: BiLSTM Layer

.

If there is more than one BiLSTM layer, these concatenated outputs would be used as input for the next layer cells at the respective time step. If there are w layers, then the outputs from the last cell of the last forward layer (h_X^{wf}) and the first cell of the last backward layer (h_1^{wb}) would be the context vector for the whole sequence in forward and reverse directions. By concatenating these two context vectors, we get the representation of the entire sub-path. These concatenated context vectors are then sent to a fully connected layer to get the final representation of the whole sequence in Equation (7). Here y is for the timing path index when we use multiple timing paths as samples for training the model.

$$\Psi^s(y) = \phi(concat[h_X^{wf}, h_1^{wb}]) \quad (7)$$

The whole process is shown in Figure 7. The sub-path is a sequence of bigrams. So, in each layer, two LSTM cells are used recurrently until all the bigrams are processed in the forward and reverse directions. From the bottom, the first BiLSTM layer serves the purpose of a single BiLSTM layer. Each cell output is concatenated and fed into the next BiLSTM Layer, and the procedure is followed up to the last layer w. In our experimental setting, we used $w = 3$ layers to construct our representation learning model. Using larger than $w = 3$ has a diminishing return, and the added complexity did not improve our representation learning accuracy. We put the values of h_X^{wf} and h_1^{wb} in Equation (7) and found the final representation of sub-path. This procedure can be followed for data path P_D, launch path P_L and capture path P_C and it will yield

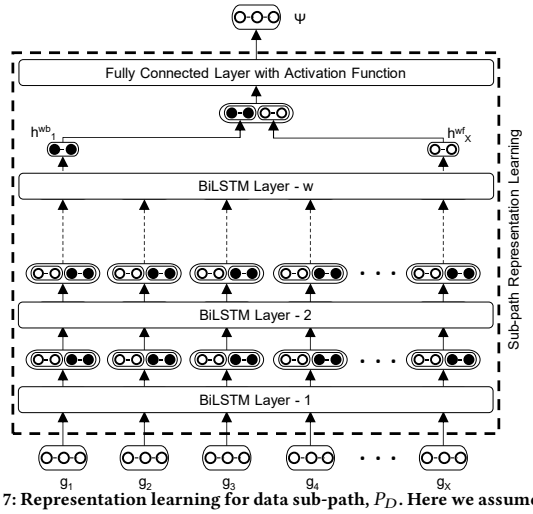

Figure 7: Representation learning for data sub-path, P_D. Here we assume there are six bigrams in the data path and the number of layers, $w = 3$ for stacked BiLSTM.

Ψ^D, Ψ^L and Ψ^C as the representation of data, launch, and capture sub-path respectively.

3.3 Main Regression Model

A subpath BiLSTM representation, $\Psi^s(y)$, is derived from Equation (7) where the first and the last output of the each BiLSTM network are concatenated and pass through an activation function to generate the latent features. The subpath BiLSTM representations of $\Psi^L(y)$, $\Psi^C(y)$, and $\Psi^D(y)$ are concatenated together in Equation (8) along with timing path's PVT information ($P_{PVT}(y)$) and timing path's register timing information ($P_{reg}(y)$) that are previously collected in Alg. 1. $M(y)$ was used as input for the main MLP module for the timing path (y) slack prediction.

$$M(y) = concat[\Psi^L(y), \Psi^D(y), \Psi^C(y), P_{reg}(y), P_{PVT}(y)] \quad (8)$$

The operation of the main MLP network is captured by Equations (9) and (10) where $l1$ AND $l2$ are the FC layer order, $\gamma(y)$ is the output of the first FC and the label (slack) prediction is the output of the second FC layer.

$$\gamma(y) = \phi(M(y) \cdot W_{l1} + b_{l1}) \quad (9)$$

$$\widehat{T(y)} = \phi(dropout(\gamma(y), p) \cdot W_{l2} + b_{l2}) \quad (10)$$

3.4 Sub-Branch Regression Models

RAPTA also embeds three sub-branches for subpath delay prediction, loss of which is computed using subpaths' sub-labels $T_{sub}(y)$, collected in Alg. 1. Note that each MLP receives only the sub-label for that sub-branch (e.g., launch path get $T_{sub}(y, 1)$, capture path get $T_{sub}(y, 2)$, and data path get $T_{sub}(y, 3)$). These sub-branches are only used during the training to facilitate the back-propagation and are removed at test time. $\Psi^s(y)$ is combined with PVT information in equation 11 of the subpath, forming the input to each of three MLPs in RAPTA's sub-branches for the subpath delay prediction.

