Placement Tomography-Based Routing Blockage Generation for DRV Hotspot Mitigation

Andrew B. Kahng
UC San Diego, USA
abk@ucsd.edu

Sayak Kundu
UC San Diego, USA
sakundu@ucsd.edu

Dooseok Yoon
UC San Diego, USA
d3yoon@ucsd.edu

ABSTRACT

A fundamental goal in modern physical design is for the post-route layout to have a *fixable* number of remaining design rule violations (DRVs). We study how to apply *routing blockages* to a fixed placement solution, so as to "condition" the routing problem and minimize DRVs in the post-route outcome. Motivated by the widening turnaround time gap between early global routing (eGR) and detailed routing, we propose *placement tomography* (that uses multiple views of a placement from near-free eGR runs) as a new basis for generating layer-wise route blockages and mitigating post-route DRVs. Our framework includes (i) DRVNet, a machine learning model that predicts layer-wise DRV hotspots; (ii) BlkgComp, a learning-based model for assessing the relative effectiveness of two different routing blockages in mitigating DRVs in hotspots; and (iii) a reinforcement learning approach with BlkgComp to generate routing blockages for the hotspots predicted by DRVNet. Experimental studies confirm that our BlkgComp model achieves up to 73% accuracy and 0.53 Kendall rank on the testing dataset for open-source and commercial enablements. Our framework produces routing blockage solutions that reduce post-route DRVs by up to 88% compared to baseline commercial tool flows and up to 21% compared to a human expert baseline that was able to access detailed route outcomes.

ACM Reference Format:
Andrew B. Kahng, Sayak Kundu, and Dooseok Yoon. 2024. Placement Tomography-Based Routing Blockage Generation for DRV Hotspot Mitigation . In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676828

1 INTRODUCTION

Achieving zero design rule violations (DRVs) is a strict requirement for tapeout. As design rules become more complex with each technology node, back-end EDA tools have increasing difficulty satisfying these rules, and human engineers often spend considerable time manually resolving DRVs. The number of DRVs can explode as product teams push density to achieve minimum die area, even as they seek to also reduce power and maximizing performance (PPA) within tight schedules. The time required to fix these DRVs is an increasing function of the number of DRVs and their distinct types. Therefore, in the physical design (PD) process, engineering teams keep or discard post-route solutions based on whether they are *fixable* or *unfixable.*[1] Discarding a solution as 'unfixable' adds to project schedule and cost, which raises the question: *Do we give up on placement solutions that actually permit fixable routing outcomes?* This paper proposes a new methodology to "condition" the input to the routing tool so as to shift the border between unfixable and fixable outcomes from a given placement.

The routing blockage lever. Modern P&R tools offer a "routing blockage" mechanism to reduce the track supply that is assumed by the router, as a percentage of the default supply, within a specified layout region.[2] Routing blockage of '100' means the router can assume that the entire default track supply is available; '70' means that 30% of the supply is removed from consideration.

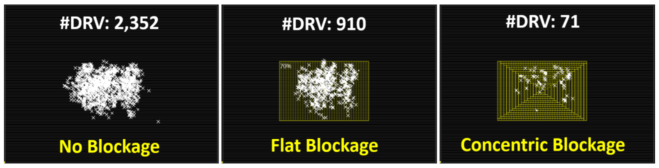


Fig. 1: Effect of routing blockages on DRV count: (l-r) no blockage (default); flat blockage; and concentric blockage.

The routing blockage lever is both powerful and complex. Routing blockages in a DRV hotspot will reduce the number of routed nets in the region, making it easier for the detailed router to route the nets with fewer DRVs. However, adding routing blockages will change the detouring of nets and shift DRV hotspots. Fig. 1 shows post-route DRV markers in the Cadence Innovus 21.1 tool for default (no blockage), best-found *flat* blockage, and best-found *concentric* blockage. The example is from the class of "PROBE" layouts in Nangate45 enablement described in Subsection 5.1 below. The figure shows that (i) routing blockages can potentially shift the outcome of a placement solution from unfixable to fixable, and (ii) their optimal application is non-obvious.

