"Unobserved Corner" Prediction: Reducing Timing Analysis Effort for Faster Design Convergence in Advanced-Node Design

Andrew B. Kahng[†‡], Uday Mallappa[‡], Lawrence Saul[†] and Shangyuan Tong[†]
[†]CSE and [‡]ECE Departments, UC San Diego, La Jolla, CA, USA
{abk, umallapp, saul, s8tong}@ucsd.edu

***Abstract*—With diminishing margins for leading-edge products in advanced technology nodes, design closure and accuracy of timing analysis have emerged as serious concerns. A significant portion of design turnaround time is spent on timing analysis at combinations of process, voltage and temperature (PVT) *corners*. At the same time, accurate, signoff-quality timing analysis is desired during place-and-route and optimization steps, to avoid loops in the flow as well as overdesign that wastes area and power. We observe that timing results for a given path at different corners will have strong correlations, if only as a consequence of physics of devices and interconnects. We investigate a data-driven approach, based on *multivariate linear regression*, to predict the timing analysis at unobserved corners from analysis results at observed corners. We use a simple backward stepwise selection strategy to choose which corners to observe and which to predict. In order to accelerate convergence of the design process, the model must yield predicted values (from analysis at a limited number of observed corners) that are sufficiently accurate to substitute for unobserved ones. Our empirical results indicate that this is likely the case. With a 1M-instance example in foundry 16nm enablement, we obtain a model based on 10 observed corners that predicts timing results at the remaining 48 unobserved corners with less than 0.5% relative root mean squared error, and 99% of the model's relative prediction errors are less than 0.6%.**

I. Introduction

In advanced-node IC design, EDA tools and flows are confronted today by multi-million-instance designs and signoff requirements at hundreds of timing scenarios. A substantial amount of tool runtime, as well as overall compute and license resource, must be spent to obtain analysis of timing across multiple process, voltage and temperature (PVT) analysis *corners*. All else being equal, design teams would like to always have accurate analyses at all signoff corners: this helps avoid expensive loops in the place-and-route and optimization steps of the flow, and also helps avoid overdesign that wastes area and power. However, as a practical matter designers can analyze timing at only a small number of corners during most of the design steps leading up to final signoff. This potentially masks many real violations, since PVT effects on signal arrival time at a given timing endpoint will depend on the *combination* of gates and wires in any given timing path. To reduce the risk of masking real violations that are realized only in final timing signoff runs, designers add flat timing margins or increase the target clock period. However, there is no canonical methodology of determining the best mix of analyzed corners, padding by flat margins, relaxation of frequency targets, etc. – all of which eventually result in overdesign.

If golden timing analysis across all signoff corners were available at low overhead, designers would be able to reduce overdesign, and detect and fix real timing violations much earlier in the design cycle. Our present work pursues a data-driven approach to this challenge of reducing timing analysis effort in advanced-node IC design, with the aim of enabling faster design convergence. A motivating observation is that path delays at different corners are strongly correlated, with correlations dependent on the topology and structure of the timing path. Our work seeks to accurately capture and exploit this correlation, so as to improve design convergence and reduce schedule cost.

Our main result shows the feasibility of machine learning model-based prediction of timing results at many *unknown* corners, based on timing results from comparatively few *known* corners. Based on an analysis of timing path reports taken across all signoff PVT corners, we apply greedy deletion (i.e., backward elimination) to determine a high-quality set of *known* corners at which timing needs to be analyzed using a golden tool. While we have explored multiple machine learning models to predict timing results at the remaining *unknown* PVT corners, a simple multivariate linear regression model produces the best results and is the focus of our discussion below.

II. Background and Preliminaries

Table I defines notations that we use below. We use N to denote the total number of analysis corners and $n < N$ to denote the number of corners whose path delay values are *known*. We denote the set of known corners by $\{T_{known}\}$. Our goal is to accurately predict $N-n$ **unknown** path delay values from n **known** path delay values. We use $\{T_{unknown}\}$ to denote the set of unknown corners.

As seen in Figure 1, timing results can be viewed as elements in a large matrix whose rows represent timing paths and whose columns represent corners.[1] We use R_{train} and R_{test} to denote the numbers of timing paths that are used, respectively, in training and testing (i.e., inference) of our model. Thus, the entire top portion of the matrix represents the *training data* that is used to train a predictive model. We refer to this matrix of training data as M_1.

A trained predictive model is used to infer the unknown elements in the bottom, *testing* (or, *inference*) matrix, which we refer to as M_2. Finally, we use X_{train} and X_{test} to respectively denote the matrix blocks of known corners that serve as inputs to our predictive model, and we use Y_{train} and Y_{test} to respectively denote the matrix blocks of unknown corners that our model is designed to predict.

Problem Statement. We formally state our problem, which is a form of *matrix completion*, as follows.