$$M^s(y) = concat[\Psi^s(y), P_{PVT}(y)] \quad (11)$$

The operation of MLP in sub-branches is described in Equations (12). In this equation, W^s and b^s are the weight and bias of FC layer used for prediction of subpath s, where s represent launch, capture or data subpath ($s \in \{L, C, D\}$).

$$\widehat{T^s(y)} = \phi(M^s(y) \cdot W^s + b^s) \tag{12}$$

3.5 Other Functions

We used a multi-level mean square error (MSE) loss calculation (in equation 13) to create a loss matrix and used adam optimizer[10] with ℓ_2 regularization for the training of this model.

$$MSE = \frac{1}{q}\sum_{t=1}^{q} e_t^2 \tag{13}$$

A common problem with backpropagation is vanishing gradients in the result of which the model training fails. To remedy this problem, we adopted standardization instead of normalizing our data. To address this problem for batch normalization and internal normalization of neural networks, used Scaled Exponential Linear Units (SELUs) [11] as activation function to avoid the vanishing gradients problem with training as in Equation 14.

$$f(\alpha, r) = \lambda \begin{cases} \alpha(e^r - 1) & \text{for } r < 0 \\ r & \text{for } r \geq 0 \end{cases} \tag{14}$$

3.6 End-to-End Training of RAPTA

RAPTA's learning process is described in Algorithm 3, summarizing the model architecture and the data flow described in this section. In this algorithm, line 1 to line 8 is related to the data processing as explained in section 3.1. The Bi-LSTM representation has been formalized in lines 11 to line 13. The label prediction procedure is contained in lines 9 to line 18. And line 19 to line 25 describe the sub-label prediction.

4 EXPERIMENT

For this study, we compared the performance of RAPTA against prior-art solutions for predicting the PBA slack of the three largest IWLS benchmarks[1] during physical design.

Algorithm 3 RAPTA Framework

1: Collecting data ▷ Algorithm 1

Phase 1 – Representation Learning Phase

2: **for** each timing path y **do**
3: **for** each subpath s in (L, D, C) **do**
4: **for** each bigram n from 1 to N = X_s **do**
5: $h_n^s(y) \leftarrow g_n^s(y)$ ▷ assign each bigram to an LSTM cell
6: **end for**
7: $\Psi^s(y) \leftarrow \varphi(concat[h_1^{wb,s}(y), h_X^{wf,s}(y)]$ ▷ subpath representation
8: $M^s(y) \leftarrow concat[\Psi^s(y), PVT(y)]$ ▷ Input to subpath FC
9: **end for**
10: $M(y) \leftarrow concat[\Psi^D(y), \Psi^L(y), \Psi^C(y), P_{reg}(y), P_{PVT}(y)]$ ▷ input to main FC network for slack prediction
11: **end for**

Phase 2 – Label and Sub-label Prediction Phase

12: **for** each timing path y **do**
13: **for** each subpath s in {L, D, C} **do**
14: $\widehat{T^s(y)} \leftarrow \varphi(M^s(y) \cdot W^s + b^s)$ ▷ subpath prediction
15: **end for**
16: $\gamma(y) \leftarrow \varphi(M(y) \cdot W_{l1} + b_{l1})$ ▷ first FC layer
17: $\widehat{T(y)} \leftarrow \varphi(\text{dropout}(\gamma(y), p) \cdot W_{l2} + b_{l2})$ ▷ dropout and 2nd FC layer
18: **end for**

$X_s \in \{X_L, X_D, X_C\}$ represent the maximum number of LSTM cells used to represent launch, data, or capture subpaths in our model.

4.1 Experimental Setup:

We used Synopsys IC Compiler (ICC) for the physical design of benchmarks in 32nm technology available under Synopsys educational license. We used the ICC timing report to extract a set of timing paths and queried ICC to extract the gate and net properties. We used one-hot encoding to represent the value of non-numerical gate properties, such as "gate type" or "threshold voltage". Synopsys Primetime was then launched for path-based timing analysis, and the delay of each subpath (P_D, P_C, and P_L) was collected as labels and sub-labels for each timing path, respectively. The data was standardized for all experiments and was split by 60, 20, and 20% into training, cross-validation, and test sets.

