From Global Route to Detailed Route: ML for Fast and Accurate Wire Parasitics and Timing Prediction

Vidya A. Chhabria
University of Minnesota
Minneapolis, MN, USA

Wenjing Jiang*
University of Minnesota
Minneapolis, MN, USA

Andrew B. Kahng
University of California
San Diego, CA, USA

Sachin S. Sapatnekar
University of Minnesota
Minneapolis, MN, USA

ABSTRACT

Timing prediction and optimization are challenging in design stages prior to detailed routing (DR) due to the unavailability of routing information. Inaccurate timing prediction wastes design effort, hurts circuit performance, and may lead to design failure. This work focuses on timing prediction after clock tree synthesis and placement legalization, which is the earliest opportunity to time and optimize a "complete" netlist. The paper first documents that having "oracle knowledge" of the final post-DR parasitics enables post-global routing (GR) optimization to produce improved final timing outcomes. Machine learning (ML)-based models are proposed to bridge the gap between GR-based parasitic and timing estimation and post-DR results *during post-GR optimization*. These models show higher accuracy than GR-based timing estimation and, when used during post-GR optimization, show demonstrable improvements in post-DR circuit performance. Results on open 45nm and 130nm enablements using OpenROAD show efficient improvements in post-DR WNS and TNS metrics without increasing congestion.

CCS CONCEPTS

• **Hardware** → **Static timing analysis**; *Best practices for EDA*; **Wire routing**; *Circuit optimization*; *Application specific integrated circuits*; **Physical design (EDA)**;

KEYWORDS

machine learning, static timing analysis, timing optimization

ACM Reference Format:
Vidya A. Chhabria, Wenjing Jiang, Andrew B. Kahng, and Sachin S. Sapatnekar. 2022. From Global Route to Detailed Route: ML for Fast and Accurate Wire Parasitics and Timing Prediction. In *Proceedings of the 2022 ACM/IEEE Workshop on Machine Learning for CAD (MLCAD '22), September 12–13, 2022, Snowbird, UT, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3551901.3556475

*Primary author

MLCAD '22, September 12–13, 2022, Snowbird, UT, USA

ACM ISBN 978-1-4503-9486-4/22/09…$15.00
https://doi.org/10.1145/3551901.3556475

1 INTRODUCTION

With the reduction in wire geometries and the resulting increase in per-unit length resistances and capacitances, wire delays contribute significantly to circuit delay and overall IC performance outcomes. For successful design closure, modern design flows must accurately estimate wire parasitics and delays, and use these estimates to perform timing optimizations such as net buffering and logic gate resizing at multiple stages of the RTL-to-GDS implementation flow.

Fig. 1 shows a standard physical design flow that highlights multiple timing optimization steps. It is essential to perform optimization several times in the flow so that netlist changes between successive stages are manageable within a convergent design methodology. This is because a too-drastic netlist change between flow stages can force looping back to earlier steps of the flow, rather than continuing forward. At the same time, the unavailability of routing information before the final stages leads to inaccurate wire delay estimates, challenging the efficiency of timing optimization.

To overcome this challenge, design flows use different models to account for wire parasitics during timing optimizations based on the information available at each stage. For example, as highlighted in Fig. 1, wireload models are used for gate-level optimization during logic synthesis. During global placement and even after clock tree synthesis (CTS), generic half-perimeter wirelength (HPWL) or FLUTE-based [11] Steiner tree estimates, scaled by layer-averaged per-unit resistances and capacitances, are used for electrical rule check (ERC, including max load and max transition rules) compliance and some gate-level optimizations. However, these models can be highly inaccurate (overly conservative or grossly optimistic) compared to the true parasitics and timing estimates after detailed route. Any optimization or synthesis using these models can produce solutions which are either overdesigned (pessimistic) or underdesigned (optimistic, i.e., failing electrical and/or performance constraints).

Despite being late in the physical design flow, the crucial routing stage also suffers from estimation inaccuracy. A design is typically routed in two stages: global routing (GR) and detailed routing (DR); see Fig. 1. The GR tool allocates routing resources to each net, and generates a routing plan that the DR tool takes as initial guidance toward a final routing solution. In modern tools, the GR's routing plan is in the form of *route guides* that contain information on layer assignments and Steiner tree topologies for each net [12][15] .

Importantly, the GR stage is typically followed by a timing optimization step (sizing, fanout clustering and buffering, etc.) as well as a final (post-optimization) placement legalization step, since better parasitic estimates are available post-GR compared to earlier flow

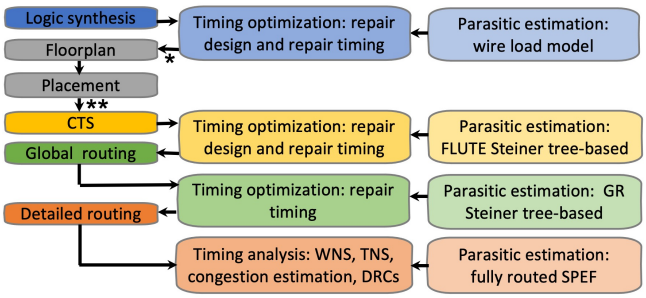


Figure 1: Timing optimizations in the physical design flow with different parasitic estimates. In modern production flows, the output of vendor A's synthesis tool is "de-buffered" when passed to vendor B's place-and-route (P&R) tool. Then, ERC compliance is enforced during global placement with buffering and resizing. The de-buffering and ERC-fixing steps are respectively marked with (*) and (**) in the figure.

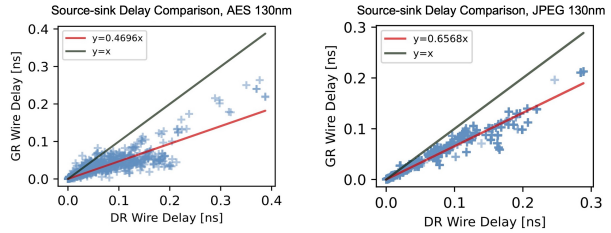


Figure 2: Discrepancy between post-GR and post-DR wire delays in AES 130nm and JPEG 130nm. Note that the slope k in the $y = kx$ best fit (red traces) differs between the designs.

stages. However, GR-based parasitic estimates are still inaccurate relative to final DR outcomes, as they do not fully comprehend such factors as detailed design rules, pin access challenges, and congestion, which are glossed over in GR. Together, these factors cause wire detours and layer reassignments during DR, which in turn cause GR-based estimates and DR-based outcomes to diverge.