Given: (i) a complete matrix M_1 split into n column vectors $\{X_{train}\}_1^n$ and $N-n$ column vectors $\{Y_{train}\}_n^N$, and

[1] Our discussion will generally use the terms *corner* and *column* interchangeably. We also use both *testing* and *inference* to refer to evaluation of model accuracy on a set of unknown data.

978-3-9819263-2-3/DATE19/©2019 EDAA

TABLE I
TERMS AND DEFINITIONS.

Term	Definition
N	Total number of columns in the matrix (= analysis corners)
n	Number of known (i.e., analyzed) columns
$N-n$	Number of unknown (i.e., to be predicted) columns
$\{T_{known}\}$	Set of known columns
$\{T_{unknown}\}$	Set of unknown columns
R	Total number of rows in the matrix (= timing paths)
R_{train}	Total number of complete rows (training)
R_{test}	Total number of rows with missing columns (testing/inference)
X_{train}	Matrix of known columns, used in model training
X_{test}	Matrix of known columns, used in model testing/inference
Y_{train}	Matrix of known columns, used in model training
Y_{test}	Matrix of unknown columns, used in model testing/inference
W	Model weight vector output by model training
M_1	Matrix of size $R_{train} \times N$
M_2	Matrix of size $R_{test} \times N$
LRM	Linear regression model

containing R_{train} rows of timing paths, along with **(ii)** an incomplete matrix M_2 split into n known column vectors $\{X_{test}\}_1^n$ and $N-n$ unknown column vectors $\{Y_{test}\}_n^N$, containing R_{test} rows of timing paths.
Use: $\{X_{train}\}_1^n$ and $\{Y_{train}\}_n^N$ to learn a model that can predict the unknown $N-n$ columns $\{Y_{test}\}_n^N$ of M_2 from the known n columns $\{X_{test}\}_1^n$ of M_2.

This is a problem in multivariate regression. The simplest models for this purpose, which we report on in this paper, are linear regressors that attempt to minimize the mean squared error of predicted values (even as other metrics may be more meaningful for real-world evaluation).

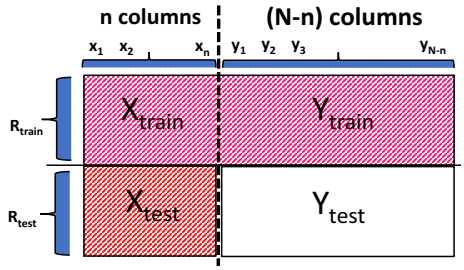

Fig. 1. Visualization of timing prediction as a problem in matrix completion.

Intuition for Multivariate Linear Regression. Our overall approach is based on the premise that the path delay values at different analysis corners are strongly correlated. Intuitively, such correlations are a consequence of the underlying physics of devices, interconnects, and signal delay propagation along timing paths. Though it may be difficult to model the detailed physics that produce these correlations, we can measure and exploit these correlations to predict large numbers of unknown path delay values from smaller numbers of known ones. A useful exercise is to imagine the path delay values at different analysis corners as the coordinates of points in a high-dimensional space. Repeated analyses of these path values generate empirical distributions over this space; when we say that these values are strongly correlated, we mean that these distributions of points are far from uniform. Indeed, as we show later, the main support of these distributions is heavily concentrated in a much lower dimensional subspace. An equivalent observation is that the large data matrix, shown in Fig 1, can be very well approximated by a matrix of much lower rank: i.e., many of the columns of this matrix are either linearly dependent or very nearly so. For this reason, we might expect even simple linear methods in multivariate regression to excel at the task of predicting certain columns from collections of others. This is the hypothesis we investigate in this paper.
Related Work. STA accuracy and runtime efficiency are a long-time focus of EDA R&D; see, e.g., the TAU Workshop [6], [15] and [17]. Methods that determine a smaller subset of analysis corners, so as to improve STA runtimes, have been actively pursued. [11] focuses on hold time analysis and determining a minimum set of *dominant corners* (whose satisfaction will result in timing feasibility at all other corners); an additional *dominance margin* is used to reduce the size of this dominance set. [13] gives a branch-and-bound methodology to identify a *single* corner that has worst delay. To avoid the analysis corner explosion entirely, [12] proposes what is effectively a "cornerless" approach that uses a single run of STA, with propagation of delay and slew models that are linear functions of process parameters, to cover all process corners. A large literature (e.g., [14]) has investigated statistical STA, which can effectively mitigate both corner explosion and the increasingly dominant impacts of manufacturing variability and low-voltage operating modes. Last, recent works such as [21] highlight the potential use of machine learning (ML) to achieve faster design convergence. Learning-based STA prediction has been studied by [16] and [22]; the latter uses linear regression, SVM and random forest models to account for dynamic NBTI aging and other correlated on-chip variations. [18] [19] [20] use ML to reduce STA miscorrelations.