Recent studies [17, 24] have introduced methods for predicting hotspots and strategies for managing them, either by altering the placement solution or by incorporating routing blockages across all layers. Nevertheless, there has been no research focused on generating routing blockages with specific configurations tailored for identified hotspots. In this paper, we address this gap, and our primary contributions are outlined as follows.

- We introduce a new type of routing blockage, the *concentric routing blockage*. We study *layer-wise* flat and concentric routing blockages to maximize the benefits in terms of DRV mitigation.
- We use a novel *tomography* [13] approach to feature extraction. We extract the rich set of tomography features from multiple early global routing (eGR) runs on a placed design. These tomography features improve our ability to predict layer-wise DRV hotspots.
- We introduce a new machine learning-based (ML) routing blockage comparator (*BlkgComp*) to compare routing blockages for given hotspots. Our model takes two routing blockages and predicts their relative quality. We use DRVNet, a U-Net-based model similar to [17, 30], for layer-wise DRV prediction. Our experimental results show that our BlkgComp model achieves up to 73% accuracy in predicting improved routing blockages, while DRVNet

[1]The threshold is determined by schedule and PD resource. Note that we use *DRV* to indicate a layout design rule violation. The term is sometimes used to indicate electrical rule violations (maxtran, maxcap, etc.), but that is outside our present scope.

[2]The command names (*createRouteBlk*, *create_route_guide*, etc.) and options/semantics differ slightly across tools. In general, (i) control can be by direction (h or v) and/or by layer, and (ii) the region can be polygonal or rectangular.

ICCAD '24, October 27–31, 2024, New York, NY, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676828

predicts layer-wise hotspots with up to 99% accuracy and an F1 score of 0.79.

- We introduce a reinforcement learning (RL)-based approach that uses tomography features along with DRVNet and BlkgComp models to generate routing blockages for a given design. This method can produce similar or superior routing blockages compared to expert human baselines, achieving up to 21% reduction in final DRV count even when the human is allowed to consult detailed route outcomes.
- We validate our routing blockage generation using Cadence Innovus 21.1, in Nangate45 (NG45) [37] and GlobalFoundries 12nm (GF12) enablements. We develop competitive baseline solutions and perform ablation studies to show the effectiveness of our approach.

The rest of the paper is organized as follows. Section 2 reviews motivating studies and related works; Section 3 introduces notations and our problem statements; Section 4 describes our overall framework; Section 5 presents experimental setup; Section 6 discusses experiment results; and we conclude in Section 7.

2 BACKGROUND AND MOTIVATING STUDIES

Ensuring routability is one of the most important goals in physical design, particularly during floorplanning and placement. Here, We review related works and three motivating studies.

2.1 Related Works

Related works on routing blockage prediction and generation include (i) analytical methods, (ii) machine learning methods, and (iii) routing blockage generation techniques.

Analytical methods often include the use of superposed MST or heuristic RSMT constructions for congestion estimation. [14, 22] apply probabilistic methods to improve routing congestion estimates. [12] introduces a modified Rent parameter to identify tangled logic structures, along with mitigations by cell inflation and logic restructuring. [27] presents a fast routing demand estimator along with its use in congestion-aware placement. Mitigations before detailed routing are exemplified by [20], which uses 3D pattern routing to reduce the probability of layer-wise congestion in global routing.

Machine Learning (ML) methods dominate the recent literature, given the rapid advance of hardware and methods, as well as increased design rule complexity which challenges analytical methods. [4] uses an SVM-based prediction framework to redistribute white space in detailed placement to reduce post-route DRVs. Works such as [6, 7, 28, 30] use convolutional neural networks (CNNs) to predict congested routing regions, while [11, 18, 19] use CNNs to predict Gcell-based DRV hotspots.[3] On the other hand, [3, 5, 10] use graph neural networks (GNNs) for congestion and short violation prediction. Similar to [12], the work in [23] uses GNNs to predict tangled logic cells in a netlist and employs cell padding during placement for congestion mitigation. [29, 31] exploit both Gcell properties and netlist connectivity for congestion prediction, while [21] uses correlation with neighboring Gcells to improve accuracy of congestion prediction. In general, while such previous methods can accurately predict DRV hotspots, the proposed mitigations modify the placement and/or cost functions in global routing. By contrast, we do not touch the placement, and we treat the router as a "black box".