Fig. 2 shows the inaccuracy in wire delay estimation when using a GR-based model that estimates wire parasitics using FLUTE-generated [11] Steiner trees from FastRoute 4.1 [17], as compared to the wire delay of corresponding detail-routed nets when using RC trees from post-DR parasitic extraction. The figures show discrepancies in the estimated wire delays in AES (15K nets) and JPEG (59K nets) implemented in the SkyWater 130nm open-source technologies [5]. For the most part, the GR-based wire delay estimates are smaller than the DR-based (ground truth) delay estimates, highlighting the optimism in GR-based parasitic and timing estimates. The figure also shows the best linear ($y = kx$) fit to the data: the slope k differs between two different designs that are implemented with the same tool (OpenROAD [2]) in the same technology.

"Oracle" knowledge of post-DR parasitics improves final outcomes. Recall from above that the use of inaccurate parasitic and timing estimates in any timing optimization can lead to harmful

Table 1: Comparison of post-DR WNS when using GR-based vs. DR-based parasitics for post-GR timing optimizations.

Design	Tech	# Nets	Post-DR WNS	
			GR-based parasitics	DR-based parasitics
DYNAMIC NODE	45nm	11598	-0.26ns	-0.26ns
AES	45nm	16836	-0.21ns	-0.19ns
IBEX	45nm	17566	-0.56ns	-0.60ns
JPEG	45nm	68247	-0.25ns	-0.17ns
RISCV32I	130nm	8150	-0.25ns	-0.17ns
IBEX	130nm	15307	-0.24ns	-0.11ns
AES	130nm	15369	-0.27ns	-0.09ns
JPEG	130nm	59573	0.24ns	0.02ns

pessimism (overdesign that wastes resources) or optimism (underdesign that leads to design iterations). We have conducted a motivating study to show the potential benefit of "oracle" knowledge of post-DR wire parasitics, were these parasitics to somehow be available to post-GR timing optimizations. Table 1 shows results on three open-source 45nm [1] testcases and four open-source 130nm [5] testcases, using an open-source flow [3]. The table highlights the cost of post-GR buffering and resizing solutions that are driven by inaccurate parasitics. For example, if the discrepancy in Fig. 2 (AES 130nm) can be corrected in post-GR, the post-DR worst negative slack improves by 180ps (−270ps → −90ps). We ascribe the WNS improvement to the early identification of true (post-DR) timing violations, which allows post-GR optimizations to efficiently buffer nets and resize logic gates on truly critical timing paths that might otherwise be missed due to optimism, or unnecessarily buffered and resized due to pessimism. This motivates the key result of our work, which is to apply machine learning (ML) to close the post-GR to post-DR parasitic estimation gap.

Related works. Several researchers have worked in the general area of ML-based delay prediction during physical design. For a specified net topology, [10] builds an XGBoost-based ML model for the wire delay of a net of fixed topology, trained on commercial parasitic extraction and timing analysis tools. The layout impact of macro blocks in floorplanning on timing is predicted using boosting and SVM in [8]. The work in [7] solves the problem of predicting wire delay and slew based on placement results, prior to GR, while [18] predicts path delays prior to routing, based on placement features, using a transformer network and residual model. The work in [14] uses a look-ahead RC network generated by a coarse routing step (decomposition of multi-pin nets into two-pin nets, then routed using L-shaped routes) on the placed design for feature extraction, and uses this to perform post-placement net-based timing prediction. In [13], a GNN model is used to estimate pre-routing slacks at the endpoints of the design.

This paper bridges the gap between GR-based parasitic and timing estimates and the post-DR-based ground truth using ML during post-GR optimizations. To the best of our knowledge, this is the first work that addresses this problem and uses the predicted timing and parasitic estimates for timing optimizations. We propose three ML models that predict net parasitics, wire delays, and wire slews, respectively. Based on features available from the route guides

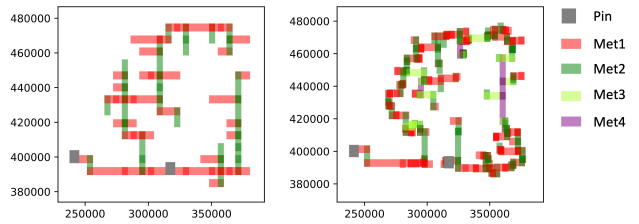


Figure 3: Difference in the GR guides (at left) and post-DR routes (right) of a net from JPEG in 130nm technology.

(e.g., HPWL, source-to-sink Manhattan distances, number of sinks), ML predicts post-DR parasitics and circuit delays during post-GR timing optimizations without performing runtime-intensive DR.

The key contributions of this work are as follows:

(1) We show how ML enables the fast and accurate prediction of post-DR parasitics (π model) and timing estimates (wire delays and slews) using post-GR information.
(2) We utilize the ML model during post-GR optimization to improve *post-DR* performance without affecting routability.
(3) The ML model can predict parasitics and timing with an average error of 6% and 8%, respectively when compared to a ground-truth-based flow, with a small increase in runtime (to perform ML inference) when compared to a traditional flow.
(4) We apply the ML model to the OpenROAD [6] physical design flow and show up to 0.12ns savings (130nm node) in post-DR worst negative slack (WNS), without degrading congestion.

2 PRELIMINARIES

2.1 Post-GR vs. Post-DR Routing Estimates

Timing-driven physical design requires an estimate of the delay of each stage of logic. This corresponds to the sum of the gate delay and the wire delay, each of which is dependent on wiring parasitics. To predict circuit timing, post-GR interconnect parasitics may be estimated based on the route guides. The estimated RCs for a net are determined by the length of its Steiner tree used for GR and the layer assignment. We use FastRoute [17], which performs precise layer assignment for each route, i.e., the route guides specify the precise layer for each segment of the Steiner tree.