In contrast, our present work seeks to explicitly predict path delays at unknown corners based on very small sets of known (analyzed) corners. Also, we do not focus on guaranteeing timing correctness (positive worst slack): since there are always some timing violations throughout the pre-tapeout IC design process, our goal is to achieve acceptable error distribution in model testing.[2] Last, while previous works offer insights for older technologies, we report results across enablements that include sub-14nm nodes, where schedule and convergence impacts of signoff corner explosion are prominent today.

III. MODELING METHODOLOGY

In this section, we describe our three-phase modeling procedure: subset selection, training, and testing.

(i) Subset Selection. For a fixed value of n, the problem of *subset selection* is to determine which n corners are most predictive of the remaining $N-n$ corners. For even moderately large values of n and N, the number of possible subsets is prohibitively large to perform an exhaustive search. A common approach is to adopt a greedy strategy [1]; for our problem we use the simple strategy of *greedy deletion* (also known as *backward elimination* [10]).[3] The outcome of this procedure is

[2]As mentioned above, we assume that there is a benefit from comprehending delays at all corners, particularly during timing closure and optimization steps that are prone to "ping-pong" effects.

[3]Our separate studies indicate that greedy *addition* results in worse model quality, especially for the small values of n that are of interest.

an array sequence $[predict\_corners]$ of length $N-1$ whose order tells us (for any value of n) which n corners should be used to predict the remaining $N-n$ corners.

(ii) Model Training. Once n and corresponding $\{T_{known}\}$ are determined, the *training* phase finds model parameters $W^* = \text{argmin}_W ||Y_{train} - X_{train} * W||_2^2$ for each n, such that the mean squared error of predictions is minimum. This is a one-time training investment.

(iii) Model Testing/Inference. The training phase generates optimal model parameters W^* for each value of n. Each model parameter set W^* has an associated error value. The cardinality of $\{T_{known}\}$ is chosen using an error budget that the user finds tolerable. The subset $\{T_{known}\}$ is then derived from the $[predict\_corners]$ array found by greedy deletion. Using the corresponding W^*, the model predicts timing at unobserved corners $\{T_{unknown}\}$ from observed corners $\{T_{known}\}$ using $Y_{test} = X_{test} * W^*$.

Modeling Flow. We study the modeling flow in Figure 2. We first determine optimal model parameters W^* for each value of n. The Inference phase predicts timing results of a test matrix with n known columns. Use cases for our model are elaborated in Section V. An important capability in practice is to incrementally train models by including outliers found from inference results, e.g., at every k executions of the model inference phase. This allows the model to learn from outliers and improve over time. Such a methodology might follow the feedback loop (blue) indicated in Figure 2.[4]

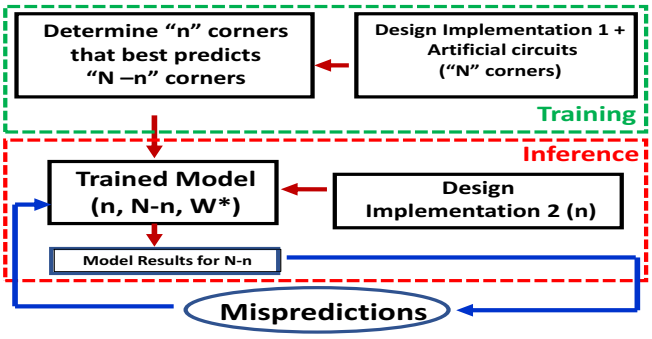

Fig. 2. Our modeling flow. The potential feedback loop (blue arrows) is discussed in Sections VI below.

IV. Data Generation

Recall that timing results are represented using matrices that can be fed into our models. We now describe our data generation, including design data and analysis conditions used in our experiments, followed by flows for our artificial circuit generation and matrix generation.[5]

[4]In a typical SOC physical implementation methodology, a given block (hard macro) may go through SP&R&Opt steps many times over the course of several months, each time with slightly different constraints or floorplan. Our approach can naturally exploit such a design process context and timeline.

[5]We do not separately present data preparation for inference. Reprising previous discussion, the data flow for model inference is as follows. Recall that some number of known corners n has been determined. Given n, we generate the test matrix, X_{test}, from timing results at n known corners. Then, using the trained model parameters W^* for the $n = |\{T_{known}\}|$ corners, we predict timing results in unknown corners $\{T_{unknown}\}$ of the test matrix.

Table II describes the data used in our modeling experiments. We use four public designs obtained from [4] [5] [7], in-house developed artificial circuits, and three industrial designs (*prod1*, *prod2* and *prod3*) in sub-14nm technology nodes. The space of analysis corners for our experiments in 28nm FDSOI (Experiments 1-4 below) is the set of combinations of Process (SS, TT, FF), Voltage (0.6 to 1.30 V in steps of 0.05), Temperature (-40C and 125C) and BEOL (RCWORST) corners. An analogous space is used for 16nm experiments, and we have only limited insight into the sub-14nm test data (prod*) obtained from our industry collaborator.

