A Machine Learning Approach to Improving Timing Consistency between Global Route and Detailed Route

VIDYA A. CHHABRIA, Arizona State University, USA
WENJING JIANG*, University of Minnesota, USA
ANDREW B. KAHNG, University of California – San Diego, USA
SACHIN S. SAPATNEKAR, University of Minnesota, USA

Due to the unavailability of routing information in design stages prior to detailed routing (DR), the tasks of timing prediction and optimization pose major challenges. 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. To bridge the gap between GR-based parasitic and timing estimation and post-DR results *during post-GR optimization*, machine learning (ML)-based models are proposed, including the use of features for macro blockages for accurate predictions for designs with macros. Based on a set of experimental evaluations, it is demonstrated that these models show higher accuracy than GR-based timing estimation. When used during post-GR optimization, the ML-based models show demonstrable improvements in post-DR circuit performance. The methodology is applied to two different tool flows – OpenROAD and a commercial tool flow – and results on an open-source 45nm bulk and a commercial 12nm FinFET enablement show improvements in post-DR timing slack metrics without increasing congestion. The models are demonstrated to be generalizable to designs generated under different clock period constraints and are robust to training data with small levels of noise.

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

Additional Key Words and Phrases: machine learning, static timing analysis, timing optimization

ACM Reference Format:
Vidya A. Chhabria, Wenjing Jiang, Andrew B. Kahng, and Sachin S. Sapatnekar. 2023. A Machine Learning Approach to Improving Timing Consistency between Global Route and Detailed Route. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1, Article 1 (January 2023), 25 pages. https://doi.org/10.1145/3626959

1 INTRODUCTION

Interconnect effects are a significant factor in determining the post-layout performance of digital designs in modern integrated circuit technology nodes. In particular, due to increased per-unit length resistances and capacitances from

*Primary author

Authors' addresses: Vidya A. Chhabria, Arizona State University, Tempe, AZ, USA; Wenjing Jiang, University of Minnesota, Minneapolis, MN, USA; Andrew B. Kahng, University of California – San Diego, La Jolla, CA, USA; Sachin S. Sapatnekar, University of Minnesota, Minneapolis, MN, USA.

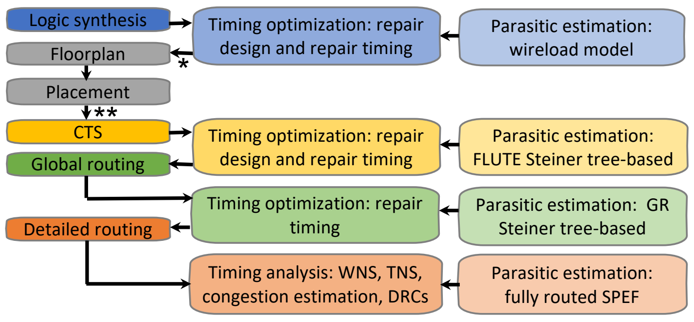

Logic synthesis
Floorplan
Placement
CTS
Global routing
Detailed routing
*
**
Timing optimization: repair design and repair timing
Parasitic estimation: wireload model
Timing optimization: repair design and repair timing
Parasitic estimation: FLUTE Steiner tree-based
Timing optimization: repair timing
Parasitic estimation: GR Steiner tree-based
Timing analysis: WNS, TNS, congestion estimation, DRCs
Parasitic estimation: fully routed SPEF

Fig. 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.

one technology generation to the next, wire delays have become a significant bottleneck in achieving design closure and overall IC performance outcomes. The accurate estimation and prediction of wire parasitics and delays is particularly important in modern design flows, as incorrect estimates can result in unnecessary design iterations; in some cases, it may be impossible to achieve design closure on schedule, which may result in increased time to market. If wire parasitics and delays can be estimated well, they can be used to guide 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. However, due to the unavailability of routing information before the final stages, perfectly accurate wire delay estimates are impossible.

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 [8] 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 that are either overdesigned (pessimistic) or underdesigned (optimistic, i.e., failing electrical and/or performance constraints).

As a design proceeds from early stages (e.g., RTL specification, floorplanning) to later stages (e.g., detailed placement, detailed routing), an increasing amount of physical information (i.e., spatial embedding) is available, leading to potentially better estimates of parasitics. However, even at relatively late stages in the physical design flow, such as routing, there can be significant estimation inaccuracy. As shown in Fig. 1, a design is typically routed in two stages: global routing (GR) and detailed routing (DR). The GR step allocates routing resources to each net, and generates a routing plan that the DR step takes as initial guidance toward a final routing solution. In modern tools, the routing plan from GR is represented in the form of *route guides* that contain information on layer assignments and Steiner tree topologies for

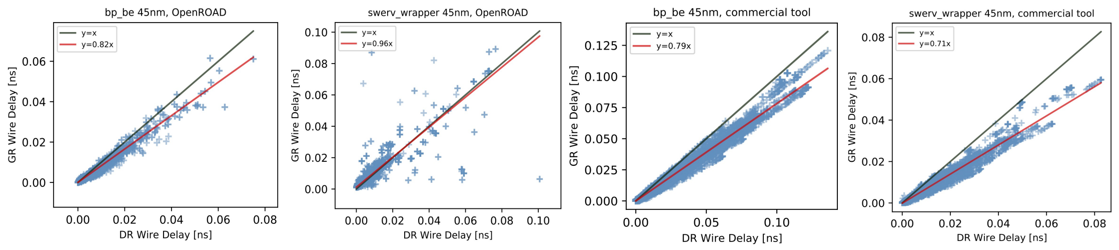


Fig. 2. Discrepancy between post-GR and post-DR wire delays of bp_be 45nm and swerv_wrapper 45nm in OpenROAD (left figures) and bp_be 45nm and swerv_wrapper 45nm in a commercial tool flow (right figures). The slope k and the intercept b in the best-fit $y = kx + b$ (red traces) differ across tool flows and across designs.

each net [9][12], and these are used to guide DR. Clearly, the ability to close timing depends on the quality of the GR route guides.

Importantly, the GR stage is typically followed by a timing optimization step (sizing, fanout clustering, buffering, etc.) as well as a final (post-optimization) placement legalization step, since better parasitic estimates are available post-GR compared to after earlier flow 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 [8] Steiner trees from FastRoute 4.1 [18], 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 delay of four designs: bp_be (42K nets) and swerv_wrapper (88K nets), each implemented in 45nm technology in both OpenROAD and commercial tool flows. Each plot shows a $y = x$ line: a perfect prediction would lie along this line; an optimistic prediction, where the GR-based delay estimate is smaller than the DR-based ground-truth delay, would lie below the line; and a pessimistic prediction would lie above the line. In OpenROAD, the GR-stage estimates are largely optimistic for bp_be, but may be either pessimistic or optimistic for swerv_wrapper. For the commercial tool flow, the GR-based wire delay estimates for the most part are optimistic. While it could be argued that the prediction can be corrected by using a best linear fit line ($y = kx + b$) instead of the $y = x$ line, this does not repair all of the discrepancies. Specifically, we draw the following conclusions:

- The slope k for calibrating the best-fit curve is tool-flow-specific, and we see different values for the OpenROAD flow and the commercial tool flow.
- Even for the same tool flow, the best-fit curve is design-specific: the value of k for bp_be is significantly different from that for swerv_wrapper. This is observed to be true for both the OpenROAD and commercial tool flows.
- Compared to the discrepancy observed in OpenROAD, the wire delay discrepancy for the commercial tool flow is lower, reflecting a superior ability to adhere to GR route guides. Even so, the discrepancy is significant.

Therefore, the problem of predicting post-DR interconnect delays from post-GR route guides is more complex than a simple correction through a curve-fit, and this motivates our approach of using an ML predictor.

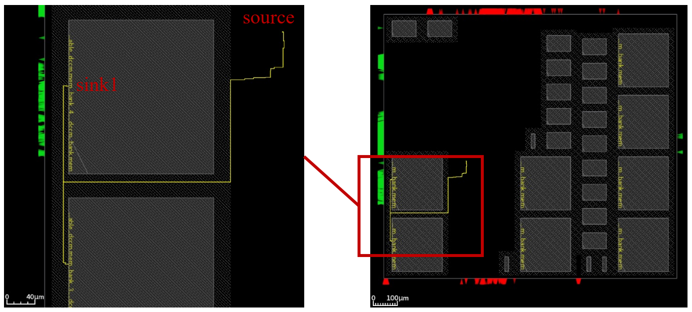
"Oracle" knowledge of post-DR parasitics improves final outcomes. As explained earlier, the use of inaccurate parasitic and timing estimates in any timing optimization can lead to harmful pessimism (overdesign that wastes

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

Design	CLK (ns)	Tech	#Nets	#Macros	Utilization	Post-DR WS (ns)	
						GR-based parasitics	DR-based parasitics
riscv32i	9.6	130nm	8150	0	67.52%	-0.26	-0.26
aes	5.4		15307	0	62.91%	-0.21	-0.19
ibex	16.0		15369	0	55.85%	-0.56	-0.60
jpeg	7.8		59573	0	62.90%	-0.25	-0.17
dynamic_node	1.0	45nm	11598	0	57.63%	-0.25	-0.17
ibex	2.0		16836	0	58.94%	-0.24	-0.11
aes	0.8		17566	0	59.60%	-0.27	-0.09
jpeg	1.4		68247	0	52.48%	0.24	0.02
swerv_wrapper	2.5		88490	28	45.87%	-0.24	-0.23
bp_fe	2.2		24883	11	43.73%	-0.15	-0.10
bp_be	2.8		41973	10	38.30%	-0.15	-0.12
swerv_wrapper	1.2	12nm	92787	28	49.08%	-0.48	-0.24
coyote	3.2		272948	15	49.00%	-0.27	-0.15

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 four open-source 130nm [20] testcases, seven open-source 45nm [13] testcases and two commercial 12nm testcases, using an open-source flow [16]. As indicated in the table, some of these layouts consist of standard cells only, with no macros, while others contain a mix of standard cells and large macro blocks. The table highlights the cost of post-GR buffering and resizing solutions that are driven by inaccurate parasitics, and also lists the utilization for each design. For example, if the discrepancy as in Fig. 2 is corrected in post-GR for each design, the improvement in the post-DR worst slack (WS) for these designs can be as much as 240ps (-0.48ns $\rightarrow$ -0.24ns). We ascribe the WS 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. In the absence of accurate prediction, these timing paths might 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. On closer inspection, we see that the designs with smaller post-DR WS improvement in Table 1 correspond to high values of utilization. For example, riscv32i has the highest utilization among 130nm designs without macros, and swerv_wrapper has the highest utilization among 45nm designs with macros. In these designs, fewer options are available for resizing and buffering in post-GR timing optimization, resulting in fewer perturbations due to these operations. This may partially explain why post-DR WNS from the flows based on GR-based and DR-based parasitics are quite close.