The precise wire routes and wire adjacency relations are available for all nets only after DR is complete. Therefore, post-DR parasitic extraction provides the exact ground truth parasitics for each route. As seen in Fig. 3, the DR solution does not always choose a path similar to that specified by the route guides. Due to such differences in the post-GR prediction and post-DR wiring topologies of the nets, the timing results based on post-GR parasitics and the ground truth can sometimes be quite different, leading to discrepancies in GR-stage timing estimates. Depending on factors such as changes in the Steiner tree, or the coupling capacitances to neighboring wires, post-GR parasitic estimation may be pessimistic or optimistic.[1]

[1]Although the plot in Fig. 2 is dominated by optimistic delay predictions, a detailed examination of the data shows a mix of optimistic and pessimistic predictions.

2.2 Timing Estimation

A unit operation in static timing analysis is the process of estimating the delay of a single logic stage, consisting of a gate driving its fanouts through a net, as shown in Fig. 4. The upper part of the figure shows the distributed RC tree for the net driven by gate C. The delay of a logic stage consists of the gate delay and interconnect delay. Given the estimated capacitive load C_L at the output of a gate (the "driving point"), and the transition time τ at the gate input, the gate delay is expressed through a lookup table (LUT) as

$$D_{gate} = f(\tau, C_L). \quad (1)$$

The gate delay for an intermediate pair of (τ, C_L) values that does not map to a LUT entry is computed using interpolation. The transition time at the gate output is estimated in a similar way, and uses the same axes as gate delay LUTs.

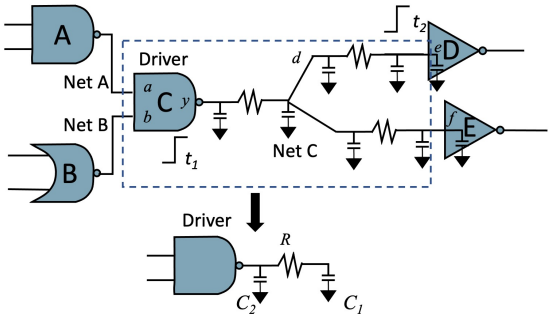


Figure 4: The RC tree model of a net in a logic stage, and its reduction to an equivalent π-model.

Since a wire is a distributed transmission line, it is modeled by the distributed RC model shown in Fig. 4, which segments the wire and creates lumped approximations for each segment. A segmented RC model cannot directly use the LUT of Eq. (1) as the load is not purely capacitive. Modern timing analyzers overcome this by (1) generating a π-model reduction for the driving point impedance [16], with elements R , C_1, and C_2 (Fig. 4, bottom) chosen so that the first three admittance moments of the interconnect match those of the reduced model; (2) using the π-model to find an effective capacitance, C_{eff}, that models resistive shielding; and (3) using $C_L = C_{\text{eff}}$ in Eq. (1) to compute the gate delay. Finally, model order reduction techniques are used to compute the transfer function from the driving point to each fanout. Based on the delay and slew at the driving point, the waveform at each fanout is computed, yielding the wire delay and slew. The sum of the gate and wire delays constitute the stage delay to the fanout, and the slew at each fanout is used as the input slew for the next logic stage.

3 DR TIMING PREDICTION FRAMEWORK

3.1 Overview

Fig. 5(a) highlights a typical routing flow in physical design, where GR is followed by timing optimization before the final DR. These optimizations rely on parasitic estimates from route guides, which use Steiner trees to construct an RC tree network. This results in timing inaccuracy, relative to the final DR timing (see Fig. 2). In

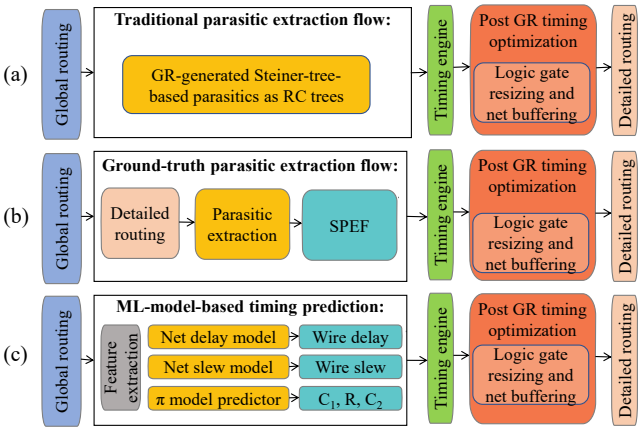


Figure 5: Three flows that use different parasitic estimates for post-GR timing optimizations: (a) traditional flow (Steiner tree-based RC estimates), (b) ground-truth (DR followed by parasitic extraction to determine post-DR parasitics), and (c) our flow (fast ML engine for post-DR parasitics and timing).

an ideal flow, shown in Fig. 5(b), performing DR and extracting parasitics would provide accurate timing estimates, but such a flow is impractical due to the high computational expense of DR.

We propose the use of an ML-based flow to predict post-DR timing from features extracted at the post-GR stage, as highlighted in Fig. 5(c). Through fast ML inference, the model can rapidly predict post-DR timing estimates without performing time-intensive DR. Our framework leverages three XGBoost-based ML models to predict the following three post-DR metrics:

(i) Source-sink wire delay: This ML model is applied on a per-sink basis and it predicts the delay between the driving point and the sink pins. For example, in Fig. 4, for net C, the model predicts the wire delay between the driving point (pin y of the driver gate C) and the sink (pin e of the gate D). Similarly, it predicts the source-sink delay between the driver pin y and sink pin f.

(ii) Source-sink wire slew: This ML model is used to predict post-DR wire slew at each sink in the design. We predict source-to-sink wire slews, i.e., the difference in the transition times between the waveforms at the driver pin and the sink pins. In Fig. 4, the source-sink wire slew is $(t_1 - t_2)$, where t_1 is the transition time at the driving point pin y, and t_2 is the transition time at the sink pin e.

(iii) Wire parasitics (π model): This ML model predicts post-DR π-model parasitics – R, C_1, and C_2 – as shown in Fig. 4.

The above three ML models estimate post-DR circuit delays with the help of an STA engine for annotated delay propagation. The first two models are directly used to annotate wire delays and wire slews in the timer while the latter is used indirectly in the timer to calculate C_{eff}, which is used as C_L in (1) to compute the gate delay.

3.2 Feature Engineering

For accurate prediction, it is critical to account for important features that impact DR timing. Since the three models are applied after GR, we are constrained to using post-GR information.