TABLE II
Design data used for experiments.

Design	# Instances	# Flip-Flops	# Data points
dec_viterbi	61K	26K	168K
netcard	303K	66K	186K
leon3mp	450K	100K	744K
megaboom	990K	350K	510K
artificial	2.4M	400K	746K
prod1	-	-	21K
prod2	-	-	111K
prod3	-	-	27K

Artificial Circuits Generation. We have evaluated the potential benefit of artificial circuits (i.e., small timing paths) used during an initial, "bootstrap" training phase of modeling. Algorithm 1 describes our artificial circuit generation flow. Input to this flow is a configuration file that contains circuit variables such as the number of stages in a timing path, $\{num\_stages\}$; standard cell types in the path defined by $\{Cell1\}$, $\{Cell2\}$, $\{Cell3\}$ and $\{Sink\}$; launch and capture flop-types defined by $\{LFlop\}$ and $\{CFlop\}$; aggressor cell types defined by $\{Agg\_Cell\}$; load cap range defined by CL_{range}; transition (slew) time values defined by TR_{range}; and clock period values defined by $\{Period\}$. The circuit generation sweeps through defined values for each variable; random values are generated between 0 and CL_{range}, and between 0 and TR_{range}, for these two variables. For each combination of the above-defined configuration variables, our flow generates a gate-level netlist (doe.v) (which comprises a collection of paths), along with an associated parasitic file (doe.spef), a transition annotation file (doe.timing) and a constraints file (doe.sdc). Figure 3 shows a schematic of a timing path in our artificial netlist. The path has three stages between a launch flop and a capture flop. Our circuits capture a wide range of coupling capacitance, wire capacitance and input transition values along with various combinations of standard cells.

Algorithm 1 Artificial circuit generation.

Input: Configuration file containing $\{num\_stages\}$, $\{Cell1\}$, $\{Cell2\}$, $\{Cell3\}$, $\{Sink\}$, $\{LFlop\}$, $\{CFlop\}$, $\{Agg\_Cell\}$, CL_{range}, TR_{range}, $\{Period\}$
Output: Design data for timing analysis
Sol = $\{doe.v, doe.spef, doe.sdc, doe.timing\}$
for i in $\{num\_stages\}$ **do**
  for j in $\{LFlop\}$ **do**
    for k in $\{CFlop\}$ **do**
      for $c1$ in $\{Cell1\}$ **do**
        for $c2$ in $\{Cell2\}$ **do**
          for $c3$ in $\{Cell3\}$ **do**
            for $s1$ in $\{Sink\}$ **do**
              for $p1$ in $\{Period\}$ **do**
                doe.v ← genVerilog () // netlist generation
                doe.spef ← genSpef () // spef generation
                doe.timing ← genTiming () // slew annotation
                doe.sdc ← genSDC () // constraints generation
$Sol \leftarrow \{doe.v, doe.spef, doe.sdc, doe.timing\}$
return Sol

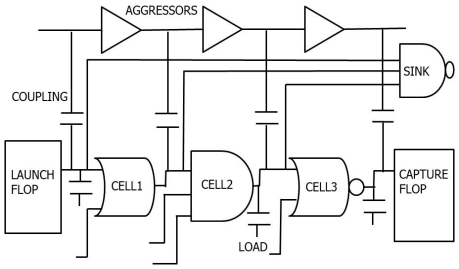

Fig. 3. Illustration of artificial circuits.

Matrix Generation Flow. Once we have design data and corresponding timing graphs at various analysis corners, we translate the timing results into an equivalent matrix in which rows represent delay values of timing paths (i.e., a single, fixed path per row), and columns represent corners. In anticipated usage, this matrix would be input to model training. Our method finds $\{P_i\}$ timing paths (one for each endpoint) from each corner's timing session db_i. It then collects the union of timing paths $\{P_{union}\}$ across all corners. Timing is evaluated for each unique path in $\{P_{union}\}$, in all corners. Since the cardinality of $\{P_{union}\}$ is high, we restrict the number of endpoints to a tractable number, num. Algorithm 2 details the matrix generation flow.[6]

Algorithm 2 Matrix generation.