Use of routing blockages. Two recent works study the routing blockage mechanism. [24] introduces an explainable AI method to discern root causes of DRV hotspots (e.g., high cell or pin density, or high routing congestion). The authors implement routing blockages and placement density screens to reduce DRVs. [17] uses CNNs to predict congestion, and demonstrates that applying routing blockages at the coarse granularity of 25, 50, or 75 percent can reduce DRVs in predicted hotspots. However, these works do not propose any means to optimize routing blockages, such as our layer-wise routing blockage definition with different partial densities and types. Nor do they explore methods to extract more relevant features that improve model performance, such as our placement tomography.

[3]A *Gcell* is a unit gridcell in global routing, with side length equal to 15 M3 track pitches [35].

2.2 Motivating Studies

Three motivating studies demonstrate the effectiveness of layer-wise routing blockage, benefits of placement tomography features, and challenges associated with routing blockage generation.

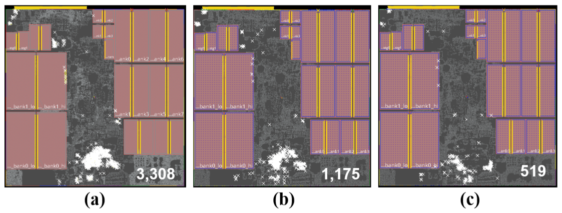


Fig. 2: Effect of layer-wise routing blockage vs. all-layer routing blockage. (a) No blockage; (b) all-layer blockage; and (c) layer-wise blockage. Testcase: CA53 in NG45 enablement.

Layer-wise routing blockage. Fig. 2(a) shows post-routing DRV markers for Arm Cortex-A53 (CA53) in NG45, where the majority of DRVs occur on the *M*2, *M*3, *M*5, and *M*6 layers. The figure shows two sets of routing blockages: (b) across all layers, and (c) limited to these four specific layers. These respectively decrease total DRV count from 3,308 in (a) to 1,175 in (b), and to 519 in (c), demonstrating substantial potential benefit from layer-wise routing blockages.

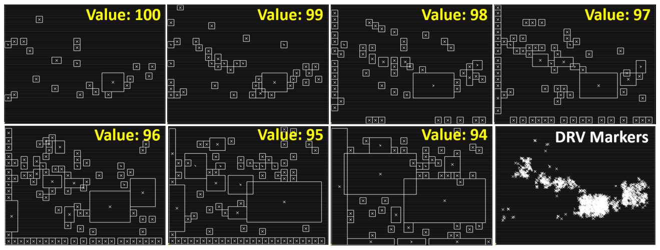


Fig. 3: Congestion report for routing blockage with varying partial density. Note: All screenshots are from the Innovus 21.1 GUI.

Placement tomography. The concept of placement "tomography" was first noted in [13], but its detailed application was not explored. Here, *tomography* – typically used for cross-sectional views of solid objects – offers a useful analogy. We use *placement tomography* to denote extraction of many views of a given placement, using multiple runs of eGR or other post-placement analyses. Fig. 3 shows seven congestion maps from Innovus generated using *earlyGlobalRoute* followed by *reportCongestion*, each produced in less than one second of runtime. By contrast, obtaining the final DRV markers at lower-right requires 1.5 hours of detailed router runtime. The congestion map for routing blockage with partial density 97 aligns best with post-route DRVs, but having multiple eGR reports affords a rich set of placement features (cell density, pin density, overflow, etc.) toward ML-enabled "conditioning" of the routing problem.