While significant inaccuracies are seen even for designs without macros, the presence of large macros in a design is a source of another major degree of difficulty in delay prediction. Depending on their structure, these macros may act as partial or complete blockages for interconnect wires, depending on whether some or all metal layers are blocked by the macro. The wiring detours that are required to circumvent the blockages can greatly impact the routing solution – notably, the wirelength, wire delay, and the need for buffer insertion along these wires. As an example, Fig. 3 shows a placement of swerv_wrapper in 45nm with several macros. The figure at right shows the overall layout, with a region of interest shown within a red rectangle, with two large macros placed close to each other. At left, we zoom into this rectangle, highlighting a three-pin net that connects a source pin at one side of two macros to two sinks at the other side of the macros: clearly this routing solution is required to detour around these macro blockages. From source to sink1, the wire bypasses a macro of size 206.9μm×219.8μm through the halo between two macros, and the source-sink1 wire



	Post-GR	Post-DR
Total Capacitance (pF)	0.0685	0.0893
Total Resistance (kΩ)	1.0468	1.7973
Wire delay (ns)	0.0354	0.1059

Fig. 3. Net detours due to macro blockage in swerv_wrapper, 45nm. Total capacitance, total resistance and wire delay of source-sink1 in the sample net.

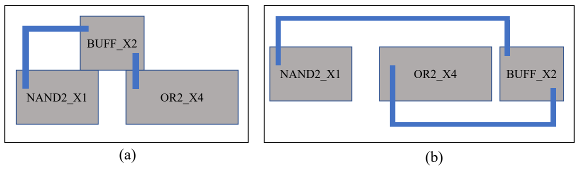


Fig. 4. The effect of different buffer locations on the routing results: (a) The inserted buffer is placed in a different row of the NAND gate and OR gate. (b) The inserted buffer is placed in the same row as the NAND gate and OR gate.

is 30% longer than the Manhattan distance between the two pins. The impact of the detour on key performance metrics for the net are listed in the table at the bottom of the figure: its post-DR capacitance and resistance are, respectively, 22.5% and 71.7% higher than those predicted at the end of GR, and its wire delay discrepancy between the post-GR and post-DR steps is 199.2%. Therefore, a critical characteristic of any prediction model must be its capability to predict interconnect delays for designs with macros.

Another practical consideration is related to the fact that EDA tools are inherently noisy [3], and can provide different results for nearly-identical inputs and runscripts. Therefore, the noisy data collected from these tools can affect the estimation of final results and may lead to incorrect buffering and sizing during optimization, thus affecting design quality. For example, a small perturbation in the location of an instance can result in a different route. This is illustrated in Fig. 4, where a buffer is inserted between a NAND gate and an OR gate during timing optimization to reduce the delay of the NAND gate. However, depending on the precise location of the inserted buffer – two possibilities are shown in Fig. 4(a) and Fig. 4(b) – the delay between the NAND gate and OR gate may be different due to dissimilarities in the wire delay. Therefore, the capability of estimation models to handle noise in the training data is another critical evaluation criterion. Due to the high cost of generating training data, it is not practical to explore the entire "noise space" around a particular training point; instead, we verify that the ML-based models we build are resilient to such noise.

Related works. Several researchers have worked in the general area of ML-based delay prediction during physical design. For a specified net topology, [6] 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 impact of macro block layout on timing is predicted using boosting and SVM in [4]. The work in [2] solves the problem of predicting wire delay and slew based on placement results, prior to GR, while [21] predicts path delays prior to routing, based on placement features, using a transformer network and residual model. The work in [11] 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 [10], a GNN model is used to estimate prerouting slacks at the endpoints of the design. In [22], wire delay and wire slew are predicted by using a graph learning architecture that encodes the information of the RC tree of a net.

Contributions. In this paper, we propose a set of techniques to enhance the accuracy of timing prediction on designs with macros in the post-GR stage to improve the quality of the DR routing solution. The key contributions of this work are as follows:

(1) We show how ML enables the fast and accurate prediction of post-DR parasitics and timing estimates using post-GR information, particularly for design with macros, in both bulk and FinFET technology nodes.
(2) We apply the ML model to the OpenROAD [1] physical design flow and to a commercial flow, and show up to 0.24ns savings (12nm node) in post-DR worst slack (WS) for OpenROAD, and 0.32ns savings (45nm node) for the commercial tool flow, without degrading congestion.
(3) We demonstrate that a similar flow that uses ML models for timing estimation can also be applied within a commercial tool flow, operating within the constraints of the information available through the available APIs, to improve the timing prediction accuracy and DR solution quality.
(4) We find that as compared to a traditional flow, with a small increase in runtime (to perform ML inference), the ML model can improve the mean %error of path slack from 5.75% to 1.15% in a 45nm testcase, and from 14.91% to 7.61% in a 12nm testcase.
(5) The proposed timing prediction models are assessed to be generalizable with respect to designs generated with different clock constraints, and can handle small noise in datasets.

A preliminary version of this research was published in [7]. This work expands upon the prior version by incorporating the impact of macros in the layout, and by extending the methodology to multiple tool flows, testing our flows on commercial technology, and evaluating multiple aspects of both the open-source and commercial flows.

2 PRELIMINARIES

2.1 Post-GR and 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 and the topology of the GR-constructed Steiner tree used for GR and the layer assignments. In this work, we use FastRoute [18], 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. For a 45nm swerv_wrapper

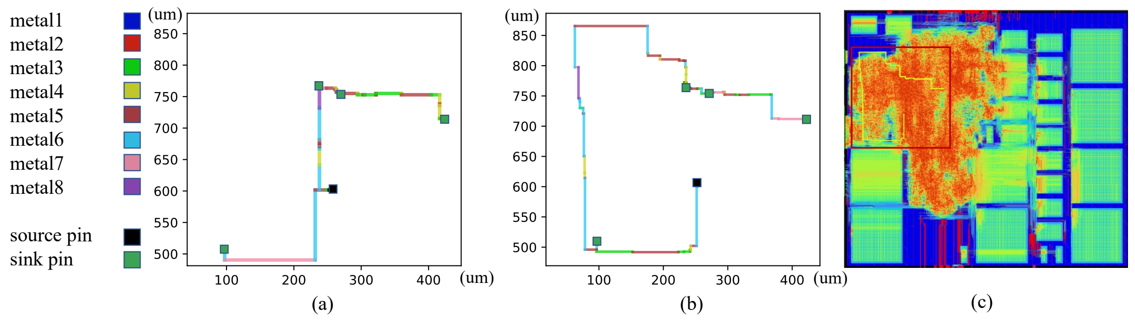


Fig. 5. Differences between the GR guides and post-DR routes of a 5-pin net from swerv_wrapper 45nm. (a) Route guides in post-GR. (b) Post-DR routes. (c) Net highlighted in congestion map.

layout, the GR and DR solutions for a 5-pin net that lies in a high-congestion region are displayed in Figs. 5(a) and (b), respectively. Fig. 5(c) superposes the net over the post-DR congestion map for the design: the red box shows the region displayed in Figs. 5(a) and (b), and the net is highlighted in yellow. It is clear that the DR solution of this net chooses a path that is significantly different from that specified in GR, and that this is due to limited available routing resources in the congested region: compared to the GR guides, the DR solution makes a large detour to either bypass the most congested region or overcome pin access issues. Due to such differences between the post-GR wire route 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. As a result, GR-stage timing estimates can be inaccurate. Moreover, 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, as seen in Fig. 2. This may mislead post-GR timing optimization steps (e.g., buffering/resizing) into performing incorrect or inaccurate optimizations.

2.2 Timing estimation

A unit operation in static timing analysis is the estimation of the delay of a single logic stage, consisting of a gate driving its fanouts through a net, as shown in Fig. 6. 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_{load} 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

$$D_{gate} = f(\tau, C_{\text{load}}). \tag{1}$$

The gate delay for an intermediate pair of (τ, C_{load}) 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, using LUTs that have the same axes as gate delay LUTs.

Since a wire is a distributed transmission line, it is modeled by the distributed RC model shown in Fig. 6, which segments the wire and creates lumped approximations for each segment. If the segments are small, this approximates the derivatives in the differential equation for a transmission line by finite sums. For a short wire, the wire resistance is dominated by the driver, and the wire can be represented by a capacitive load. Thus, the C_{load} model above can be used directly to compute the gate delay, which dominates the stage delay as the wire delay is negligible in this case. However,

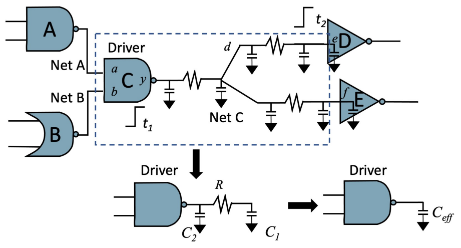


Fig. 6. The RC tree model of a net in a logic stage, and its reduction to an equivalent π-model.

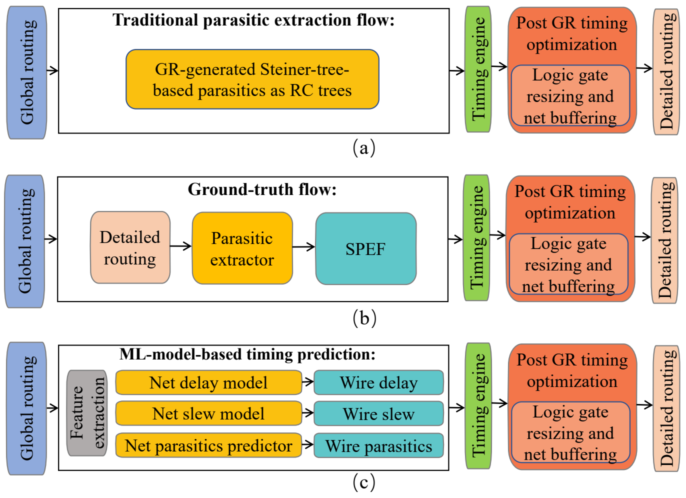


Fig. 7. Flows that use different parasitic estimates for post-GR timing optimizations: (a) traditional flow (Steiner tree-based RC estimates), (b) ground-truth (DR followed by parasitic extraction to determine post-DR parasitics), and (c) our flow (fast ML engine for post-DR parasitics and timing prediction).

for longer interconnects, resistive shielding effects can significantly impact the delay [19]. For such wires, a segmented RC model cannot directly use the LUT of Eq. (1) as the load is not purely capacitive. A typical approach to overcome this in timing analysis is by (a) generating a π-model reduction for the driving point impedance at the gate output [14], with elements R, C_1, and C_2 (Fig. 6, bottom) chosen so that the first three admittance moments of the interconnect match those of the reduced model; (b) using the π-model to find an effective capacitance, C_{eff}, that models resistive shielding; and (3) using $C_{\text{load}} = C_{\text{eff}}$ 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 constitutes 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 Timing prediction in routing flow

Fig. 7(a) highlights a typical routing flow in physical design, where the GR stage is followed by timing optimization before the final DR stage. 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 an ideal flow, shown in Fig. 7(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 in both OpenROAD and a commercial tool flow, as highlighted in Fig. 7(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: Our 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. 6, 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 gate D). Similarly, it predicts the source-sink delay between the driver pin y and sink pin f.