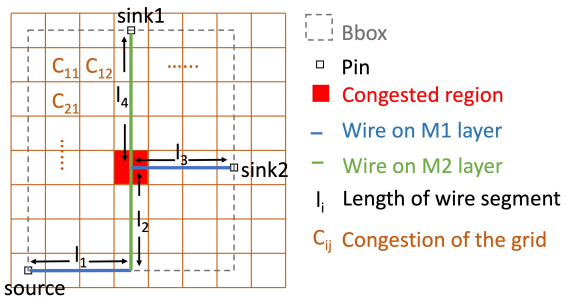


Figure 6: A route guide for a three-pin net with a source and two sinks. The net is routed using four wire segments with lengths l_1, on M1, l_2 on M2, l_3, on M1, and l_4, on M2.

3.2.1 Input features: Source-sink delay and slew prediction models.

HPWL: This feature is extracted after placement, and is shown by the bounding box in Fig. 6. It is a lower bound on the wire length: for nets with numerous sinks, it can be a significant underestimate.

Number of sinks: This feature is extracted from the gate-level netlist. Multi-sink nets show larger discrepancies between post-GR and post-DR wirelengths as they tend to detour and/or have net lengths that exceed the HPWL, as noted above.

Slew at the driving point: This indicates the signal strength at the source pin. The slew is extracted from a timing analysis tool, which uses GR-generated Steiner tree-based parasitics. The slew at the source pin affects both wire delay and wire slew. A smaller driving point slew leads to smaller source-sink delay and slew.

Congestion estimates: Congestion is critical to bridging the discrepancy between GR and DR. Nets whose GR-generated route guides go through congested regions tend to detour in DR, for routability, pin access, and DRC-related reasons. Therefore, congestion estimates are essential for predicting wire detours. As features, we use both the mean and standard deviation of the congestion in all GCells in the bounding box of the net as shown in Fig. 6.

Rise and fall transitions: Since the wire delays and slews are different for the rise and fall transitions, we encode the switching direction with a binary-encoded feature (0 for rise and 1 for fall).

The above-listed features are on a per-net basis, i.e., identical for all sinks on the net. Since we predict wire delays and slews on a per-sink basis, we also provide the following sink-specific features:

Source-sink length: The source-sink length is defined as the total length of the wire segments that connect the driving point to the target sink pin, and is extracted from the GR-generated route guides. For the example net in Fig. 6, routed in layers M1 and M2, the source-sink length for sink1 is defined as $(l_1 + l_2 + l_4)$. Since the source-sink length is proportional to the source-sink delay and slews, this is a critical feature in estimating post-DR wire and slew delays.

Source-sink R, C: These features list to the total resistance and capacitance, respectively, between the driving point and the target sink. The total resistance [capacitance] is the sum of the products of the segment length and the per-unit resistance R_{Mi} [capacitance C_{Mi}] of its assigned layer Mi, over all source→sink wire segments. In Fig. 6, the total resistance to sink1 is $(l_1 \times R_{M1} + l_2 \times R_{M2} + l_4 \times R_{M2})$, and the total capacitance is $(l_1 \times C_{M1} + l_2 \times C_{M2} + l_4 \times C_{M2})$.

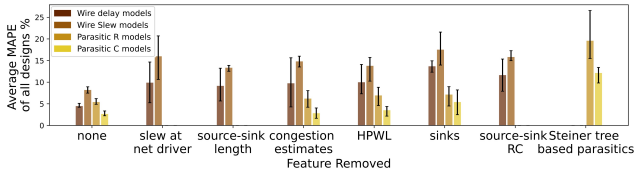


Figure 7: Feature sensitivity analysis showing the average mean absolute percentage error (MAPE) (y-axis), for each removed feature (x-axis), for all 45nm designs.

3.2.2 Input features for post-DR π-model parameter prediction model. For the ML model that predicts the parameters of the π-model at the driving point, we use the HPWL, number of sinks, and congestion estimates as features. We use additional features related to the values of R, C_1, and C_2, generated by applying the O'Brien/Savarino model to the GR-generated Steiner tree.

3.2.3 Feature importance. To demonstrate that each selected feature is indispensable to the model, we perform a sensitivity analysis for each feature. For example, for source-sink wire delay prediction, we measure the test accuracy for different models, each trained by removing one specific feature at a time. The x-axis of Fig. 7 lists the feature that has been removed, and the y-axis highlights the average %error of all 45nm designs. The figure is annotated with an error bar that shows the maximum and minimum %error across all the designs. We find that the model has the best test accuracy when all the features are selected. Thus, each feature contributes to improving the accuracy of the model.[2]

3.3 ML Models and Training

XGBoost ML engine. All three ML models are implemented using XGBoost [9], an ensemble learning algorithm based on gradient boosting. XGBoost predicts the target variable using parallel tree boosting, combining estimates from several models including gradient-boosted decision trees. A linear combination of multiple trees is used to describe the complex nonlinear relationship between input and output data. New trees are generated based on previous trees, using gradient descent to minimize a loss function.
Ground-truth data generation. Our ground-truth data is generated using the flow highlighted in Fig. 5(b). We use a branch of OpenROAD-flow-scripts [3] which performs post-GR optimizations. The ground-truth data is generated for four designs implemented in two open-source technologies, a 45nm technology (IBEX, AES, JPEG, Dynamic node), and a 130nm technology (IBEX, AES, JPEG, and RISCV32I). To increase the diversity of our training data set we run the ground-truth flow for each design with different floorplan areas and placement utilization settings. Different utilization settings result in designs with different levels of congestion and consequently in nets that have different lengths due to detours. In total we use 16 designs (four designs with four different utilizations each) for generating ground-truth data. For each net in the design, we extract the features described in Section 3.2 and its corresponding ground-truth labels. The labels for the three ML models are extracted after timing analysis (green box in Fig. 5(b)) and include the source-sink wire delays, source-sink wire slews, and π-model parameters. For the source-sink delay and slew model, each sink in the design represents a single datapoint, and for the π-model predictor each net in the design is a single datapoint. Our ground-truth dataset contains 456,661 datapoints in 45nm technology and 394,162 datapoints in 130nm technology.