We used Pytorch [15] as a learning framework for developing our proposed model. We used Adam optimizer [10], with a learning rate of 1×10^{-3} and batch size of 128. The value of p for Equation (9) was 1×10^{-4}, ℓ_2-norm regularization was conducted for $\beta = 1\times10^{-4}$. For activation function we used 'Selu' [11]. For the rest of the models, we employed Sklearn [16] as the model framework. Each model was trained by the tuned hyperparameter stated in Table 1. All the experiments of our proposed model were conducted on a 64-bit machine with Intel(R) Xeon(R) W-2123 CPU 3.60GHz processor and 32GB memory and NVIDIA TITAN RTX GPU.

4.2 Models used for comparison

To evaluate the performance of our proposed solution, we compared it against several learning algorithms. This section describes each of the learning solutions selected and used for comparison.

Linear Regression: Linear regression models are formulated based on Equation (15). In this equation, $F(y)$, W_l, and b_l represent the input feature set (Algorithm 1), model weights, and the bias, respectively. We evaluated three linear regression models, namely Lasso, Ridge, and Elastic net (Enet). Lasso uses ℓ_1 regularization, Ridge uses ℓ_2 regularization and Enet uses both ℓ_1 and ℓ_2 regularization [14] to minimize the MSE error.

$$\widehat{T(y)} = F(y) \cdot W_l + b_l \tag{15}$$

Table 1: Model Parameters used for RAPTA and other Comparison models.

Models	Hyper-Parameters
RAPTA	optimizer=Adam, learning rate = 0.001, batch size = 128, dropout = 0.0001, ℓ_2 regularization = 0.00001. input dimension = 65, activation function = 'Selu'
Ridge	max_iter=1000, tol=0.00001, alpha=10, solver=sag
Lasso	max_iter=1000, tol=0.00001, alpha=0.0001
Enet	max_iter=1000, tol=0.00001, alpha=0.0001, li_ratio=1.5
RF	min_samples_split=2, min_samples_leaf=1, bootstrap=TRUE, n_estimators=2048
MLP	max_iter=2000, tol=0.0001, alpha=0.0005, solver=lbfgs, learning_rate=adaptive, hidden_layer_sizes=50, 100, activation=logistic
Stacking	regressors=Ridge, Lasso, Enet, MLP, RF, meta_regressor=RF, verbose=1, n_jobs=2, use_features_in_secondary=True

Multilayer Perceptron (MLP): MLP is a non-linear model that could be described by Equation (16) to (17). It uses an activation function, σ, to map the input features into a latent space [5]. The latent space features are then linearly separated. MLP has one input layer, one or more hidden layer $\gamma_h(y)$, and one neuron in the output

Table 2: Model complexity comparison

Models	Ridge	Lasso	Enet	RF	MLP	Stacking	RAPTA
MFLOPS	1.38	1.38	1.38	22,546.68	142.15	45,239.64	2,332.62
Params	1,659	1,659	1,659	3,318	171,001	182,614	353,973

layer for a regression problem. In each layer, there are weight W and bias b with different layer orders $l1$, $l2$.

$$\gamma_h(y) = \sigma(F(y) \cdot W_{l1} + b_{l1}) \tag{16}$$

$$\widehat{T(y)} = \sigma(\gamma_h(y) \cdot W_{l2} + b_{l2}) \tag{17}$$

Random Forest (RF): RF is an implementation of bagging techniques where several models are used to come to the final decision. The basic component of Random Forest is a decision tree [6]. RF uses MSE to measure the distance of each node from the predicted actual values, allowing RF to decide which branch yields a better decision.

Stacking Model: The stacking learning model is a 2-level ensemble of regressors; each model in the ensemble can be used separately as a prediction model; however, by stacking, a more generalized model is obtained [23]. For comparison, we deployed and trained a regression stacking model described in [3]. The architectural detail of the model is described in Table 1.

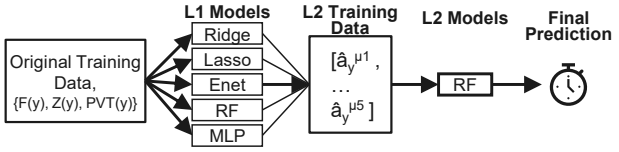

Figure 8: A general view of the used two-layer stacked model.

4.3 Results:

Table 2 reports the complexity of RAPTA and prior-art solutions in terms of required FLOPS and parameter count. As reflected in this table, RAPTA has the highest parameter count. But, in terms of computational complexity (expressed in MFLOPS), RAPTA is less complex than RF and the Stacking model.