Input: Timing database sessions at N corners DB = $\{db_1, db_2, ..., db_N\}$, number of endpoints num
Output: Matrix with path delay values
Sol = Matrix M_1 $\{R_{train} \times N\}$

1: **for** i in DB **do**
2: $\{P_i\} \leftarrow$ getTimingPaths(i, num, $nworst = 1$) // List of timing paths, num endpoints per endpoint
3: **if** True **then**
4: $\{P_{union}\} \leftarrow \{P_1\}\ U\ \{P_2\}\ U\ \{P_3\}\ ..\ U\ \{P_N\}$ // union of paths across all corners
5: $\{P_{unique}\} \leftarrow$ uniquePaths($\{P_{union}\}$) // unique paths from union of paths
6: **for** i in DB **do**
7: **for** j in $\{P_{unique}\}$ **do**
8: $M_1[j][i] \leftarrow$ evalTiming(j, i) // estimate delay for path j in i
9: $Sol \leftarrow M_1$
// Delay matrix $R_{train} \times N$
10: **return** Sol

V. Experimental Validation

Our simple, regression-based modeling approach is premised on the fact that the path delay values at different analysis corners are strongly correlated. These correlations are seen via principal component analysis [9]. Figure 4 plots the eigenvalues (normalized by the leading eigenvalue) of the covariance matrices for the data sets of the first four public benchmark designs listed in Table II. The relative magnitudes of these eigenvalues measure the relative variance captured by different principal components of the data. Note that while the data for each design consists of path delay values at 44 different analysis corners, the variance of the data is concentrated in a subspace of much lower dimensionality. In particular, several orders of magnitude separate the leading eigenvalues in these covariance matrices from those at the middle or bottom of the spectrum.

[6]We have also explored an alternate method that finds worst timing paths across all endpoints in one corner and evaluate timing for these paths in the rest of the $N-1$ corners. This ensures that all endpoints are covered, but is biased toward worst paths of a single corner. We have evaluated both methods and observe only negligible differences in our inference results.

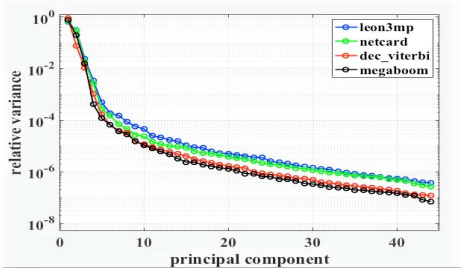

Fig. 4. Relative variance captured by successive principal components of the first four data sets in Table II (44 analysis corners).

A. Reporting Metrics

During the inference phase, we predict elements in the Y_{test} region of Figure 1. This region has R_{test} data points and $N-n$ columns. For each element (i^{th} row and j^{th} column) of Y_{test}, we define ${d_m}^{ij}$ as the predicted path delay and ${d_a}^{ij}$ as the corresponding actual path delay from golden timing analysis. We also define ${\epsilon_{abs}}^{ij}$ as the absolute error for each element of Y_{test}.

Though our model aims at reducing the square of absolute mispredictions, we understand from industry collaborators that relative delay prediction error is a valuable criterion. Intuitively, a timing path with larger path delay value can afford larger misprediction, compared to a timing path with smaller path delay. This is because the former has scope to fix violations with data path optimizations, even with a misprediction; the latter, having less implementation flexibility, cannot afford misprediction as easily (hence, cost of model misprediction is higher). With this in mind, we also report relative errors ${\epsilon_{rel}}^{ij}$.

To quantify the model accuracy for the entire Y_{test} region of the matrix using a single value, we report three lumped metrics, namely, $mean$, 99^{th}percentile and 99.99^{th}percentile values denoted by ϵ_{mse}, ϵ_{99p} and ϵ_{99p99}. Table III explains our model accuracy metrics.

TABLE III
Model accuracy metrics.

Notation	Meaning
${d_m}^{ij}$	Model predicted delay (i^{th} row and j^{th} column of Y_{test})
${d_a}^{ij}$	Actual delay, i^{th} row and j^{th} column of Y_{test}
${\epsilon_{abs}}^{ij}$	$\lvert {d_m}^{ij} - {d_a}^{ij} \rvert$ (Absolute error)
${\epsilon_{rel}}^{ij}$	$\frac{{\epsilon_{abs}}^{ij}}{{d_a}^{ij}}$ (Absolute relative error)
ϵ_{mse}	$\sqrt{\frac{1}{R_{test}} \sum_{ij} ({\epsilon_{rel}}^{ij})^2}$ (Root of mean squared relative errors)
ϵ_w	$\max_{ij}\{\epsilon_{rel}\}$ (Max of all absolute relative error values)
ϵ_{99p}	99^{th} percentile value of $\{\epsilon_{rel}\}$ (sorted in ascending order)
ϵ_{99p99}	99.99^{th} percentile value of $\{\epsilon_{rel}\}$ (sorted in ascending order)

B. Design of Experiments

In our experiments, we use in-house developed artificial designs and four public benchmark designs, listed in Table II. We use 28nm FDSOI and 16nm foundry enablements for our model evaluation. We also use sub-14nm foundry enablement on three industrial designs for our model evaluation.