The pace of ground-truth data generation is slow (40 minutes per design, on average) but is a one-time cost per technology since the trained model can be applied to new designs to rapidly and accurately predict post-DR parasitics and timing.
Model training. The model is trained using root mean squared error (RMSE) as the loss function. We use XGBoost regressor, with a learning rate 0.01. We choose the maximum treedepth = 4, the number of estimators = 900, and the subsampling ratio = 0.8. The hyperparameter values are chosen based on a full search of a discrete grid on the domain of the hyperparameters: the assignment with the highest score is used for parameter tuning. The models are not sensitive to a small change in the hyperparameter.

3.4 ML Inference in the Physical Design Flow

The trained ML models are applied to the flow shown in Fig. 5(c). At the post-GR phase, the ML model features are extracted from the design data and route guides. Next, the features are fed into the three ML models, which perform a fast and accurate inference to predict source-to-sink wire delays and wire slews, as well as π-model parameters. The predicted estimates are then annotated via Tcl APIs in the STA engine: the π model parameters are used by the timing engine to estimate C_{eff} which is used in turn to calculate gate delays, while the source-sink wire delays and slews are directly used to compute the net delays and net transition times. The timer performs an update to propagate these annotated wire delays and gate delay estimates. The new ML-predicted timing estimates are used by the timing optimizer to perform gate resizing and buffer insertions which fix setup, hold, maximum slew, maximum fanout, and maximum load violations.

The ML-based inference flow provides an accurate estimate of post-DR timing without performing time-intensive DR. These estimates are useful for efficient buffering and resizing, as the optimizer now has knowledge of the truly (post-DR) critical paths.

4 EXPERIMENTAL SETUP AND RESULTS

To evaluate our ML model for accuracy and its application in a physical design flow we set up the three flows shown in Fig. 5 using OpenROAD-flow-scripts [3]. The first flow is the traditional flow where Steiner tree-based parasitics are used for post-GR timing optimizations (Fig. 5(a)). The second flow is an ideal ground-truth-based flow where we use post-DR parasitics for post-GR timing optimizations (Fig. 5(b)). This is the flow that we strive to match with our ML model. The third flow is our ML-prediction-based flow where we apply the three models to predict post-DR timing for post-GR optimizations (Fig. 5(c)). Our experiments are performed on two open technologies, NanGate 45nm [1] and SkyWater 130nm [5]. The ML model training and inference is implemented in Python 3.6 and performed on a machine with Intel Core i7 CPU @3.6GHz and NVIDIA RTX 2080Ti GPU and 64GB RAM. The predicted parasitics,

[2]Note that not all models use all features. Therefore, there are missing bars for certain features in the figure.

wire delays, and wire slews are annotated into the timing engine by modifying OpenROAD [2] and OpenSTA [4] source code.

We evaluate our ML-based flow against the traditional ("Trad.") and ground-truth-based flows for (i) accuracy, and (ii) impact on post-DR outcomes – worst slack (WS), total negative slack (TNS), runtime, and congestion. All results presented are for a test design that is unseen during training. For example, in 130nm technology, the predicted labels of al utilization setting of IBEX design are generated from a model trained on all utilization settings of AES, JPEG and RISCV32I, and the predicted labels of all utilization settings of AES design are generated from a model trained on data from all utilization settings of IBEX, JPEG and RISCV32I.

4.1 ML Model Accuracy Evaluation

We analyze the accuracy of the ML models at both the net/sink level (the ML model performs inference on a per-sink/per-net basis) and at the path level. It is critical to evaluate the predicted timing at both levels as net-level errors have the potential to accumulate or cancel during delay propagation.

4.1.1 Net/sink-level accuracy: We use mean, maximum, and standard deviations of the absolute %error as metrics for evaluation. The absolute percentage error is the absolute difference in the ground-truth and the predicted value with respect to a reference. Our references are: stage delay for wire delay; ground-truth sink slew for wire slew; and ground-truth labels for π-model parameters.

Table 2 summarizes these metrics for all the three ML models. The table shows small values of mean %error indicating transferability of the model to an unseen test design. While the values of maximum %error are large (except RISCV32I), we verify that these errors are for very short nets that have negligible wire delays and slews. The errors for RISCV32I are the least, as it has the least number of nets in the design, and it is predicted by the model trained with samples from larger designs. The standard deviation of %errors is low, indicating that few nets have large errors. For our application of post-GR timing optimization, these accuracy levels are sufficient to realize the benefits of the ML model.

Fig. 8 shows the discrepancy in wire delay and wire slew between post-GR results and post-DR results for the JPEG design in a 130nm technology. The DR deviates significantly from the GR creating discrepancies in wire delay and slew as highlighted by Fig. 8(a) and (c). The ML model uses features from GR and accurately predicts wire delays and slews as shown in Fig. 8(b) and (d). The model has much better match with post-DR delays and slews when compared to GR. The wire delays and slews are predicted with high accuracy as they track the 45 degree line in the scatter plots in Figs. 8(b), (d). The discrepancy between GR and DR wire delays or slews cannot be corrected by simply multiplying by a scaling factor as this scaling factor will differ for each design in the same technology (Fig. 2).

4.1.2 Path-level accuracy: The path delays are estimated by annotating the predicted π-model values, wire delays, and wire slews into a timing engine. We compare the path slacks across the traditional flow, ML-based flow, and the ground-truth flow. For example, Fig. 9 shows the slack comparison of JPEG in a 130nm technology. The figure on the left shows the discrepancy in path slack between GR and DR timing estimates and the figure on the right shows the

Table 2: Evaluation of the ML models using mean, maximum, and standard deviations of the absolute %error as metrics.