(ii) Source-sink wire slew: Our 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 at each of the corresponding sink pins. In Fig. 6, 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: In the OpenROAD flow, our ML model predicts post-DR π-model parasitics (R, C_1, and C_2, as shown in Fig. 6). The timing engine uses these three parameters to estimate C_{eff} which is used, in turn, to calculate gate delays. For the commercial flow, the model predicts the post-DR total load capacitance. In the OpenROAD flow, the predicted wire delay, wire slew and π-model parameters are annotated by corresponding commands through Tcl APIs, and the STA engine updates C_{eff} and gate delays based on the π-model parameters. In the commercial tool flow, equivalent commands are used to annotate the predicted wire delay and wire slew, and the parasitics are annotated. However, the available APIs do not expose the π-model parameters, and therefore, we settle for using the total load capacitance, C_{load}, for these designs. For reasons described in Section 5, this does not result in significant inaccuracies in the commercial tool flow.

Together, the above ML models (i)-(iii) 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_{load} in (1) to compute the gate delay.

3.2 ML engine

All three ML models are implemented using XGBoost [5], 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.

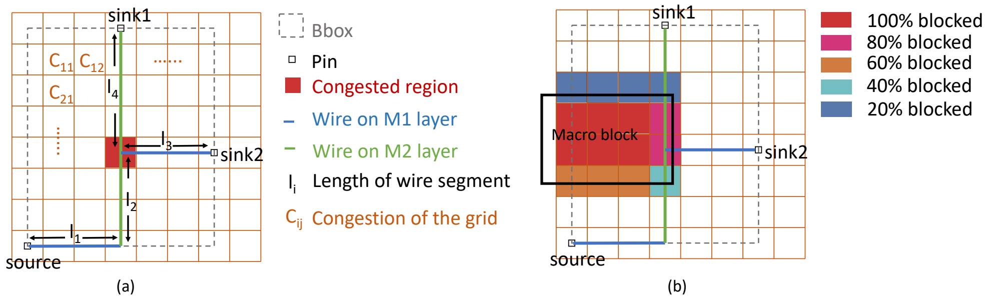


Fig. 8. A route guide for a three-pin net with a source and two sinks. (a) The net is routed using four wire segments with lengths l_1, on M1; l_2, on M2; l_3, on M1; and l_4, on M2. (b) The net traverses two GCells that are 80% blocked by a macro and two GCells that are 40% blocked and 20% blocked.

3.3 Feature engineering

3.3.1 Input features for source-sink delay and slew prediction models. We identify the following set of input features that are used to build the ML model that predicts the source-sink delay and slew of a net. The first set of features described below are derived on a per-net basis, while another set of features, described later, are on a per-sink basis.

HPWL: This feature (half-perimeter of the minimum bounding box of the net) is illustrated by the dashed bounding box in Fig. 8(a). It provides a lower bound on the wirelength.

Number of sinks: It is well known that for nets with numerous sinks, the HPWL can be a significant underestimate. 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. We therefore pass this feature, which is extracted from the gate-level netlist, to the ML model.

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

Congestion estimates: As illustrated in Fig. 5, 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. 8(a).

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 use 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. 8(a), 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 strongly reflected in 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 give 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-to-sink wire segments. In Fig. 8(a), 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 $(l_1 \times C_{M1} + l_2 \times C_{M2} + l_4 \times C_{M2})$.
Routing blockage due to macros: Since macro blocks squeeze the space available for routing, any nets traversing the GCells blocked by macros must detour around these GCells, with probability that is a function of the degree of blockage; the detouring resutls in increased wire delay, total wire resistance, and total wire capacitance. As shown in Fig. 8(b), the path from the source to sink1 traverses four GCells that are blocked by macros: two of these GCells are 80% blocked, one is 40% blocked, and one is 20% blocked. Due to limited routing resources, it is likely that the wire connecting source and sink1 will bypass the GCells that are 80% blocked in the final routing result. The features associated with the macros must capture the variation in GCell blockage over the regions traversed by a net. This may be achieved in several ways, e.g., (1) using the mean of the percentage blockage of the GCells in the bounding box of an entire net; (2) using the maximum percentage blockage of the GCells in a source-to-sink bounding box; or (3) using the mean of the percentage blockage of the GCells in the bounding box for a source-sink pair.

However, for a net with multiple sinks, the first option using the mean blocking percentage of the GCells in the net bounding box can be misleading: it does not estimate the macro impact accurately when one source-sink path goes through a highly blocked GCell (e.g., >80% blocked) but other source-sink paths traverse less blocked or unblocked GCells. The second option, using the maximum blocking percentage of the GCells in the source-sink bounding box, could mispredict the wire detour when only a small number of GCells are largely blocked by macros. Therefore, we base our approach on the third option, using the mean blocking percentage of all GCells in the source-sink bounding box, to estimate the impact of macro blockage on the wiring inside the source-sink bounding box.

Furthermore, we observe that even this approach is imperfect. Notably, a detoured route may traverse GCells outside of the bounding box, and the macro blockages in these GCells outside the bounding box will also affect the routing result. Hence, the average blocking percentage in an expanded bounding box is introduced as a feature to estimate the macro impact, where the expanded bounding box is scaled by scaling factor $\alpha > 1$ over the source-sink bounding box. Empirically, we find that $\alpha = 2$ is a good choice and we therefore use an expanded bounding box with $\alpha = 2$, i.e., doubling the length and width of the bounding box, to determine the mean percentage blockage that is used as a feature in the ML model.

3.3.2 Input features for the load prediction model. For the ML model that predicts the parameters of the π-model at the driving point for use with the OpenROAD timer, we use the HPWL, number of sinks, congestion estimates and macro blockage 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 [14] to the GR-generated Steiner tree. Since the timing engine of the commercial tool uses C_{load} instead of C_{eff}, and does not provide visibility into the parameters R, C_1, and C_2 of the π-model, our best option is to use HPWL, number of sinks, congestion estimates and macro blockage estimates, along with C_{load} based on the GR result, as model features for the commercial tool flow.

4 MODEL TRAINING AND INFERENCE IN THE PHYSICAL DESIGN FLOW

4.1 Ground-truth data generation and model training

Our ground-truth data is generated using the flow highlighted in Fig. 7(b). Since our experiments are performed in different tool flows, we use both a branch of OpenROAD-flow-scripts [16] and a commercial flow. The ground-truth data

Table 2. Summary of designs implemented in 45nm, 12nm, and 130nm technology nodes.

Design	Tech	# Nets	Macros
ibex	45nm	17566	0
aes		16836	0
jpeg		68247	0
dynamic_node		11598	0
swerv_wrapper		88490	28
bp_fe		24883	11
bp_be		41973	10
swerv_wrapper	12nm	92787	28
coyote		272948	15
ibex	130nm	15307	0
aes		15369	0
jpeg		59573	0
riscv32i		8150	0

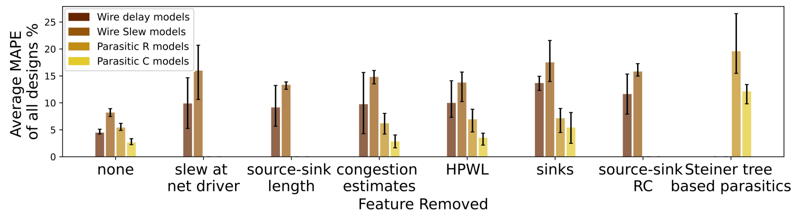
is generated for multiple designs implemented in two open-source bulk CMOS technologies (a 45nm technology [13]) and a 130nm technology [20], and one commercial FinFET technology. All of the tested designs are summarized in Table 2.

To increase the diversity of our training data set, we run the ground-truth flow for each design with three 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 placements and detours. We extract the features described in Section 3.3 and their corresponding ground-truth labels. The training labels for the three ML models are extracted after timing analysis (corresponding to the green box in Fig. 7(b)) including the source-sink wire delays, source-sink wire slews, and load parameters. Each source-sink pair represents a single data point for the source-sink delay and slew at the sink. For the load capacitance predictor, each net in the design is a single data point. The training data is normalized before training; however, in practice we observe that the XGBoost model is not very sensitive to this normalization and performs adequately using skewed data. The models for designs with macro blocks and without macro blocks are trained separately. For designs without macros, our ground-truth dataset contains 456,661 data points in 45nm technology, 348,429 data points in 12nm technology, and 394,162 data points in 130nm technology. For designs with macros, our ground-truth dataset contains 880,225 data points in 45nm, and 643,793 data points in 12nm. The pace of ground-truth data generation is slow (1 hour per design, on average) but this 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.

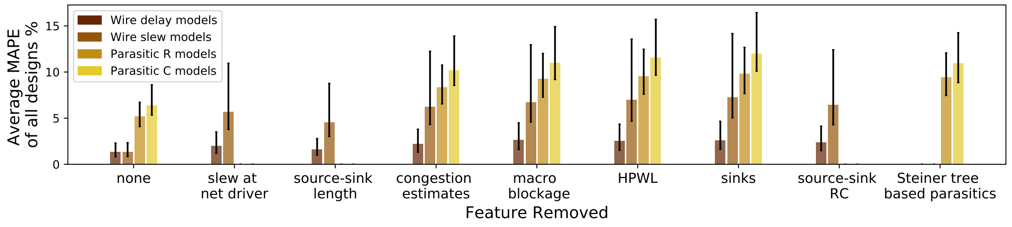
The model is trained using root mean squared error (RMSE) as the loss function. For the XGBoost regressor [5], we use a learning rate 0.01. We choose the maximum tree depth = 4, the number of estimators = 900, and the subsampling ratio = 0.8. The hyperparameter values are chosen based on a full search of a discrete grid on the domain of the hyperparameters: the assignment with the highest score is used for parameter tuning. We find that the models are not sensitive to small changes in the hyperparameter values.