Data Path Delay Model: Experiment 1. The model is trained with data points from post-routed implementation of a real design and tested on an unseen post-routed implementation

of the same design. This use case is relevant to exploring multiple physical implementations of the same design (or, re-analyzing implementations through the months-long physical design process) without having to analyze timing in all corners.

Artificial Testcases: Experiment 2. The goal of this experiment is to assess potential benefits of our artificial testcase development methodology. This reflects a hypothetical "ideal scenario", wherein we have the capability to produce artificial testcases that span the entire space of real designs. In this experiment, our model is trained with artificial testcases and tested on unseen real designs. The motivating scenario: a one-time trained model using artificial circuits can predict timing analysis in unknown corners of any real design, using timing analysis in few known corners of the same real design.

Clock Insertion Delay Model: Experiment 3. Accurate clock network synthesis and optimization are essential for advanced-node IC design. The goal of this experiment is to demonstrate the usefulness of our model in predicting clock insertion delay at unknown corners $\{T_{unknown}\}$ using clock insertion delay at known corners $\{T_{known}\}$. This ensures that the clock network is synthesized considering its delay values at all timing corners, without overdesigning the clock tree or leaving violations unattended till very late in the design cycle.

Corner Scalability: Experiment 4. Experiments 1-3 use $N = 44$ corners. Since design methodologies in advanced nodes require timing analysis at $N \gg 44$ corners, we investigate whether increasing the number of corners N demands an increase of n on the same scale. We increase the number of corners from 44 to 82 in this experiment.

Technology Independence: Experiments 5 and 6. All of Experiments 1-4 use 28nm foundry enablement. Experiments 5 and 6 seek to confirm the utility of our modeling in more advanced 16nm and sub-14nm foundry enablements.

C. Experimental Results

In all of our results, given as plots below, we report (y-axis) values of $mean$, $99^{th}percentile$ and $99.99^{th}percentile$ metrics, denoted as ϵ_{mse}, ϵ_{99p} and ϵ_{99p99} (see Table III). On the x-axis of each plot, we show n (number of known corners), ranging from 1 to $N-1$.

Results of Experiment 1. We use timing paths from a physical implementation (0.85 utilization, aspect ratio 1) for training, and test on an unseen physical implementation (0.75 utilization, aspect ratio 0.9) of the same design. The plots in Column 1 of Figure 5 show that $\epsilon_{mse} \leq 0.005$ (0.5% error) for $n = 4$, $\epsilon_{99p} \leq 0.01$ (1% error) for $n = 5$ and $\epsilon_{99p99} \leq 0.01$ (1% error) for $n = 14$.

Results of Experiment 2. We use timing paths from in-house developed artificial designs for training, and test on unseen real designs. The plots in Column 2 of Figure 5 show that $\epsilon_{mse} \leq 0.01$ for $n = 18$, $\epsilon_{99p} \leq 0.01$ for $n = 23$ and $\epsilon_{99p99} \leq 0.01$ for $n = 32$. Larger n needed to predict timing at unobserved corners suggests a need for improvement of our artificial circuit generation methodology.

Results of Experiment 3. We use 10% of clock paths for training, and test on unseen 90% clock paths of the same design. The plots in Column 3 of Figure 5 show that $\epsilon_{mse} \leq 0.005$ for $n = 3$, $\epsilon_{99p} \leq 0.01$ for $n = 4$ and $\epsilon_{99p99} \leq 0.01$ for $n = 6$.

Results of Experiment 4. We use 10% of data paths for training and test on unseen 90% data paths of the same design. The plots in Column 4 of Figure 5 show that $\epsilon_{mse} \leq 0.005$ for $n = 4$, $\epsilon_{99p} \leq 0.01$ for $n = 6$ and $\epsilon_{99p99} \leq 0.01$ for $n = 23$. These results suggest that a small number of known analysis corners n can suffice to accurately predict timing analyses at unknown corners, even as N grows large.

Results of Experiments 5 and 6. We use 10% of data points from a real design, for training and test on unseen 90% data points of the same design. We use $N = 58$ for experiments using 16nm foundry enablement. The plots in Column 5 of Figure 5 show that $\epsilon_{mse} \leq 0.005$ for $n = 11$, $\epsilon_{99p} \leq 0.01$ for $n = 6$ and $\epsilon_{99p99} \leq 0.01$ for $n = 21$.

Figure 6 shows results using sub-14nm technology libraries on industrial designs *prod1*, *prod2* and *prod3*. The plots in Figure 6 show that $\epsilon_{mse} \leq 0.005$ for $n = 5$, $n = 7$ and $n = 9$, $\epsilon_{99p} \leq 0.02$ for $n = 6$, $n = 9$ and $n = 12$, and $\epsilon_{99p99} \leq 0.03$ for $n = 8$, $n = 21$ and $n = 15$, for designs *prod1*, *prod2* and *prod3* respectively.