We have evaluated our proposed model's accuracy and compared it against prior art solutions in two settings: 1) when training and testing the model on a single PVT corner, 2) when training the model on multiple PVT corners and using it for interpolative PBA prediction. These settings and our experimental results are described next:

4.3.1 ***Training and Testing on Single PVT Corner.*** We evaluated the accuracy of the model in predicting the PBA slack when the model is trained and tested in the same PVT corner. The result of these experiments for RAPTA and prior-art models is illustrated in Fig. 9. In this experiment, the temperature is 125C, the process is nominal, and the experiment is repeated for different voltages, from 0.78v to 1.05v. As illustrated, RAPTA outperforms all prior models in terms of accuracy; the regression result has an error with a standard deviation in the range of 3.90 to 16.05ps. Note that in lower voltages, the delay of the timing paths is longer. Hence, although the standard deviation of regression error is slightly larger in lower voltages, it remains roughly the same percentage of the total path delay.

4.3.2 ***Interpolative PBA prediction.*** The most exciting use-case of the RAPTA is for interpolative PBA timing-slack prediction. In this mode, the model is trained on a limited number of PVT corners and is used to predict the PBA timing results in both seen and unseen timing corners.

Table 3: Average standard deviation of error of RAPTA and prior-art learning models, reported in picoseconds (ps) for training and testing on the largest IWLS benchmarks.

	Models	Avg standard deviation of error (ps) Number of PVT corners used for training the model					
		Single	Two	Three	Five	Eight	All
S38417	Ridge	17.48	63.98	62.16	61.77	60.58	62.67
	Lasso	17.29	73.05	61.47	55.82	54.79	54.51
	Enet	16.79	68.81	56.51	55.57	54.86	54.98
	RF	18.15	38.34	36.93	25.30	20.65	17.19
	MLP	13.02	56.61	21.85	18.72	15.78	16.96
	Stacking	10.07	46.96	22.80	16.54	17.73	15.74
	RAPTA	**8.47**	**36.14**	**13.97**	**9.37**	**8.33**	**8.36**
AES128	Ridge	19.76	53.61	51.28	51.34	50.96	52.83
	Lasso	19.21	59.25	52.48	51.07	50.20	50.42
	Enet	19.36	56.85	51.39	50.75	50.25	50.45
	RF	12.78	59.90	31.05	21.54	16.35	12.68
	MLP	16.38	46.06	20.51	20.81	20.86	18.08
	Stacking	**11.80**	53.59	21.52	19.59	18.38	14.46
	RAPTA	11.98	**37.05**	**19.81**	**12.41**	**12.69**	**12.31**
Ethernet	Ridge	28.21	79.85	77.56	77.14	75.48	74.16
	Lasso	26.99	63.60	64.52	66.29	65.58	67.02
	Enet	26.86	73.76	76.88	72.50	67.33	66.13
	RF	8.15	70.04	35.53	21.78	14.17	7.78
	MLP	12.39	68.87	30.31	18.92	19.79	21.57
	Stacking	6.94	66.98	34.69	20.92	9.33	7.06
	RAPTA	**5.12**	**35.88**	**14.62**	**10.69**	**6.34**	**6.03**

Table 3 summarizes the accuracy of RAPTA and prior-art learning solutions for interpolative PBA prediction. The number of selected corners for training the models is varied in the range of 2, 3, 5, 8, and 16 corners. In this figure, the temperature and process are fixed at 125C Temperature and nominal Process, respectively. The selected corners for each multi-corner training scenario are those with the largest Euclidean distance between voltages among the range of corners. For example, the high, mid, and low voltage corners are used in the three-corner training. Across all experiments, RAPTA yields the lowest regression error. The exception is for the single-corner experiment for benchmark AES128, where the stacking model generates slightly better prediction, but when used in multi-corner interpolative PBA prediction mode, RAPTA outperforms the stacking model. As illustrated, the range of change in the standard deviation of the error reduces as the number of corner cases included in the training process increases. In our experiment, depending on the benchmark, by having 5 to 8 corners in training, we get similar performance as single corner training.