4.2 ML inference in physical design flow

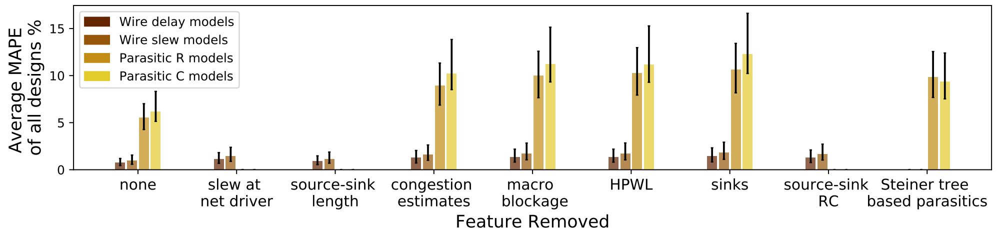
The trained ML models are applied to the flow shown in Fig. 7(c). At the post-GR phase, the ML model features are extracted from the design data and route guides. Then, the features are fed into the three ML models, which perform



(a) Average MAPE on designs without macros in 45nm.



(b) Average MAPE on designs with macros in 45nm.



(c) Average MAPE on designs with macros in 12nm.

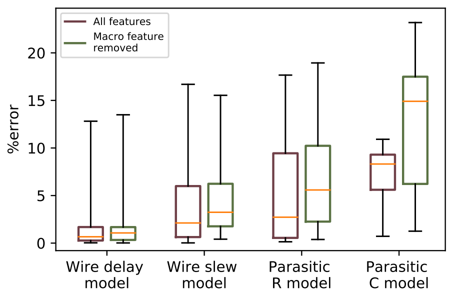
Fig. 9. Feature sensitivity analysis showing the average mean absolute percentage error (MAPE) (y-axis), for each removed feature (x-axis), for all 45nm designs and all 12nm designs.

a fast and accurate inference to predict source-to-sink wire delays and wire slews, as well as π-model parameters in OpenROAD or effective load capacitance in the commercial tool flow. The predicted estimates are annotated via Tcl APIs in the STA engine: load parameters are used by the timing engine to estimate the gate delay, while the source-sink wire delays and slews are directly used as 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.

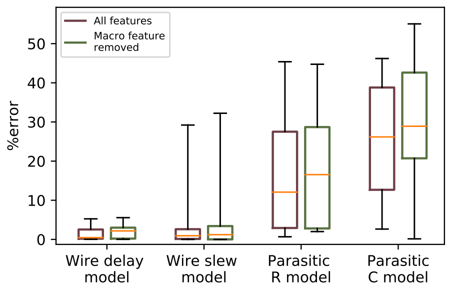
As confirmed in our experimental studies (Seciton 5 below), the ML-based inference flow provides an accurate estimate of post-DR timing without performing time-intensive DR. These estimates are useful for efficient buffering and resizing, as the optimizer now has improved knowledge of the truly (post-DR) critical paths.

4.3 Feature sensitivity

To demonstrate that each selected feature is indispensable to the model, we perform a sensitivity analysis for each feature, for each of the four models (wire delay, wire slew, parasitic R, and parasitic C) from Section 3.1. Note that not all models use all features; therefore, there are missing bars for certain features in the figure. Fig. 9 performs an ablation



(a) Macro feature sensitivity for 45nm designs.



(b) Macro feature sensitivity for 12nm designs.

Fig. 10. Macro feature sensitivity for wires that have area in a bounding box that with >80% blockage.

study on the set of features, removing one feature at a time and examining its impact on the accuracy of the ML model for designs with macros. The analysis of the ML model for designs without macros was conducted in our previous conference version [7]. For example, for source-sink wire delay prediction, we measure the test accuracy for wire delay prediction models, each trained by removing one specific feature at a time. Similar experiments are also performed on the wire slew model, parasitic R model, and parasitic C model. The y-axes of Figs. 9(a)-(c) respectively show the average of the mean average percentage errors (MAPEs) over all 45nm designs without macros, all 45nm designs with macros, and all 12nm designs with macros. The x-axis of each figure lists the feature that has been removed. 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. For the 12nm model, which has lower parasitics, the errors approximately double with removal of slew and source-sink features, but since the baseline ("None") is low, the error after removal is also low. Nevertheless, we keep these features in the model to enable generality.

We further examine the macro feature sensitivity in Fig. 10 for the four ML models. The x-axis of the figure lists the models trained with the macro features, and the y-axis highlights the average percentage error of the nets in all 45nm designs and all 12nm designs. Nets whose expanded bounding box has a large overlap with macro blockages are much more greatly affected by macro features than those that are more distant. For these nets, we plot data for source-sink pairs that have 80% of their source-sink bounding box area blocked by macros. Fig. 10 shows that introducing macro features improves the average absolute %error from (1.87%, 3.24%, 5.58%, 14.90%) to (1.41%, 2.11%, 2.72%, 8.31%) for 45nm designs, where the four percentages correspond to average absolute %error of the four models in the figure. For 12nm designs, the corresponding numbers in the average absolute %error show an improvement from (2.17%, 1.21%, 16.56%, 28.90%) to (0.45%, 0.94%, 12.07%, 26.17%).

5 EXPERIMENTAL SETUP AND EVALUATION

Our experiments are performed on benchmarks from OpenROAD across three technologies: two open technologies, NanGate 45nm [13] and SkyWater 130nm [20], and one 12nm commercial technology. We perform physical design using two flows: with OpenROAD and a commercial EDA tool flow. Our target designs fall into two classes: those without macros (IBEX, AES, JPEG and Dynamic Node in 45nm; IBEX, AES, JPEG and RISCV32I in 130nm; and IBEX, AES and JPEG in 12nm) and those with macros (swerv_wrapper, bp_fe and bp_be in 45nm; and swerv_wrapper and coyote in

Table 3. Prediction error of the shared ML model, and the separate ML models for designs with macros and designs without macros in 45nm.

Designs	**Clock period**	**# Macros**	**Wire delay**						**Wire slew**					
			ML model (separate)			**ML Model (shared)**			**ML model (separate)**			**ML Model (shared)**		
			Mean	Max	Stdev.	Mean	Max	Stdev.	Mean	Max	Stdev.	Mean	Max	Stdev.
swerv_wrapper	2.5ns	28	0.84%	115.19%	2.78%	2.95%	149.41%	4.83%	0.04%	19.81%	0.34%	0.55%	22.72%	0.61%
bp_fe	2.2ns	11	0.64%	19.14%	0.91%	2.76%	32.34%	2.78%	0.04%	4.42%	0.13%	0.47%	6.46%	0.65%
bp_be	2.8ns	10	2.16%	34.58%	3.42%	2.93%	48.90%	3.06%	0.07%	9.15%	0.25%	0.49%	8.85%	0.59%
dynamic_node	1.0ns	0	4.21%	39.82%	5.00%	6.60%	53.29%	7.58%	8.19%	39.96%	8.18%	8.96%	67.78%	7.60%
ibex	2.0ns	0	4.30%	38.38%	5.99%	4.93%	35.33%	3.54%	8.94%	38.32%	7.35%	9.51%	50.48%	7.06%
aes	0.8ns	0	5.12%	39.90%	7.46%	7.35%	52.59%	10.43%	8.23%	39.98%	8.17%	14.88%	65.03%	12.62%
jpeg	1.4ns	0	4.13%	39.68%	5.56%	8.41%	54.71%	8.01%	7.47%	39.97%	9.09%	10.54%	46.30%	8.12%

12nm). As compared to the preliminary version of this paper [7], where the results were shown only on bulk nodes, we now show results here on a FinFET technology node; we apply the approach using not only OpenROAD, but also using a commercial tool flow; and we expand the algorithm to address designs with macros. In addition, we test the ability of our models to generalize with respect to clock periods by running inference on unseen designs generated by using different clock constraints, and we also test the robustness of our models to noise by adding Gaussian noise to training datasets. As detailed below, these comprehensive experiments confirm the general applicability of our approach.

The ML models for designs with and without macros are trained separately. In Table 3, we compare the prediction results of *one shared* model for both designs with and without macros, versus the results of *two separate* models for designs with macros and without macros. For the training of the shared model, we leave out each test design from all designs in 45nm and use the remaining designs for training. The table shows that the model trained specifically for designs with macros or designs without macros has better accuracy than a shared model trained for all designs. We explain this by observing that macro-induced detours and blockages pose specific challenges due to the 100% blockage and the large sizes of the macros, which cannot be easily captured in the shared model. Therefore, building separate models is better than using a shared model. For designs without macros, the models tested on each design are trained using data from the remaining designs without macros in the same technology. For designs with macros, we are handicapped by the limited number of available designs: i.e., only three designs in the 45nm node and two in the 12nm node. Therefore, the models are trained by using the data from other designs with macros at various utilization settings, and also from the same design, but for different utilization settings. Thus, even with this limited dataset, we ensure that the ML model is evaluated for an unseen design having a different utilization than the set of designs in the training set.

Training and inference for the ML model are implemented in Python 3.6 and performed on a machine with Intel Xeon Silver 4214 CPU @2.2GHz and NVIDIA A100 PCIe 40GB GPU. For the OpenROAD flow, the predicted parasitics, wire delays, and wire slews are annotated into the timing engine by modifying the OpenROAD [15] and OpenSTA [17] source code. For the commercial flow, the predicted values are annotated into the timing engine through available Tcl commands in the commercial tool flow.

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. To highlight the importance of incorporating macro-based features, we also compare the results from the new flow, using a macro-based ML model, to a flow similar to our preliminary work [7], where the models are trained without macro features.

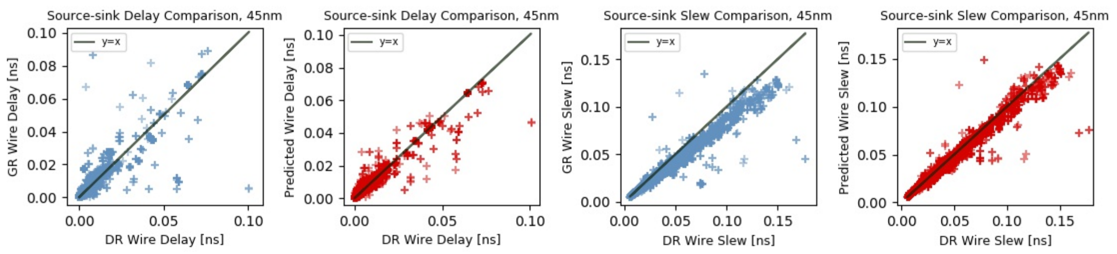


Fig. 11. Wire delay and slew comparisons for swerv_wrapper 45nm.

5.1 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-net/per-sink 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.

5.1.1 Net-level accuracy. Table 4 summarizes the metrics for the ML models and the accuracy for 45nm designs and 12nm designs in both OpenROAD and the commercial tool flow, and 130nm designs in OpenROAD, evaluated on designs with and without macros. We use the mean, maximum, and standard deviation of the absolute %error as metrics for evaluation, where the absolute percentage error is the absolute difference between the ground-truth and the predicted value with respect to a reference. The references that we use are the stage delay for evaluating wire delay; the ground-truth sink slew for wire slew; and the ground-truth labels for parasitics.