Worst-case Errors and Outliers. The results of Experiment 6 show that improved accuracy in sub-14nm nodes is an important direction for our future work. Furthermore, in each of our experiments, we see a handful of data points that fail to be reconstructed to their high-dimensional space accurately, even with large values of n. (In the industry datasets studied with Experiment 6, we understand that outliers are unsurprising for reasons such as (i) existence of rare path types (e.g., memory as opposed to reg-to-reg) with limited training examples, and (ii) existence of isolated corners with no similar corners in the provided dataset.) While root-cause analysis and improvement of outlier (high ϵ_w) predictions is another important direction for our future work, we note that industry design methodology teams consider such outliers to be expected, and that effects of a few mispredictions are insignificant relative to (i) the analysis improvement and design convergence benefits from a predictive model, and (ii) analysis inaccuracies that exist in current methods.[7]

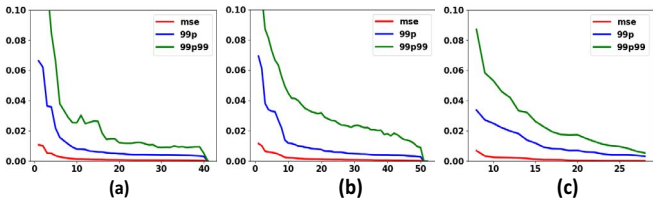

Fig. 6. Results of Experiment 6 using sub-14nm technology libraries. Plots of ϵ_{mse} (red), ϵ_{99p} (blue) and ϵ_{99p99} (green) versus n for industrial designs (a) *prod1*, (b) *prod2* and (c) *prod3*.

VI. Conclusions

In this work, we have taken a data-driven approach to the problem of timing analysis in advanced-node IC design.

[7]Our industry collaborator indicates that minor violations must be dealt with at the end of timing closure anyway, hence worst-case outliers even with 10% or greater prediction error would not cause concern. If deemed necessary, such outliers could be caught up front by an STA run covering all corners, incurring a one-time cost. Incremental model training, as suggested in the blue arrows of Figure 2, could also help cure outliers through physical design iterations.