The time needed for training and testing RAPTA is reported in table 4. As reported, the time required for training the model is relatively short compared to the design cycle time. The training is done only once in the early design cycle. The reported number for testing is the time needed to generate PBA prediction for 10K timing paths. As illustrated, RAPTA can be used to get instant feedback on the timing impact of physical design changes (without having to re-run STA) and is a very effective tool for "what if" analysis. The difference between the PBA-predicted slack and GBA slack suggested by the PnR tool could be back annotated in the PnR tool, helping it in making better PPA optimization decisions and avoiding overdesign to reduce design cycle time.

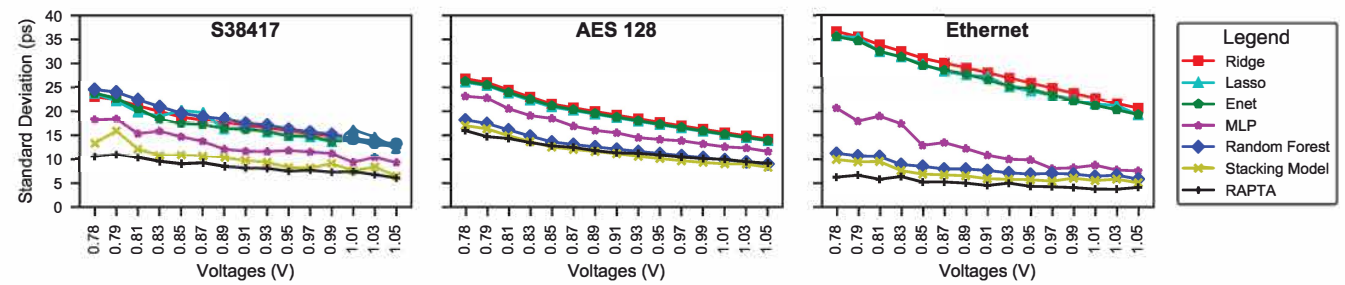

Figure 9: Models performance comparison in a single corner: RAPTA outperformed all the models in single corner experiments for S38417 and Ethernet benchmark. The range of standard deviation of error from RAPTA for S38417 and Ethernet are 6.08–10.88ps, 3.90ps–6.80ps, which are on average 15.89%, 26.29% improvement from its closest competitor stacking model. In AES128, the stacking model and RAPTA performed in a similar range. The average standard deviation of errors was 11.80ps and 11.98ps for the stacking model and RAPTA.

Table 4: RAPTA average train and test time on GPU. The reported number for the test is the time needed to generate PBA prediction for 10K timing paths. The training is only done once during the design cycle. The short test time enables the physical designer to get real-time feedback and perform what-if analysis at a low cost.

Metric	Benchmark	Number of PVT corners used for training the model					
		Single	Two	Three	Five	Eight	All
Train time (Hr)	S38417	0.132	0.243	0.155	0.958	2.354	2.727
	AES128	1.146	1.132	2.711	14.299	11.872	16.911
	Ethernet	1.815	2.454	4.973	20.814	58.013	59.724
Test time (Sec)	S38417	0.27	0.26	0.26	0.28	0.27	0.27
	AES128	0.20	0.17	0.17	0.17	0.17	0.17
	Ethernet	0.20	0.18	0.18	0.18	0.18	0.18

5 CONCLUSION

This paper presented RAPTA, a novel representation-learning Architecture for Path-based timing slack prediction. RAPTA does not need explicit feature engineering and takes as input the raw gate and net properties reported by the PnR tool. Our experimental result verifies that RAPTA can generate PBA slack prediction with an average standard deviation of the error in the range of 5 to 12 ps in 32nm technology, which outperforms prior-art by a considerable margin. RAPTA architecture is adaptive to variable path size and feature-set size resulting from the novel representation-learning solution is utilized to process timing path features. The high accuracy of RAPTA makes it a perfect companion for PnR tools, giving the physical designer a glance of PBA-accurate timing prediction on any desired group of timing paths without having to run a full Static Timing Analysis.

REFERENCES

[1] 2005. IWLS Benchmarks. http://iwls.org/iwls2005/benchmarks.html.

[2] Sayed Aresh Beheshti-Shirazi, Ashkan Vakil, Sai Manoj, Ioannis Savidis, Houman Homayoun, and Avesta Sasan. 2021. *A Reinforced Learning Solution for Clock Skew Engineering to Reduce Peak Current and IR Drop.* Association for Computing Machinery, New York, NY, USA, 181–187. https://doi.org/10.1145/3453688.3461754

[3] Leo Breiman. 1996. Stacked regressions. *Machine learning* 24, 1 (1996), 49–64.