The table shows better mean %error and maximum %error metrics for our ML-based wire delay and wire slew predictions compared to GR Steiner-tree-based estimation from both OpenROAD and the commercial tool flow for most designs. In general, the ML-based model shows significant reductions in the %error. In only two cases, denoted in red, the error for our approach is worse than the GR-based approach, but we have verified that these errors are from short nets that have negligible wire delay and small stage delay, and therefore do not affect the critical path in the circuit.

In some cases in the table, the maximum %error is very large (e.g., over 100% for swerv_wrapper 45nm). We have confirmed that this error can also be attributed to a short wire that also has a small stage delay as a reference. Scatter plots of the wire delay and wire slew comparisons in ns, without normalization, are presented in Fig. 11, showing a more complete picture of the match between ML prediction and the ground truth, relative to the GR-based approach. It can be seen that the match for the ML predictor is considerably better than for the GR-based approach (even the outlier at the right of the second plot is significantly closer to the $x = y$ line than for the GR-based case). Since the large errors are only in cases where the ground-truth wire delay and wire slew are very small, the path delay is not greatly affected by these errors. From the table and the scatter plots, it can be seen that the standard deviation of %error from ML models is also lower, indicating that few nets have larger errors. For our application of post-GR timing optimization, these accuracy levels are sufficient to realize the benefit of the ML models.

It is worth noting that the accuracy of the commercial-tool-flow-based flows is generally lower than that of the OpenROAD-based flow. The reason why OpenROAD provides better accuracy is because we have visibility into the full π-model, while the commercial tool API only provides a view of the total C_{load}. However, the error in the commercial flow is still acceptable. To examine this further, we show histograms of the distribution of wirelengths for a representative design, swerv_wrapper in 45nm, under both flows. Fig. 12 shows the number of nets with lengths greater than 100μm

Table 4. Evaluation of the ML models using mean, maximum, and standard deviation of the absolute %error as metrics.

Tool	Designs	Clock period	Technology node	Wire delay						Wire slew						Path delay					
				ML-based			GR-based			ML-based			GR-based			ML-based			GR-based		
				Mean	Max	Stdev.	Mean	Max	Stdev.	Mean	Max	Stdev.	Mean	Max	Stdev.	Mean	Max	Stdev.	Mean	Max	Stdev.
OpenROAD	swerv_wrapper*	2.5ns	45nm	0.84%	115.19%	2.78%	4.27%	149.41%	7.28%	0.04%	19.81%	0.34%	0.20%	40.63%	1.22%	0.34%	4.62%	0.30%	1.48%	7.54%	0.69%
	bp_fe*	2.2ns		0.64%	19.14%	0.91%	3.63%	15.68%	4.65%	0.04%	4.42%	0.13%	0.24%	5.83%	0.98%	0.65%	1.91%	0.41%	0.69%	2.47%	0.52%
	bp_be*	2.8ns		2.16%	34.58%	3.42%	3.00%	55.64%	3.88%	0.07%	9.15%	0.25%	0.12%	10.45%	0.38%	0.63%	2.21%	0.45%	0.80%	2.57%	0.34%
	dynamic_node	1.0ns		4.21%	39.82%	5.00%	8.43%	51.33%	6.45%	8.19%	39.96%	8.18%	10.56%	51.51%	10.54%	0.57%	5.84%	0.32%	2.35%	10.76%	2.43%
	ibex	2.0ns		4.30%	38.38%	5.99%	6.54%	49.47%	9.62%	8.94%	38.32%	7.35%	11.52%	49.40%	9.47%	0.70%	6.64%	0.42%	2.11%	10.66%	1.18%
	aes	0.8ns		5.12%	39.90%	7.46%	11.60%	68.43%	9.62%	8.23%	39.98%	8.17%	10.61%	54.53%	10.53%	0.82%	4.21%	0.30%	2.63%	11.04%	1.72%
	jpeg	1.4ns		4.13%	39.68%	5.56%	5.32%	57.15%	7.17%	7.47%	39.97%	9.09%	9.63%	51.52%	11.72%	2.82%	10.60%	2.03%	6.07%	29.78%	4.28%
	swerv_wrapper*	1.2ns	12nm	0.87%	22.80%	0.93%	4.26%	44.95%	5.76%	0.04%	29.81%	0.17%	0.70%	71.75%	3.85%	1.15%	7.61%	1.26%	5.75%	14.91%	4.77%
	coyote*	3.2ns		0.60%	26.58%	0.88%	3.47%	39.65%	4.73%	0.22%	40.13%	0.62%	1.97%	51.34%	4.51%	1.09%	2.65%	0.27%	1.23%	7.73%	1.51%
	ibex	16.0ns	130nm	3.35%	21.21%	6.33%	4.32%	27.34%	8.16%	4.15%	21.54%	5.45%	5.35%	36.77%	7.03%	0.64%	8.17%	1.05%	1.09%	4.99%	1.13%
	aes	5.4ns		11.15%	39.05%	8.03%	25.37%	72.34%	10.35%	2.46%	29.18%	3.15%	3.17%	65.61%	4.06%	0.47%	3.22%	0.40%	1.48%	7.77%	0.53%
	jpeg	7.8ns		4.17%	37.83%	6.51%	14.38%	48.76%	8.39%	5.84%	39.97%	6.22%	7.53%	67.52%	8.02%	0.82%	6.87%	0.65%	3.20%	12.65%	2.31%
	riscv32i	9.6ns		1.54%	3.44%	0.67%	2.99%	8.43%	0.86%	1.15%	2.58%	0.83%	1.48%	3.33%	0.84%	1.08%	2.85%	0.42%	4.13%	19.90%	3.27%
Commercial tool	swerv_wrapper*	2.2ns	45nm	7.21%	63.59%	6.50%	29.80%	70.56%	11.56%	0.24%	9.14%	0.71%	1.46%	50.93%	5.28%	0.35%	5.49%	0.53%	1.00%	5.94%	0.60%
	bp_fe*	1.6ns		5.88%	43.79%	4.96%	24.41%	51.07%	9.26%	0.36%	6.08%	0.79%	3.73%	37.84%	8.83%	0.54%	4.81%	0.46%	1.64%	7.05%	0.67%
	bp_be*	2.0ns		6.09%	57.99%	5.46%	25.84%	56.00%	9.29%	0.36%	19.30%	0.83%	3.04%	99.94%	7.84%	0.60%	5.08%	0.47%	1.65%	6.28%	0.50%
	swerv_wrapper*	1.0ns	12nm	6.24%	36.47%	1.27%	12.30%	56.30%	2.78%	0.36%	26.26%	0.95%	0.85%	79.14%	2.78%	3.52%	8.28%	2.29%	5.98%	13.69%	3.24%
	coyote*	3.2ns		5.22%	83.62%	1.05%	15.42%	98.70%	2.99%	3.68%	59.35%	0.88%	11.75%	93.02%	2.86%	2.22%	7.36%	1.74%	4.67%	10.85%	2.87%

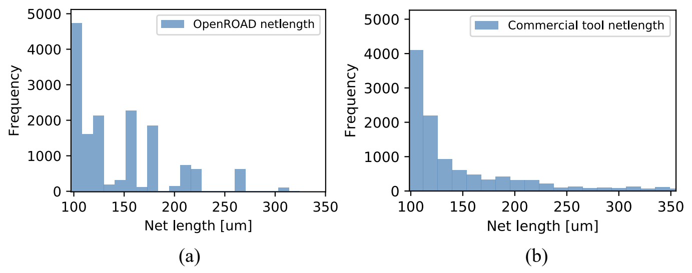


Fig. 12. Wirelength distribution of swerv_wrapper 45nm: (a) for a design with size of $1.20 \times 1.09mm^2$ generated in OpenROAD, and (b) for a design with size of $1.18 \times 1.18mm^2$ generated in a commercial tool flow.

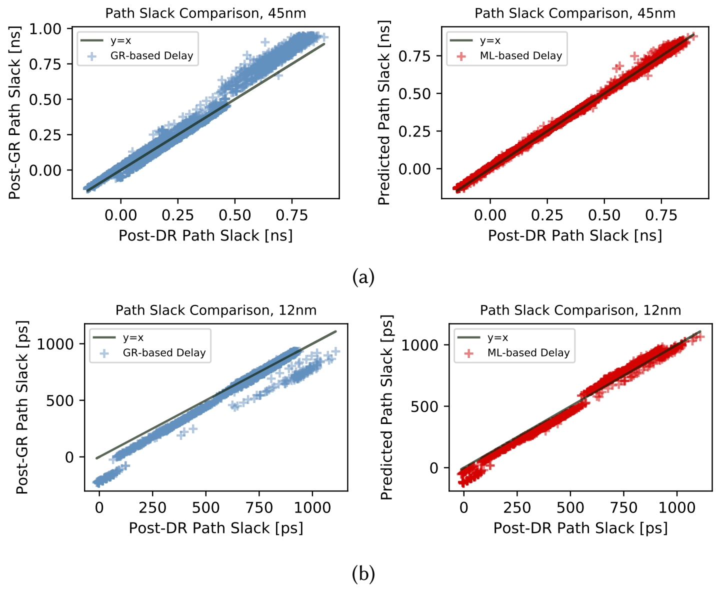


Fig. 13. Path slack comparison for (a) swerv_wrapper 45nm and (b) coyote 12nm.

under both flows. From the histograms, it is apparent that the designs generated by the commercial flow have fewer long wires as shown in Fig. 12; this can be attributed to lower utilizations for OpenROAD *vis a vis* the commercial tool flow. Due to these shorter wirelengths, the simpler C_{load} feature is adequate to capture the wire capacitance, and is supplemented by the source-to-sink wirelength feature that acts as a surrogate for a wire resistance feature. For the OpenROAD-based flow, with longer wires, we find that the use of these features results in significant accuracy loss, and that the π-model features are required to achieve the accuracy levels shown in Table 4.

5.1.2 Path-level accuracy. We analyze the accuracy of the predictor in estimating the delays of a sample of paths in the circuit. These path delays are estimated by annotating the predicted parasitic values, wire delays, and wire slews into a timing engine. We compare the path slacks across the traditional flow and the ML-based flow. Specifically, we analyze paths whose slack is below 40% of the clock period, considering one worst-case path for each endpoint.

Table 5. Impact of ML-based prediction for designs with macros on post-DR metrics for traditional, ground-truth and ML-based flows in OpenROAD.