Designs	Tech	Metrics	Wire delay	Wire slew	C_1	R	C_2
IBEX	45nm	Mean %error	4.30	8.94	3.36	6.21	9.09
		Max %error	38.38	38.32	35.09	37.86	39.08
		Std. dev. %error	5.99	7.35	2.99	7.43	9.13
AES	45nm	Mean %error	5.12	8.23	2.60	4.88	7.76
		Max %error	39.90	39.98	33.09	28.34	35.37
		Std. dev. %error	7.46	8.17	5.75	4.50	7.56
JPEG	45nm	Mean %error	4.31	7.47	2.47	5.63	7.63
		Max %error	39.68	39.97	33.53	39.91	39.86
		Std. dev. %error	5.56	9.09	3.02	7.06	9.03
DYNAMIC NODE	45nm	Mean %error	4.21	8.19	2.24	4.95	9.07
		Max %error	39.82	39.96	22.03	38.38	44.95
		Std. dev. %error	5.00	8.18	3.05	6.18	10.76
IBEX	130nm	Mean %error	3.35	4.15	4.51	5.08	16.76
		Max %error	21.21	21.54	20.97	27.06	37.87
		Std. dev. %error	6.33	5.45	4.51	4.08	5.56
AES	130nm	Mean %error	11.15	2.46	5.83	2.96	10.38
		Max %error	39.05	29.18	37.43	29.30	37.08
		Std. dev. %error	8.03	3.15	4.71	3.05	8.61
JPEG	130nm	Mean %error	4.17	5.84	3.92	4.05	7.72
		Max %error	37.83	39.97	35.88	37.84	38.44
		Std. dev. %error	6.51	6.22	4.10	4.79	7.75
RISCV32I	130nm	Mean %error	1.54	1.15	2.79	3.39	12.64
		Max %error	3.44	2.58	18.21	35.07	31.16
		Std. dev. %error	0.67	0.83	3.48	4.04	6.06

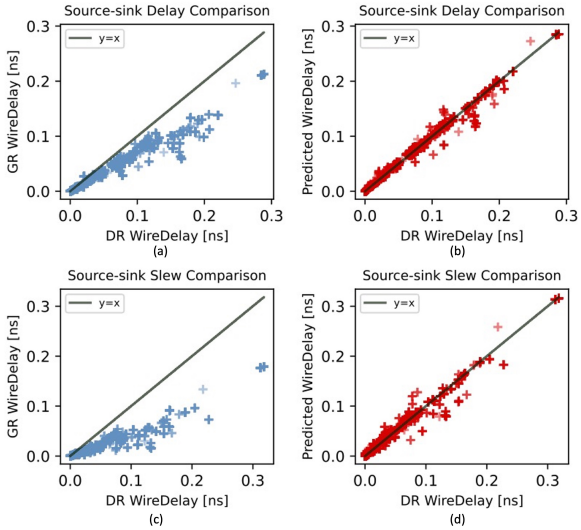


Figure 8: Wire delay and slew comparisons for JPEG 130nm.

ML-corrected path slacks versus the path slacks post-DR. With the ML-based timing correction applied after GR, the path slacks post-GR have a better match with post-DR slacks. As before, we note that the GR-based wire delays cannot be corrected using a scaling factor plus a constant because the scaling factor and constant value are different for different designs in the same technology.

In Table 3 we highlight the mean path slack %error which is defined as the mean of the absolute difference between the post-DR paths slacks and the ML-based slacks. The table compares the mean path slack %errors from the ML model against the %errors from the traditional flow across multiple designs for different utilizations or core die areas. For all the three flows, we compare the path slacks post-GR. The traditional flow has a higher %error when

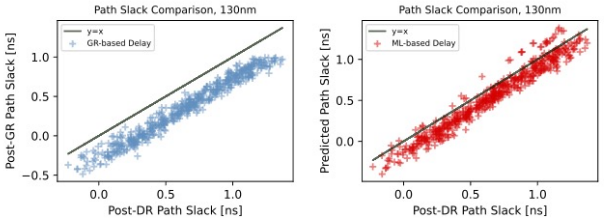


Figure 9: Path slacks comparison for JPEG 130nm.

Table 3: Post-GR paths slack errors of the traditional and ML-based flows for all designs in different utilizations.

Design	Die Size (mm^2)	Mean Path Slack Error %		Design	Die Size (mm^2)	Mean Path Slack Error %	
		Trad.	ML-based			Trad.	ML-based
IBEX clk = 2ns 45nm	0.55x0.54	2.11	0.70	**IBEX clk = 16ns 130nm**	0.67x0.67	1.09	0.64
	0.50x0.49	2.25	0.64		0.62x0.62	1.48	0.47
	0.45x0.44	1.97	0.71		0.58x0.58	0.56	0.67
	0.40x0.39	2.38	0.58		0.55x0.55	1.65	0.99
AES clk = 0.8ns 45nm	0.62x0.62	2.63	0.82	**AES clk = 5.4 ns 130nm**	0.57x0.57	1.90	1.73
	0.57x0.57	3.95	1.08		0.53x0.53	1.30	1.68
	0.52x0.52	2.20	0.87		0.50x0.50	1.32	1.23
	0.47x0.47	2.40	0.85		0.47x0.47	1.43	1.26
JPEG clk = 1.4ns 45nm	1.20x1.19	6.07	2.82	**JPEG clk = 7.8 ns 130nm**	1.15x1.15	3.20	0.82
	1.10x1.09	7.05	2.64		1.08x1.08	3.48	0.87
	1.00x0.99	5.78	2.50		1.02x1.02	3.40	1.02
	0.90x0.89	6.09	2.24		0.97x0.97	3.72	1.62
DYNAMIC NODE clk = 1ns 45nm	0.45x0.45	2.35	0.57	**RISCV32I clk = 9.6ns 130nm**	0.43x0.43	4.13	1.08
	0.40x0.40	1.45	0.63		0.40x0.40	1.65	0.39
	0.35x0.35	2.30	0.86		0.38x0.38	1.68	0.22
	0.30x0.30	1.58	1.14		0.36x0.36	2.06	0.71

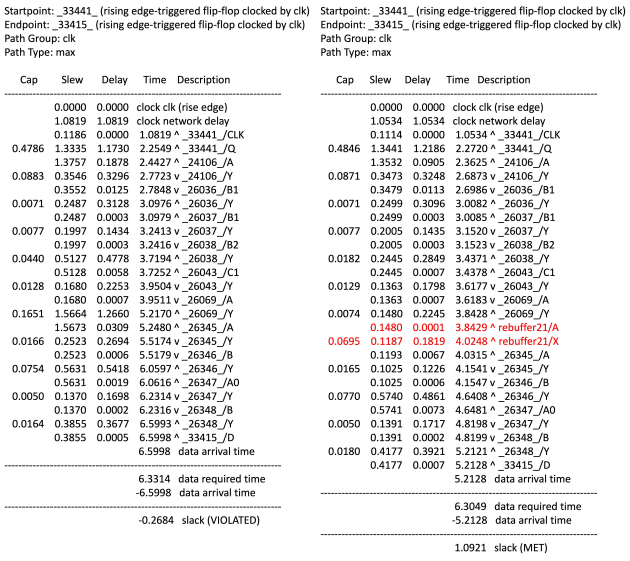
Table 4: Impact of ML-based prediction on post-DR metrics for the traditional, ground-truth and ML-based flows.