[4] Peng Cao, Wei Bao, and Jingjing Guo. 2020. An Accurate and Efficient Timing Prediction Framework for Wide Supply Voltage Design Based on Learning Method. *Electronics* 9, 4 (2020), 580.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction.* Springer Science & Business Media.

[6] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd Int. Conf. on document analysis and recognition*, Vol. 1. IEEE, 278–282.

[7] Andrew B. Kahng. 2018. Machine Learning Applications in Physical Design: Recent Results and Directions. In *ISPD*. Association for Computing Machinery, 68–73. https://doi.org/10.1145/3177540.3177554

[8] Andrew B Kahng, Uday Mallappa, and Lawrence Saul. 2018. Using machine learning to predict path-based slack from graph-based timing analysis. In *ICCD*. IEEE, 603–612.

[9] Andrew B Kahng, Uday Mallappa, Lawrence Saul, and Shangyuan Tong. 2019. " Unobserved Corner" Prediction: Reducing Timing Analysis Effort for Faster Design Convergence in Advanced-Node Design. In *DATE*. IEEE, 168–173.

[10] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[11] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In *Advances in neural information processing systems*. 971–980.

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

[13] R Molina. 2013. *EDA Vendors Should Improve the Runtime Performance of PathBased Timing Analysis.* https://www.electronicdesign.com/technologies/eda/article/21796368/eda-vendors-should-improve-the-runtime-performance-of-pathbased-analysis

[14] Joseph O Ogutu, Torben Schulz-Streeck, and Hans-Peter Piepho. 2012. Genomic selection using regularized linear regression models: ridge regression, lasso, elastic net and their extensions. In *BMC proceedings*, Vol. 6. Springer, S10.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS Proceedings*. 8026–8037.

[16] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research* 12, Oct (2011), 2825–2830.

[17] M Amin Savari and Hadi Jahanirad. 2020. NN-SSTA: A deep neural network approach for statistical static timing analysis. *Expert Systems with Applications* 149 (2020), 113309.

[18] Hao-Chiang Shao, Chao-Yi Peng, Jun-Rei Wu, Chia-Wen Lin, Shao-Yun Fang, Pin-Yen Tsai, and Yan-Hsiu Liu. 2020. From IC Layout to Die Photo: A CNN-Based Data-Driven Approach. *arXiv preprint arXiv:2002.04967* (2020).

[19] Ashkan Vakil, Farnaz Behnia, Ali Mirzaeian, Houman Homayoun, Naghmeh Karimi, and Avesta Sasan. 2020. LASCA: Learning Assisted Side Channel Delay Analysis for Hardware Trojan Detection. In *2020 21st International Symposium on Quality Electronic Design (ISQED)*. 40–45. https://doi.org/10.1109/ISQED48828.2020.9137007

[20] Ashkan Vakil, Ali Mirzaeian, Houman Homayoun, Naghmeh Karimi, and Avesta Sasan. 2021. AVATAR: NN-Assisted Variation Aware Timing Analysis and Reporting for Hardware Trojan Detection. *IEEE Access* 9 (2021), 92881–92900. https://doi.org/10.1109/ACCESS.2021.3093160

[21] Ashkan Vakil, Farzad Niknia, Ali Mirzaeian, Avesta Sasan, and Naghmeh Karimi. 2021. Learning Assisted Side Channel Delay Test for Detection of Recycled ICs. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference* (Tokyo, Japan) *(ASPDAC '21)*. Association for Computing Machinery, New York, NY, USA, 455–462. https://doi.org/10.1145/3394885.3431640

[22] Han Wang, Hossein Sayadi, Sai Manoj Pudukotai Dinakarrao, Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2021. Enabling Micro AI for Securing Edge Devices at Hardware Level. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11, 4 (2021), 803–815. https://doi.org/10.1109/JETCAS.2021.3126816

[23] David H Wolpert. 1992. Stacked generalization. *Neural networks* 5, 2 (1992), 241–259.

[24] Zhiyao Xie, Yu-Hung Huang, Guan-Qi Fang, Haoxing Ren, Shao-Yun Fang, Yiran Chen, and Jiang Hu. 2018. RouteNet: Routability prediction for mixed-size designs using convolutional neural network. In *ICCAD*. IEEE, 1–8.

[25] Bei Yu, David Z Pan, Tetsuaki Matsunawa, and Xuan Zeng. 2015. Machine learning and pattern matching in physical design. In *ASPDAC*. IEEE, 286–293.