Design	Tech	Die size (mm^2)	Utilization	Post-DR WS (ns)				Post-DR TNS (ns)				Runtimes (s)		
				Trad.	ML based	[7]	Ground Truth	Trad.	ML based	[7]	Ground Truth	Trad.	ML based	Ground Truth
swerv_wrapper CLK=2.5ns	45nm	1.20x1.09	45.87%	-0.14	-0.14	-0.13	-0.13	-23.45	**-21.96**	-22.78	-22.58	1629	1657	1995
		1.36x1.20	39.22%	-0.20	-0.20	-0.20	-0.20	-38.40	-37.22	-37.69	-35.03	1601	1630	1881
		1.55x1.34	30.71%	-0.24	-0.23	-0.23	-0.21	-57.22	**-43.22**	-50.16	-49.16	1588	1613	1898
bp_fe CLK=2.2ns		0.78x0.63	43.73%	-0.15	-0.10	-0.13	-0.10	-1.60	-0.95	-1.40	-0.72	700	707	891
		0.84x0.68	37.45%	0.01	0.02	0.01	0.02	0.00	0.00	0.00	0.00	619	628	681
		0.99x0.79	27.09%	-0.01	**0.02**	-0.01	0.00	-0.01	0.00	-0.03	0.00	598	604	709
bp_be CLK=2.8ns		0.75x0.75	41.82%	-0.18	-0.17	-0.17	-0.15	-9.63	-7.82	-7.92	-6.60	792	798	1217
		0.79x0.78	38.30%	-0.15	-0.12	-0.12	-0.10	-10.86	-6.81	-7.76	-4.15	768	774	988
		0.98x0.92	25.95%	-0.09	**-0.07**	-0.09	-0.10	-2.30	-2.10	-2.63	-2.09	622	629	889
swerv_wrapper CLK=1.2ns	12nm	0.64x0.48	49.08%	-0.48	**-0.24**	-0.32	-0.36	-207.68	-202.39	-204.37	-200.55	6041	6060	13861
		0.70x0.54	39.86%	-0.35	**-0.23**	-0.27	-0.33	-364.02	-343.81	-347.05	-337.56	5418	5433	12072
		0.80x0.64	29.39%	-0.21	-0.18	-0.20	-0.18	-306.18	**-234.98**	-284.44	-237.94	4846	4865	10900
coyote CLK=3.2ns		0.66x0.66	49.08%	-0.02	0.05	0.04	0.09	-0.14	0.00	0.00	0.00	7243	7251	13405
		0.75x0.75	35.87%	-0.27	-0.15	-0.19	-0.14	-24.73	**-12.93**	-20.50	-14.06	6029	6038	10883
		0.85x0.85	27.91%	-0.08	-0.05	-0.06	-0.04	-1.21	-0.94	-1.04	-0.75	6054	6064	11465

Fig. 13 shows an example of the slack comparison for (a) swerv_wrapper in 45nm technology and (b) coyote in 12nm technology. The figures on the left show the discrepancy in path slack between GR and DR timing estimates and the figures on the right show the ML-corrected path slacks versus the post-DR path slacks. With the ML-based timing correction applied after GR, the post-GR path slacks have a better match with post-DR slacks.

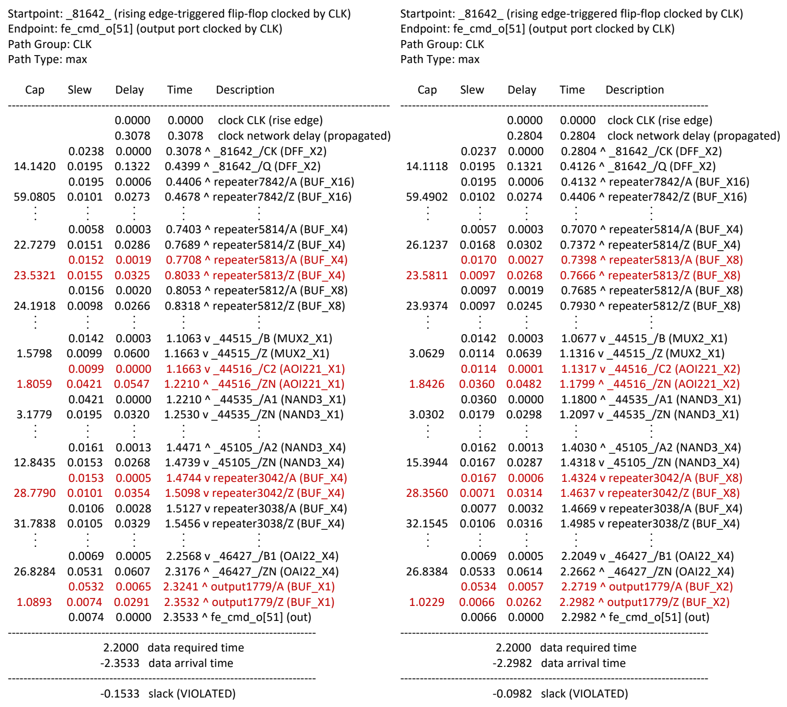
The last six columns in Table 4 compare the mean, maximum and standard deviation of path delay %errors from the ML model against the %errors from the traditional flow for multiple designs in both OpenROAD and the commercial tool flow. The mean path delay %error is defined as the mean of the absolute percentage difference between the ML-based delays and the post-DR path delays, using the clock constraint for each design as the (normalization) reference. The traditional flow has a higher mean, maximum, and standard deviation of %error when compared to the ML-based flow, with just one exception – the standard deviation of bp_be 45nm in OpenROAD is slightly worse than with the traditional flow. This indicates that on average the ML-based post-GR delays correlate better with post-DR delays than the traditional-flow-based delays. The path delay accuracy is improved by our approach, with the average error reducing from 2.50% to 1.11%. This enables timing optimizations to buffer nets and resize logic gates on truly critical paths.

5.2 Impact on post-DR outcomes

The three ML models are applied to the timing analysis step before post-GR timing optimization transforms, based on sizing and buffering, in the physical design flow. We compare our post-DR outcomes against the flows in Fig. 7(a) and (b). Tables 5 and 6 compare post-DR WS and post-DR TNS for the four flows applied in OpenROAD and in a commercial tool. 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. 14 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 have an accurate estimation of the path that becomes critical after DR and hence does not have correct resizing or buffering, while our ML-based flow identifies this as a critical path during post-GR optimization, and resizes the gate and buffers the net to ensure that the path has better timing after DR. As a consequence of better optimization, we improve post-DR WS.

Table 6. Impact of ML-based prediction on post-DR metrics for traditional, ground-truth and ML-based flows in a commercial tool.

Design	Tech	Die size (mm^2)	Utilization	Post-DR WS (ns)			Post-DR TNS (ns)			Runtime(s)		
				Trad.	ML based	Ground Truth	Trad.	ML based	Ground Truth	Trad.	ML based	Ground Truth
swerv_wrapper CLK=2.2ns	45nm	1.18x1.18	48.85%	-0.04	-0.02	-0.01	-0.22	-0.12	-0.01	629	661	2435
		1.25x1.25	40.76%	0.00	0.00	0.00	0.00	0.00	0.00	501	533	1995
		1.44x1.44	36.91%	-0.08	-0.08	-0.04	-0.89	-0.52	-0.40	507	602	2243
bp_fe CLK=1.6ns		0.63x0.63	50.72%	-0.29	-0.24	-0.01	-10.96	-8.75	0.00	257	265	867
		0.71x0.71	41.77%	-0.31	-0.15	-0.04	-11.96	-3.32	-0.10	317	325	1216
		0.82x0.82	30.76%	-0.41	-0.09	-0.03	-14.46	-1.20	-0.30	254	262	1009
bp_be CLK=2.0ns		0.66x0.66	53.26%	-0.04	-0.04	-0.01	-2.65	-1.20	-0.04	401	415	1659
		0.74x0.74	42.83%	-0.04	-0.04	-0.01	-2.35	-1.01	-0.03	342	356	1445
		0.85x0.85	32.25%	-0.03	-0.01	-0.01	-1.49	-0.06	-0.03	336	351	1351
swerv_wrapper CLK=1.0ns	12nm	0.52x0.52	46.85%	-0.11	-0.08	0.01	-18.68	-5.28	0.00	2320	2346	9291
		0.58x0.58	38.14%	0.18	0.10	0.29	0.00	0.00	0.00	2126	2152	8498
		0.67x0.67	30.10%	-0.28	**-0.15**	-0.18	-53.91	-49.30	-47.37	2897	2923	9775
coyote CLK=3.2ns		0.56x0.56	47.87%	-0.09	-0.01	0.01	-0.80	-0.01	0.01	3455	3514	13804
		0.59x0.59	42.64%	-0.15	-0.05	-0.02	-2.87	-0.12	-0.02	3986	4045	15872
		0.68x0.68	38.33%	-0.14	-0.09	0.00	-1.07	-0.48	0.00	3698	3757	14715



```
Startpoint: _81642_ (rising edge-triggered flip-flop clocked by CLK)
Endpoint: fe_cmd_o[51] (output port clocked by CLK)
Path Group: CLK
Path Type: max

    Cap     Slew    Delay    Time    Description
---------------------------------------------------------------------------
                    0.0000   0.0000  clock CLK (rise edge)
                    0.3078   0.3078  clock network delay (propagated)
           0.0238   0.0000   0.3078 ^ _81642_/CK (DFF_X2)
  14.1420  0.0195   0.1322   0.4399 ^ _81642_/Q (DFF_X2)
           0.0195   0.0006   0.4406 ^ repeater7842/A (BUF_X16)
  59.0805  0.0101   0.0273   0.4678 ^ repeater7842/Z (BUF_X16)
     :       :        :        :              :
           0.0058   0.0003   0.7403 ^ repeater5814/A (BUF_X4)
  22.7279  0.0151   0.0286   0.7689 ^ repeater5814/Z (BUF_X4)
           0.0152   0.0019   0.7708 ^ repeater5813/A (BUF_X4)
  23.5321  0.0155   0.0325   0.8033 ^ repeater5813/Z (BUF_X4)
           0.0156   0.0020   0.8053 ^ repeater5812/A (BUF_X8)
  24.1918  0.0098   0.0266   0.8318 ^ repeater5812/Z (BUF_X8)
     :       :        :        :              :
           0.0142   0.0003   1.1063 v _44515_/B (MUX2_X1)
   1.5798  0.0099   0.0600   1.1663 v _44515_/Z (MUX2_X1)
           0.0099   0.0000   1.1663 v _44516_/C2 (AOI221_X1)
   1.8059  0.0421   0.0547   1.2210 ^ _44516_/ZN (AOI221_X1)
           0.0421   0.0000   1.2210 ^ _44535_/A1 (NAND3_X1)
   3.1779  0.0195   0.0320   1.2530 v _44535_/ZN (NAND3_X1)
     :       :        :        :              :
           0.0161   0.0013   1.4471 ^ _45105_/A2 (NAND3_X4)
  12.8435  0.0153   0.0268   1.4739 v _45105_/ZN (NAND3_X4)
           0.0153   0.0005   1.4744 v repeater3042/A (BUF_X4)
  28.7790  0.0101   0.0354   1.5098 v repeater3042/Z (BUF_X4)
           0.0106   0.0028   1.5127 v repeater3038/A (BUF_X4)
  31.7838  0.0105   0.0329   1.5456 v repeater3038/Z (BUF_X4)
     :       :        :        :              :
           0.0069   0.0005   2.2568 v _46427_/B1 (OAI22_X4)
  26.8284  0.0531   0.0607   2.3176 ^ _46427_/ZN (OAI22_X4)
           0.0532   0.0065   2.3241 ^ output1779/A (BUF_X1)
   1.0893  0.0074   0.0291   2.3532 ^ output1779/Z (BUF_X1)
           0.0074   0.0000   2.3533 ^ fe_cmd_o[51] (out)
---------------------------------------------------------------
                    2.2000   data required time
                   -2.3533   data arrival time
---------------------------------------------------------------
                   -0.1533   slack (VIOLATED)
```

```
Startpoint: _81642_ (rising edge-triggered flip-flop clocked by CLK)
Endpoint: fe_cmd_o[51] (output port clocked by CLK)
Path Group: CLK
Path Type: max

    Cap     Slew    Delay    Time    Description
---------------------------------------------------------------
                    0.0000   0.0000  clock CLK (rise edge)
                    0.2804   0.2804  clock network delay (propagated)
           0.0237   0.0000   0.2804 ^ _81642_/CK (DFF_X2)
  14.1118  0.0195   0.1321   0.4126 ^ _81642_/Q (DFF_X2)
           0.0195   0.0006   0.4132 ^ repeater7842/A (BUF_X16)
  59.4902  0.0102   0.0274   0.4406 ^ repeater7842/Z (BUF_X16)
     :       :        :        :              :
           0.0057   0.0003   0.7070 ^ repeater5814/A (BUF_X4)
  26.1237  0.0168   0.0302   0.7372 ^ repeater5814/Z (BUF_X4)
           0.0170   0.0027   0.7398 ^ repeater5813/A (BUF_X8)
  23.5811  0.0097   0.0268   0.7666 ^ repeater5813/Z (BUF_X8)
           0.0097   0.0019   0.7685 ^ repeater5812/A (BUF_X8)
  23.9374  0.0097   0.0245   0.7930 ^ repeater5812/Z (BUF_X8)
     :       :        :        :              :
           0.0142   0.0003   1.0677 v _44515_/B (MUX2_X1)
   3.0629  0.0114   0.0639   1.1316 v _44515_/Z (MUX2_X1)
           0.0114   0.0001   1.1317 v _44516_/C2 (AOI221_X2)
   1.8426  0.0360   0.0482   1.1799 ^ _44516_/ZN (AOI221_X2)
           0.0360   0.0000   1.1800 ^ _44535_/A1 (NAND3_X1)
   3.0302  0.0179   0.0298   1.2097 v _44535_/ZN (NAND3_X1)
     :       :        :        :              :
           0.0162   0.0013   1.4030 ^ _45105_/A2 (NAND3_X4)
  15.3944  0.0167   0.0287   1.4318 v _45105_/ZN (NAND3_X4)
           0.0167   0.0006   1.4324 v repeater3042/A (BUF_X8)
  28.3560  0.0071   0.0314   1.4637 v repeater3042/Z (BUF_X8)
           0.0077   0.0032   1.4669 v repeater3038/A (BUF_X4)
  32.1545  0.0106   0.0316   1.4985 v repeater3038/Z (BUF_X4)
     :       :        :        :              :
           0.0069   0.0005   2.2049 v _46427_/B1 (OAI22_X4)
  26.8384  0.0533   0.0614   2.2662 ^ _46427_/ZN (OAI22_X4)
           0.0534   0.0057   2.2719 ^ output1779/A (BUF_X2)
   1.0229  0.0066   0.0262   2.2982 ^ output1779/Z (BUF_X2)
           0.0066   0.0000   2.2982 ^ fe_cmd_o[51] (out)
---------------------------------------------------------------
                    2.2000   data required time
                   -2.2982   data arrival time
---------------------------------------------------------------
                   -0.0982   slack (VIOLATED)
```