Design	Tech	#Nets	Post-DR WS (ns)			Post-DR TNS (ns)			Runtimes (s)		
			Trad.	Ground truth	ML based	Trad.	Ground truth	ML based	Trad.	Ground truth	ML based
IBEX	45nm	17566	-0.56	-0.60	-0.58	-465.14	-501.43	-479.08	7	342	17
AES	45nm	16836	-0.21	-0.19	-0.20	-27.33	-26.27	-26.44	14	718	22
JPEG	45nm	68247	-0.25	-0.17	-0.22	-79.08	-39.72	-74.20	15	712	58
DYNAMIC NODE	45nm	11598	-0.26	-0.26	-0.26	-23.96	-22.94	-23.72	4	232	16
IBEX	130nm	15307	-0.24	-0.11	-0.31	-0.24	-0.11	-0.32	5	1005	11
AES	130nm	15369	-0.27	-0.09	-0.15	-0.36	-0.17	-0.29	5	15206	16
JPEG	130nm	59573	0.24	0.02	-0.01	0.00	0.00	0.00	9	516	29
RISCV32I	130nm	8150	-0.25	-0.17	-0.16	-0.60	-0.52	-0.26	2	1569	5

compared to the ML-based flow. This indicates that on average the ML-based post-GR slacks correlate better with post-DR slacks than the traditional-flow-based slacks. This enables timing optimizations to buffer nets and resize logic gates on truly critical paths.

4.2 Impact on Post-DR Outcomes

The three ML models are applied to the physical design flow. We compare our post-DR outcomes against the other two flows in Fig. 5. Table 4 compares post-DR WS, post-DR TNS, and runtimes for the three flows. The ground-truth flow optimizes paths that are critical as per the DR-based parasitics while the traditional flow optimizes paths that may or may not be critical post-DR. For example, Fig. 10 shows a timing path from OpenSTA after DR. At left, we see the post-DR timing path from the traditional flow, and at right we list the same path from the ML-based flow. The post-GR timing optimization does not realize that the path becomes critical after



Startpoint: _33441_ (rising edge-triggered flip-flop clocked by clk)
Endpoint: _33415_ (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Cap	Slew	Delay	Time	Description
	0.0000	0.0000	0.0000	clock clk (rise edge)
	1.0819	1.0819	1.0819	clock network delay
	0.1186	0.0000	1.0819	^ _33441_/CLK
0.4786	1.3335	1.1730	2.2549	^ _33441_/Q
	1.3757	0.1878	2.4427	^ _24106_/A
0.0883	0.3546	0.3296	2.7723	v _24106_/Y
	0.3552	0.0125	2.7848	v _26036_/B1
0.0071	0.2487	0.3128	3.0976	^ _26036_/Y
	0.2487	0.0003	3.0979	^ _26037_/B1
0.0077	0.1997	0.1434	3.2413	v _26037_/Y
	0.1997	0.0003	3.2416	v _26038_/B2
0.0440	0.5127	0.4778	3.7194	^ _26038_/Y
	0.5128	0.0058	3.7252	^ _26043_/C1
0.0128	0.1680	0.2253	3.9504	v _26043_/Y
	0.1680	0.0007	3.9511	v _26069_/A
0.1651	1.5664	1.2660	5.2170	^ _26069_/Y
	1.5673	0.0309	5.2480	^ _26345_/A
0.0166	0.2523	0.2694	5.5174	v _26345_/Y
	0.2523	0.0006	5.5179	v _26346_/B
0.0754	0.5631	0.5418	6.0597	^ _26346_/Y
	0.5631	0.0019	6.0616	^ _26347_/A0
0.0050	0.1370	0.1698	6.2314	v _26347_/Y
	0.1370	0.0002	6.2316	v _26348_/B
0.0164	0.3855	0.3677	6.5993	^ _26348_/Y
	0.3855	0.0005	6.5998	^ _33415_/D
			6.5998	data arrival time
			6.3314	data required time
			-6.5998	data arrival time
			-0.2684	slack (VIOLATED)

Startpoint: _33441_ (rising edge-triggered flip-flop clocked by clk)
Endpoint: _33415_ (rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max

Cap	Slew	Delay	Time	Description
	0.0000	0.0000	0.0000	clock clk (rise edge)
	1.0534	1.0534	1.0534	clock network delay
	0.1114	0.0000	1.0534	^ _33441_/CLK
0.4846	1.3441	1.2186	2.2720	^ _33441_/Q
	1.3532	0.0905	2.3625	^ _24106_/A
0.0871	0.3473	0.3248	2.6873	v _24106_/Y
	0.3479	0.0113	2.6986	v _26036_/B1
0.0071	0.2499	0.3096	3.0082	^ _26036_/Y
	0.2499	0.0003	3.0085	^ _26037_/B1
0.0077	0.2005	0.1435	3.1520	v _26037_/Y
	0.2005	0.0003	3.1523	v _26038_/B2
0.0182	0.2445	0.2849	3.4371	^ _26038_/Y
	0.2445	0.0007	3.4378	^ _26043_/C1
0.0129	0.1363	0.1798	3.6177	v _26043_/Y
	0.1363	0.0007	3.6183	v _26069_/A
0.0074	0.1480	0.2245	3.8428	^ _26069_/Y
	0.1480	0.0001	3.8429	^ rebuffer21/A
0.0695	0.1187	0.1819	4.0248	^ rebuffer21/X
	0.1193	0.0067	4.0315	^ _26345_/A
0.0165	0.1025	0.1226	4.1541	v _26345_/Y
	0.1025	0.0006	4.1547	v _26346_/B
0.0770	0.5740	0.4861	4.6408	^ _26346_/Y
	0.5741	0.0073	4.6481	^ _26347_/A0
0.0050	0.1391	0.1717	4.8198	v _26347_/Y
	0.1391	0.0002	4.8199	v _26348_/B
0.0180	0.4177	0.3921	5.2121	^ _26348_/Y
	0.4177	0.0007	5.2128	^ _33415_/D
			5.2128	data arrival time
			6.3049	data required time
			-5.2128	data arrival time
			1.0921	slack (MET)

Figure 10: A critical path from AES 130nm after DR from the traditional flow (left) and the ML-based flow (right).