Interaction between hotspots. Fig. 4 shows how two closely placed hotspots affect each other, and how the best routing blockages for individual hotspots become less useful when jointly applied. A key challenge is to find the best blockages for multiple hotspots with consideration of such proximity effects.

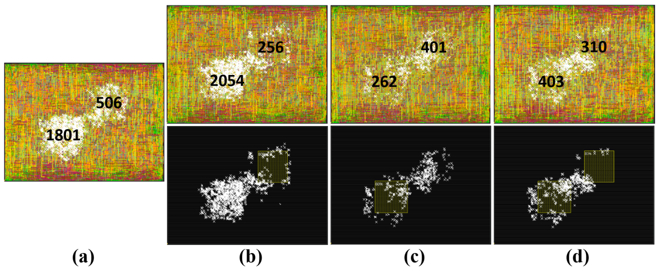


Fig. 4: Interaction between hotspots. (a) No blockage; (b) blockage on upper right hotspot; (c) blockage on lower left hotspot; and (d) blockage on both hotspots.

3 PRELIMINARIES

3.1 Notations

We use the following terms in our work.

- A *Gcell* is a rectangular tile that is a unit gridcell for global routing.
- A *congested Gcell* has at least one design rule violation (DRV) after detailed routing.
- A *hotspot* is a cluster of congested Gcells. In our implementation, a hotspot has at least 16 congested Gcells, each of which has at least six other congested Gcells within a three-Gcell radius in the same cluster (Section 4.2).
- A *routing blockage* (Blkg) reduces the available routing resources for one or more layers in its region. We apply Blkgs to hotspots.
- The *partial density* of a Blkg is the percentage of total routing resource that is available in the Gcells covered by the Blkg.
- A *flat* blockage (fBlkg) is a routing blockage with a constant partial density d for layer l_i, denoted as fBlkg(d, l_i).
- A *concentric* blockage (cBlkg) consists of multiple fBlkgs, each having a rectangular cutout, except for the central one, which is solid. All of these blockages share a common center, and their dimensions and partial densities vary linearly. We denote the configuration of a cBlkg on routing layer l_i as cBlkg(d_o, d_i, c, l_i), where the outermost blockage is fBlkg(d_o, l_i), the innermost blockage is fBlkg(d_i, l_i), and the total number of blockages is c.
- A *set of blockages* (BlkgSet) contains one or more Blkgs (flat or concentric), with one for each hotspot region of a given design.

3.2 Problem Formulation

Our approach to layer-wise routing blockage generation entails solving three distinct problems. For a fixed placement and a given set of Gcell-based features $\{x_1, x_2, \ldots\}$, these are as follows.

- **Problem 1:** Train an ML model (DRVNet) to predict layer-wise congested Gcells in hotspots.

$$(H, L) = DRVNet(x_1, x_2, \ldots); \quad H, x_i \in \mathbb{R}^{h\times w}, L \in \mathbb{R}^{h\times w\times l} \tag{1}$$

  Here, h and w represent the number of Gcells along the height and width of the design, respectively, while l denotes the number of layers. H is a matrix where each entry is a binary variable corresponding to a Gcell in the design. For Gcells located within a hotspot, the corresponding entry in H is set to 1. L is a matrix where each entry consists of a list of binary variables indicating the layer-wise congestion of the corresponding Gcell.
- **Problem 2:** Train an ML model (BlkgComp) to predict the relative effectiveness of two BlkgSets, e_1 and e_2.

$$(p_1, p_2) = BlkgComp(x_1, x_2, \ldots, e_1, e_2, L) \tag{2}$$

  Here, $p_1 > p_2$ indicates that the BlkgSet e_1 results in fewer DRVs than e_2, and $p_1 < p_2$ indicates the opposite.
- **Problem 3:** Train a reinforcement learning agent (RL-agent) to sample BlkgSets for the layer-wise hotspots predicted by the DRVNet model, where the RL-agent evaluates the quality of the BlkgSet using the BlkgComp model.

$$(E) = RL\text{-}Model(x_1, x_2, \ldots, L) \tag{3}$$

  E is the sampled BlkgSets expected to mitigate DRVs.