Fig. 14. A critical path from bp_be 45nm after DR, from the traditional flow (left) and from the ML-based flow (right).

Table 5 compares the post-DR WS and TNS from the ML-based flow and from the traditional flow, in OpenROAD. Out of 15 designs, the ML-based flow improves post-DR WS for 13 designs and improves post-DR TNS for 14 designs as highlighted in green, indicating effective timing optimization post-GR using the ML model. Of the remaining cases, "Trad." and "ML" achieve the same results. For the cases in bold green, the post-DR results from the ML-based flow are even better than the results from the ground-truth-based flow. Based on a detailed analysis of the results, we attribute this to two reasons. First, the path slack from the ML-based estimation can be pessimistic due to prediction error; this

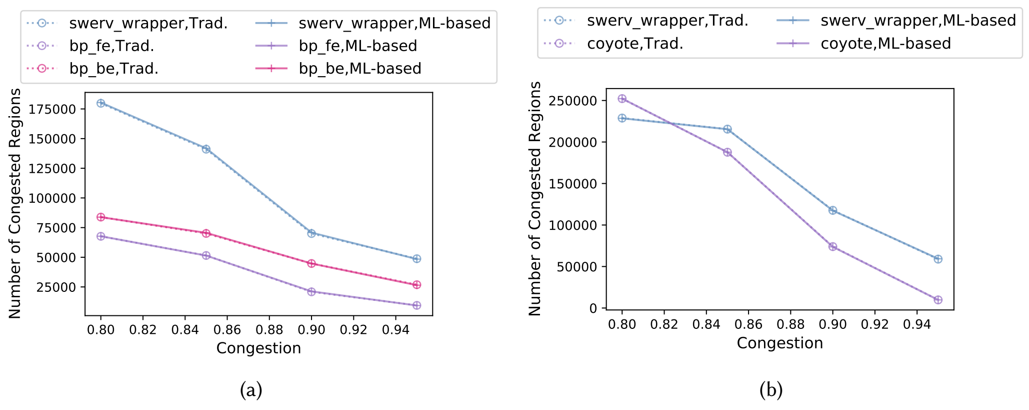


Fig. 15. Number of post-DR congested regions in the traditional and ML-based flows in (a) 45nm designs and (b) 12nm designs.

makes the tool flow optimize delays more aggressively during post-GR optimization and thus helps improve post-DR WNS. Second, the routing of the worst post-DR timing path of the ground-truth-based flow has larger detours than the routing of the same path in the ML-based flow, which leads to more parasitics and worse timing, even though the two flows achieve the same buffering and sizing during post-GR optimization.

Table 5 also compares the ML-based flow to the flow proposed in the preliminary version of this paper in [7]. We find that the post-DR WS of 10 designs and the post-DR TNS of 13 designs are improved by our introduction of new features.

To determine the impact of ML-based post-GR timing optimizations on routability, we analyze congestion under traditional and ML-based flows. We define a GCell to be congested if its congestion exceeds a specified threshold. Fig. 15 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 (i.e., the traces are superposed), indicating that it does not impact routability.

The last three columns of Table 5 compare the runtimes of three different flows. We see that the ML-based flow can leverage prediction of post-DR timing at the cost of a few 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 31.87% over the ground-truth flow. When compared to the traditional flow, the ML-based flow is only marginally (0.84%) slower than the traditional flow, which is acceptable as the model can achieve a quality that is closer to, and sometimes better than, the ground-truth flow.

Table 6 compares the post-DR WS and TNS results from the commercial tool flow. Out of 15 designs, the ML-based flow improves post-DR WS for 10 designs and improves post-DR TNS 13 designs. Four designs have the same post-DR WS in the traditional flow and the ML-based flow. One design (swerv_wrapper in 12nm) improves the WS of 0.18ns from the traditional flow to 0.10ns using our approach, with lower buffering and sizing cost. For post-DR TNS, the ML-based flow shows better results in 13 designs, and the remaining designs do not have path violations.

5.3 Generalization to different clock periods

In the experiments above, the designs used for training have been implemented at a single target clock period. We now analyze the ability of our ML models to generalize with respect to unseen designs generated by using different clock

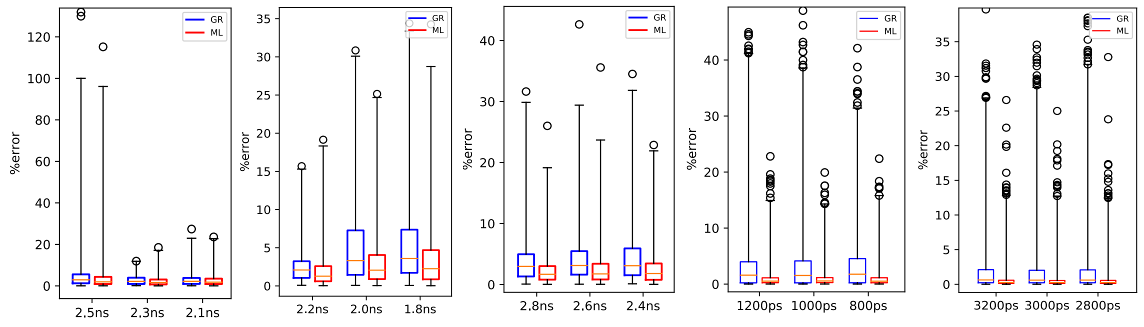


Fig. 16. Sensitivity of wire delay models to different clock periods for (from left to right) swerv_wrapper, bp_fe, bp_be in 45nm, and swerv_wrapper, coyote in 12nm.

Table 7. Mean %error of prediction for designs generated using different clock periods.

Design	Tech	Clock period	ML wire delay	GR wire delay	ML wire slew	GR wire slew
swerv_wrapper	45nm	2.50ns	0.84%	4.27%	0.04%	0.20%
		2.30ns	2.40%	4.50%	0.06%	0.06%
		2.10ns	3.35%	5.61%	0.07%	0.09%
bp_fe		2.20ns	0.64%	3.63%	0.04%	0.24%
		2.00ns	2.91%	3.15%	0.23%	0.31%
		1.80ns	2.40%	3.78%	0.16%	0.25%
bp_be		2.80ns	2.16%	3.00%	0.07%	0.12%
		2.60ns	2.33%	3.13%	0.12%	0.13%
		2.40ns	2.51%	3.70%	0.11%	0.13%
swerv_wrapper	12nm	1.20ns	0.87%	4.26%	0.04%	0.70%
		1.00ns	1.21%	4.03%	0.12%	0.56%
		0.80ns	1.22%	4.79%	0.26%	1.31%
coyote		3.20ns	0.60%	3.47%	0.22%	1.97%
		3.00ns	0.75%	3.23%	0.32%	1.99%
		2.80ns	0.81%	3.17%	0.31%	1.68%

periods from the one used for training. To test the sensitivity of the models to different clock periods, we implement each design with two more clock periods and extract the input features for running ML inferences on two unseen designs. Fig. 16 shows the prediction accuracy for all designs generated using three different clock constraints. Here, the model presented in Tables 4 and 5 is used for the leftmost clock period. The leftmost clock period of each plot is used for training, which is also denoted in blue in Table 7, and the other two clock periods are unseen during the training. The mean %error for each case is listed in Table 7. In the box-whisker plot, the boxes indicate the 25th percentile, 50th percentile (the median, shown by a yellow line), and 75th percentile of the prediction error of the traditional method in GR and ML-based method. For the ML-based method, the box and the median are generally lower than for the traditional method, indicating that ML models have good ability to generalize with respect to unseen designs with different clock constraints. The table summarizes the mean %error of the prediction for different clock periods, one of which used for training and is highlighted in blue. The smaller ML prediction errors for designs in 12nm also indicate that our models generalize well in 12nm.