DR and hence does not include any buffers, while our ML-based flow identifies this as a critical path during post-GR optimization, and buffers the net to ensure that the path meets timing after DR. As a consequence of better optimization, we improve post-DR WS.

Out of eight designs, the ML model improves post-DR WS/TNS for five designs, indicating effective timing optimization post-GR using the ML model. Of the remaining cases, IBEX 45nm has very small WS degradation; JPEG 130nm is an improvement over "Trad." (which overcorrects WS to 0.24ps; "ML" brings it close to zero, with lower buffering/sizing cost); IBEX 130nm is worse, due to ML error that leads to the incorrect gate sizing in post-GR timing optimization and degrades the post-DR slack.

To determine the impact of ML-based post-GR timing optimizations on routability, we analyze congestion under the traditional and ML-based flows. We define a GCell to be congested if its congestion exceeds a specified threshold. Fig. 11 shows the number of congested GCells in the traditional and ML-based flows, for different thresholds. The ML-based model does not increase the number of congested GCells, indicating that it does not impact routability.

From Table 4, we see that the ML-based flow predicts post-DR timing in tens of seconds without time-intensive DR. The ML-based flow achieves comparable solutions to the ground-truth-based flow, with an average speedup of 15× over the ground-truth flow. When compared to the traditional flow, ML-based flow is 2.85× slower. However, the runtime increase when compared to the traditional flow is acceptable as the model is able to reap the benefits of the ground-truth flow.

5 CONCLUSION

We perform the first study on an ML-based method to predict post-DR timing at the post-GR stage, which is different from [7, 18] that perform timing prediction based on placement results, prior to GR. Since some features in post-GR are not available in the prior

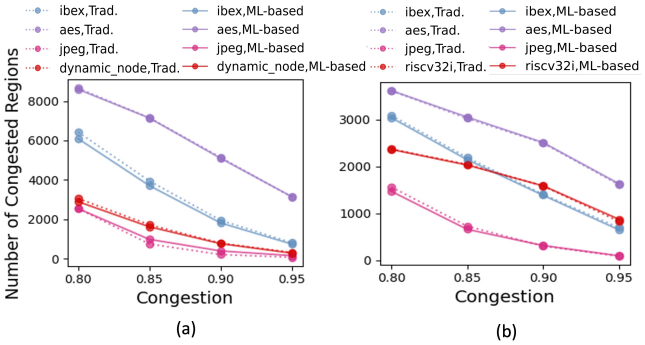


Figure 11: Number of post-DR congested regions in the traditional and ML-based flow in (a) 45nm (b) 130nm technology.

stages of design, the prediction based on post-GR is more accurate than the prediction based only on the placement. Specifically, Fig. 7 shows that the accuracy of the prediction depends on the inclusion of features such as source-sink RC and Steiner-tree-based parasitics that are only available in the post-GR phase. Our models show better accuracy of timing and parasitic estimation in post-GR than the traditional methods used in post-GR, and improve the circuit performance in post-DR efficiently without increasing congestion.

ACKNOWLEDGMENTS

This work was supported in part by DARPA HR0011-18-2-0032 (The OpenROAD Project). The work of ABK is also supported in part by NSF CCF-2122665.

REFERENCES

[1] 2022. *NanGate 45nm FreePDK and Cell Library.* https://si2.org/open-cell-library
[2] 2022. *OpenROAD.* https://github.com/The-OpenROAD-Project/OpenROAD
[3] 2022. *OpenROAD-flow-scripts.* https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts
[4] 2022. *OpenSTA.* https://github.com/The-OpenROAD-Project/OpenSTA
[5] 2022. *SkyWater 130nm PDK.* https://github.com/google/skywater-pdk
[6] T. Ajayi et al. 2019. Toward an Open-source Digital Flow: First Learnings from the OpenROAD Project. In *Proc. DAC.* 1–4.
[7] E. C. Barboza et al. 2019. Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism. In *Proc. DAC.* 1–6.
[8] W.-T. J. Chan et al. 2016. Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design. In *Proc. ASP-DAC.* 178–185.
[9] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. SIGKDD.* 785–794.
[10] H.-H. Cheng, I. H.-R. Jiang, and O. Ou. 2020. Fast and Accurate Wire Timing Estimation on Tree and Non-Tree Net Structures. In *Proc. DAC.* 1–6.
[11] C. Chu and Y.-C. Wong. 2008. FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. *IEEE T. Comput. Aid. D.* 27, 1 (2008), 70–83.
[12] S. Dolgov et al. 2019. 2019 CAD Contest: LEF/DEF Based Global Routing. In *Proc. ICCAD.* 1–8.
[13] Z. Guo, M. Liu, J. Gu, S. Zhang, D. Z. Pan, and Y. Lin. 2022. A Timing Engine Inspired Graph Neural Network Model for Pre-Routing Slack Prediction. In *Proc. DAC.*
[14] X. He, Y. Wang Z. Fu, C. Liu, and Y. Guo. 2022. Accurate Timing Prediction at Placement Stage with Look-Ahead RC Network. In *Proc. DAC.*
[15] A. B. Kahng, L. Wang, and B. Xu. 2021. TritonRoute: The Open-Source Detailed Router. *IEEE T. Comput. Aid. D.* 40, 3 (2021), 547–559.
[16] P.R. O'Brien and T.L. Savarino. 1989. Modeling the Driving-point Characteristic of Resistive Interconnect for Accurate Delay Estimation. In *Proc. ICCAD.* 512–515.
[17] M. Pan, Y. Xu, Y. Zhang, and C. Chu. 2012. FastRoute: An Efficient and High-Quality Global Router. *VLSI Des.* 2012, Article 14 (January 2012).
[18] T. Yang, G. He, and P. Cao. 2022. Pre-Routing Path Delay Estimation Based on Transformer and Residual Framework. In *Proc. ASP-DAC.* 184–189.