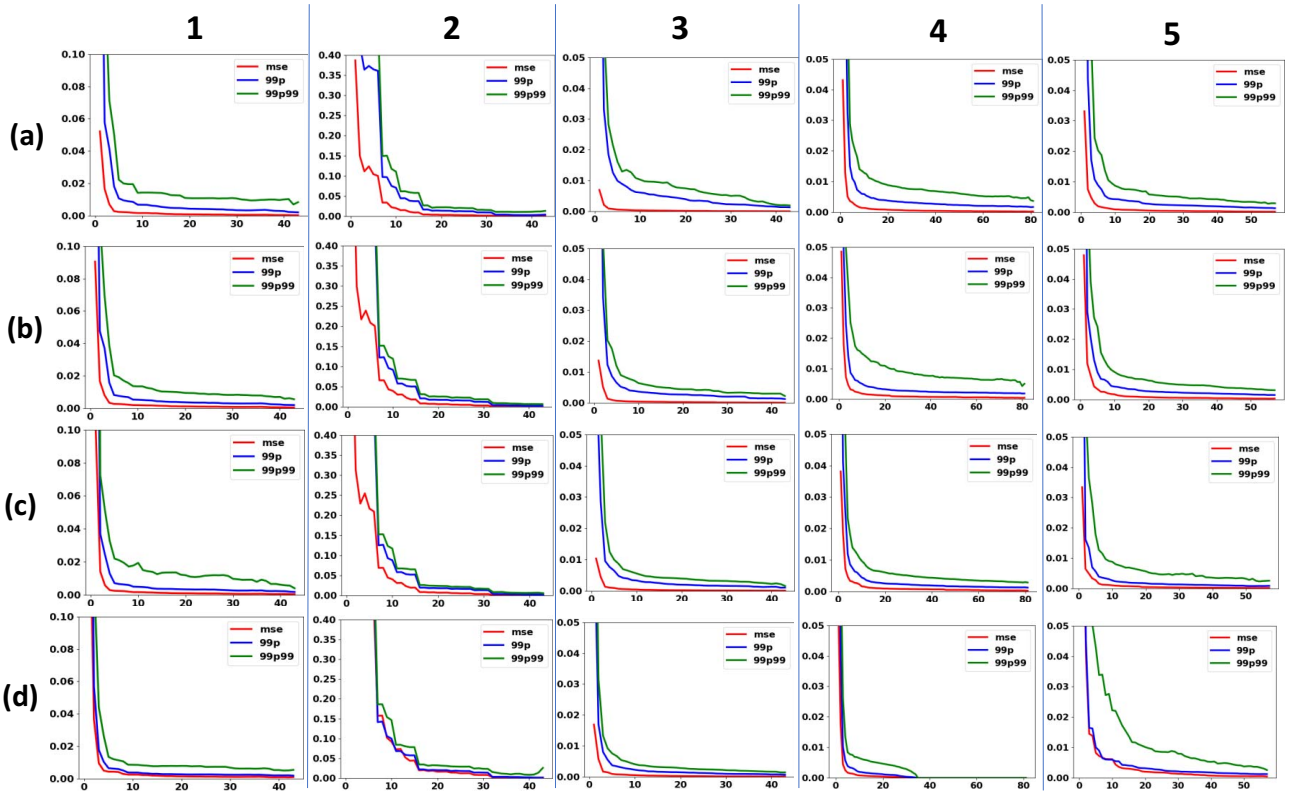

Fig. 5. Results of Experiments 1, 2, 3, 4 and 5 (left to right): Plots of ϵ_{mse} (red), ϵ_{99p} (blue) and ϵ_{99p99} (green) versus n, for designs (a) *dec_viterbi*, (b) *netcard*, (c) *leon3mp* and (d) *megaboom* (top to bottom).

We have shown that simple linear methods in multivariate regression can accurately predict a large set of unknown corners from a smaller set of known ones. For example, with a 1M-instance example in foundry 16nm enablement (10% training, 90% testing), we obtain a model based on 10 observed corners that predicts timing results at the remaining 48 unobserved corners with less than 0.5% relative root mean squared error, and 99^{th} percentile relative prediction error less than 0.6%. We are currently exploring numerous other directions to address further challenges. With large data sets, for example, it is possible to learn more flexible statistical models that do not make strong assumptions of linearity. Also, to handle outliers, it is possible to optimize more robust criteria in our statistical fits. So far these approaches have yielded incremental benefits, but it remains to find the optimal combination of strategies for the problem of timing analysis in advanced-node IC design.

ACKNOWLEDGMENTS

Research at UCSD is supported by Qualcomm, Samsung, NXP Semiconductors, Mentor Graphics, DARPA (HR0011-18-2-0032), NSF (CCF-1564302) and the C-DEN center. We thank Dr. Tuck-Boon Chan for many helpful discussions and inputs.

REFERENCES

[1] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.
[2] *Synopsys, Inc.*, https://www.synopsys.com
[3] *scikit-learn*, http://scikit-learn.org
[4] *OpenCores*, https://opencores.org
[5] *RISC-V*, https://riscv.org
[6] TAU Workshop, https://www.tauworkshop.com
[7] M. M. Ozdal, C. Amin et al., "The ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ISPD*, 2012, pp. 161-164.
[8] *Synopsys PrimeTime User Guide*, http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx
[9] I. T. Jolliffe, *Principal Component Analysis*, Springer, 2002.
[10] I. Guyon and A. Elisseeff, "An Introduction to Variable and Feature Selection", *J. Machine Learning Research* 3 (2003), pp. 1157-1182.
[11] S. Onaissi, F. Taraporevala, J. Liu and F. Najm, "A Fast Approach for Static Timing Analysis Covering All PVT Corners", *Proc. DAC*, 2011, pp. 777-782.
[12] S. Onaissi and F. N. Najm, "A Linear-time Approach for Static Timing Analysis Covering All Process Corners", *IEEE Trans. on CAD* 27(7) (2008), pp. 1291-1304.
[13] L. G. e Silva, L. M. Silveira and J. R. Phillips, "Efficient Computation of the Worst-Delay Corner", *Proc. DATE*, 2007, pp. 1-6.
[14] C. Visweswariah, K. Ravindran et al., "First-order Incremental Block-based Statistical Timing Analysis", *IEEE Trans. on CAD* 25(10) (2006), pp. 2170-2180.
[15] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, Springer, 2009.
[16] D. Stamoulis et al., "Linear Regression Techniques for Efficient Analysis of Transistor Variability", *Proc. IEEE ICECS*, 2014, pp. 267-270.
[17] J. J. Nian, S. H. Tsai and C. Y. Huang, "A Unified Multi-corner Multi-mode Static Timing Analysis Engine", *Proc. ASP-DAC*, 2010, pp. 669-674.
[18] A. B. Kahng, M. Luo and S. Nath, "SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects", *Proc. SLIP*, 2015, pp. 1-8.
[19] A. B. Kahng, S. Kang et al., "Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools", *Proc. SLIP*, 2013, pp. 1-8.
[20] S. S. Han et al., "A Deep Learning Methodology to Proliferate Golden Signoff Timing", *Proc. DATE*, 2014, pp. 1-6.
[21] A. B. Kahng, "Machine Learning Applications in Physical Design: Recent Results and Directions", *Proc. ISPD*, 2018, pp. 68-73.
[22] S. Bian et al., "LSTA: Learning-Based Static Timing Analysis for High-dimensional Correlated On-chip Variations", *Proc. DAC*, 2017, pp. 1-6.