Table 8. Impact of training-stage noise, for various values of the noise standard deviation, on prediction accuracy.

Design	Tech	Metrics	Wire Delay			Wire Slew		
			Trad.	ML	ML-Noise(0.01, 0.05, 0.1)	Trad.	ML	ML-Noise(0.01, 0.05, 0.1)
swerv_wrapper clk=2.5ns	45nm	Mean %error	4.27%	0.84%	(1.37%, 1.89%, 2.65%)	0.20%	0.04%	(0.09%, 0.20%, 0.33%)
		Max %error	149.41%	115.19%	(124.90%, 133.41%, 137.82%)	40.63%	19.81%	(33.79%, 33.76%, 34.38%)
		Std. dev. %error	7.28%	2.78%	(4.34%, 5.24%, 5.73%)	1.22%	0.34%	(0.65%, 0.76%, 0.75%)
bp_fe clk=2.2ns		Mean %error	3.63%	0.64%	(1.60%,4.39%, 7.06%)	0.24%	0.04%	(0.10%, 0.31%, 0.51%)
		Max %error	15.68%	19.14%	(25.32%, 27.98%, 47.65%)	5.83%	4.42%	(3.70%, 5.36%, 7.39%)
		Std. dev. %error	4.65%	0.91%	(1.47%, 4.17%, 7.31%)	0.98%	0.13%	(0.22%, 0.52%, 0.92%)
bp_be clk=2.8ns		Mean %error	3.00%	2.16%	(2.64%, 2.71%, 4.36%)	0.12%	0.07%	(0.08%, 0.19%, 0.33%)
		Max %error	55.64%	34.58%	(31.64%, 39.62%, 47.77%)	10.45%	9.15%	(14.62%, 18.69%, 28.10%)
		Std. dev. %error	3.88%	3.42%	(3.78%, 3.08%, 3.23%)	0.38%	0.25%	(0.44%, 0.62%, 0.91%)
swerv_wrapper clk=1200ps	12nm	Mean %error	4.26%	0.87%	(1.03%, 2.73%, 4.89%)	0.70%	0.04%	(0.04%, 0.09%, 0.16%)
		Max %error	44.95%	22.80%	(25.90%, 32.35%, 67.92%)	71.75%	29.81%	(29.28%, 29.65%, 33.14%)
		Std. dev. %error	5.76%	0.93%	(0.99%, 2.69%, 5.25%)	3.85%	0.17%	(0.18%, 0.22%, 0.32%)
coyote clk=3200ps		Mean %error	3.47%	0.60%	(0.68%, 2.34%, 4.16%)	1.97%	0.22%	(0.23%, 0.80%, 1.43%)
		Max %error	39.65%	26.58%	(27.65%, 30.56%, 63.40%)	51.34%	40.13%	(41.80%, 43.21%, 43.42%)
		Std. dev. %error	4.73%	0.88%	(0.91%, 3.10%, 5.87%)	4.51%	0.62%	(0.64%, 1.10%, 1.78%)

Table 9. Impact of noise with various standard deviation on post-DR results.

Design	Tech	Post-DR WS (ns)			Post-DR TNS (ns)		
		Trad.	ML-Noise(0, 0.01, 0.05, 0.1)	Ground-truth	Trad.	ML-Noise(0, 0.01, 0.05, 0.1)	Ground-truth
swerv_wrapper clk=2.5ns	45nm	-0.14	(-0.14, -0.14, -0.14, -0.15)	-0.13	-23.45	(-21.96, -22.58, -23.52, -26.89)	-22.58
bp_fe clk=2.2ns		-0.15	(-0.10, -0.14, -0.16, -0.21)	-0.10	-1.60	(-0.95, -1.40, -1.88, -2.30)	-0.72
bp_be clk=2.8ns		-0.18	(-0.17, -0.17, -0.18, -0.18)	-0.15	-9.63	(-7.82, -7.65, -8.14, -8.77)	-6.60
swerv_wrapper clk=1200ps	12nm	-0.48	(-0.23, -0.28, -0.46, -0.58)	-0.35	-207.68	(-202.38, -207.43,-209.80, -230.48)	-200.55
coyote clk=3200ps		-0.02	(0.49, 0.47, 0.40, 0.34)	0.86	-0.14	(0, 0, 0, 0)	0.00

5.4 Noise impact on model performance

As there is inherent noise from EDA tools, the data with noise collected from EDA tools and used in our ML-based flow can affect the DR solution quality. To study the impact of noise on the model accuracy and DR outcomes from the ML-based flow, we add Gaussian noise $\mathcal{N}(0, \sigma)$ to labels and train the models using the datasets with noise. A set of σ values (0, 0.01, 0.05, 0.1) are selected to evaluate the robustness of the models to the noise with different standard deviations. We generate noise matrices that have the same dimension with our training dataset using three different random seeds for each σ value, then add each noise matrix to the training dataset separately. We take the average value of the three predictions from the models trained by the three different training datasets as our prediction result. Then, we compare the results from a modified ML-based flow (where the ML models are replaced by the models trained with noisy data) to the results from the original ML-based flow.

In Table 8, the prediction errors of models trained with noisy data are compared to the traditional estimation and the prediction from models trained with a dataset without added noise. According to the results for $\sigma = 0.01$, ML models can handle the noise with $\sigma = 0.01$ value in most cases; the exceptions are maximum %error for wire delay of bp_fe in 45nm, and maximum and standard deviation of %error for wire slew of bp_be in 45nm. These are denoted in red, indicating no improvement compared to the estimation from traditional methods. According to the results for $\sigma = 0.05$, the ML models perform better on four out of five designs for both wire delay and wire delay. However, when $\sigma = 0.1$, which is approximately equal to the maximum wire delay and 2× maximum slew in our datasets, the models trained by noisy data cannot make accurate predictions for most designs.

The impact of noise on post-DR solution quality is summarized in Table 9. We compare the post-DR results from the ML-based flow using the models trained with noisy data, to the traditional flow and ground-truth-based flows. The ML-based flow generates better DR solutions than the traditional flow for all testcases when the noise has $\sigma = 0.01$. As we increase the σ value, the DR results of some testcases denoted in red are no longer better than the results from the traditional flow, due to inaccurate timing prediction being used in post-GR timing optimization. For $\sigma = 0.05$ and $\sigma = 0.1$, only two designs out of five designs still see better solutions from the ML-based flow.

6 CONCLUSION

We study an ML-based method to predict post-DR timing at the post-route stage. Our models demonstrate better accuracy of timing and parasitic estimation at the post-GR stage compared to traditional methods, using both OpenROAD and a commercial tool flow. Our approach shows improvements in the DR solution quality; it is shown to be computationally efficient, and does not increase congestion compared to the prior methods. Moreover, our experimental results show that our models generalize well to designs generated using unseen clock periods, and that our flow generates better DR solutions even when models are trained using datasets with noise.

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-2112665.

REFERENCES

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, Jeongsup Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu. 2019. Toward an Open-source Digital Flow: First Learnings from the OpenROAD Project. In *Proc. DAC*. Association for Computing Machinery, New York, NY, 1–4.

[2] E. C. Barboza, N. Shukla, Y. Chen, and J. Hu. 2019. Machine Learning-Based Pre-Routing Timing Prediction with Reduced Pessimism. In *Proc. DAC*. Association for Computing Machinery, New York, NY, USA, 1–6.

[3] T.-B. Chan, A. B. Kahng, and M. Woo. 2020. Revisiting Inherent Noise Floors for Interconnect Prediction. In *Proc. SLIP*. Association for Computing Machinery, New York, NY, USA, 7 pages.

[4] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. D. MacDonald, and S. Nath. 2016. Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design. In *Proc. ASP-DAC*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 178–185.

[5] T. Chen and C. Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. SIGKDD*. Association for Computing Machinery, New York, NY, USA, 785–794.

[6] 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*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 1–6.

[7] V. A. Chhabria, W. Jiang, A. B. Kahng, and S. S. Sapatnekar. 2022. From Global Route to Detailed Route: ML for Fast and Accurate Wire Parasitics and Timing Prediction. In *Proc. MLCAD*. Association for Computing Machinery, New York, NY, USA, 7–14.

[8] 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.

[9] S. Dolgov, A. Volkov, L. Wang, and B. Xu. 2019. 2019 CAD Contest: LEF/DEF Based Global Routing. In *Proc. ICCAD*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 1–8.

[10] 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*. Association for Computing Machinery, New York, NY, USA, 6 pages.

[11] X. He, Z. Fu, Y. Wang, C. Liu, and Y. Guo. 2022. Accurate Timing Prediction at Placement Stage with Look-Ahead RC Network. In *Proc. DAC*. Association for Computing Machinery, New York, NY, USA, 6 pages.

[12] 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.

[13] NanGate45 2022. *NanGate 45nm FreePDK and Cell Library.* https://si2.org/open-cell-library

[14] P. R. O'Brien and T. L. Savarino. 1989. Modeling the Driving-point Characteristic of Resistive Interconnect for Accurate Delay Estimation. In *Proc. ICCAD*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 512–515.

[15] OpenROAD 2022. *OpenROAD*. https://github.com/The-OpenROAD-Project/OpenROAD

[16] OpenROAD-flow-scripts 2022. *OpenROAD-flow-scripts*. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts

[17] OpenSTA 2022. *OpenSTA*. https://github.com/The-OpenROAD-Project/OpenSTA

[18] 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 pages.

[19] S. S. Sapatnekar. 2004. *Timing*. Springer, Boston, MA.

[20] SkyWater130 2022. *SkyWater 130nm PDK*. https://github.com/google/skywater-pdk

[21] T. Yang, G. He, and P. Cao. 2022. Pre-Routing Path Delay Estimation Based on Transformer and Residual Framework. In *Proc. ASP-DAC*. Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, 184–189.

[22] Y. Ye, T. Chen, Y. Gao, H. Yan, B. Yu, and L. Shi. 2023. Fast and Accurate Wire Timing Estimation Based on Graph Learning. In *Proc. DATE*. Association for Computing Machinery, New York, NY, USA, 6 pages.