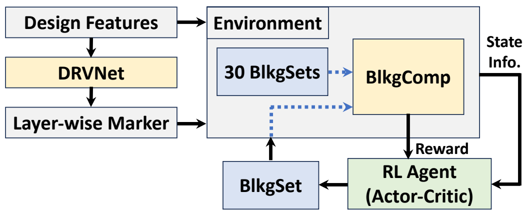


Fig. 5: Overall flow for BlkgSets generation.

4 OUR APPROACH

Fig. 5 presents the overall flow for routing blockage generation. We first extract features of a given design. Next, we use these features and the DRVNet model to predict layer-wise routing hotspots. We then use the design features, predicted layer-wise hotspot information, the BlkgComp model, and the RL agent to sample better BlkgSets for the design. We now describe these features and our models.

4.1 Features

We divide the feature list into *design* and *tomography* features. We extract the following *design* features of each Gcell.

- **Cell and macro density:** ratio of total standard cell and macro area overlapping with the Gcell to the Gcell area.
- **Pin count:** number of standard cell and macro signal pins overlapping with the Gcell.
- **Horizontal and vertical resource:** total number of usable horizontal and vertical routing tracks in the Gcell.

We extract the following per-layer *design* features of each Gcell when there are no Blkgs covering the whole design.

- **Layer-wise resource:** total number of usable routing tracks for the routing layer in the Gcell.
- **Layer-wise via:** total number of used vias in contact with the routing layer in the Gcell, e.g., for the M3 layer, the layer-wise via count is the total number of V2 and V3 vias in the Gcell.

We extract the following *tomography* features of each Gcell:

- **Horizontal (vertical) routing resource usage:** fraction of horizontal (vertical) routing tracks used in the Gcell by eGR at varying partial densities of Blkgs.
- **Layer-wise resource usage:** fraction of routing tracks for each layer used in the Gcell by eGR at varying partial densities of Blkgs.

To capture this information, we add a flat routing blockage and then run eGR to capture the horizontal and vertical routing resource usage features. We do this seven times, varying the partial density of the routing blockage from 70% to 100% in increments of 5%. For each layer and routing blockage, we pass the layer-wise resource, usage, and via as different channels to the model. In modern technology nodes, each routing layer has a preferred routing direction. Separately providing the horizontal and vertical resource and usage features helps the model distinguish different types of hotspots.

4.2 DRVNet Model

For the DRVNet model, we use the U-Net [25] architecture, which is often used for image segmentation tasks. We consider each Gcell as a pixel of the image. Unlike images, which typically have only three channels, here the channel count equals the number of features. Fig. 6 shows the input and output details of DRVNet. For model

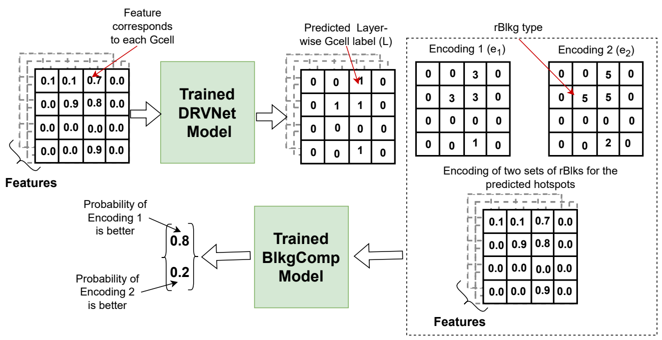

Feature corresponds to each Gcell
Features
Trained DRVNet Model
Predicted Layer-wise Gcell label (L)
rBlkg type
Encoding 1 (e1)
Encoding 2 (e2)
Encoding of two sets of rBlks for the predicted hotspots
Features
Trained BlkgComp Model
Probability of Encoding 1 is better
0.8
0.2
Probability of Encoding 2 is better

Fig. 6: Input and output of DRVNet and BlkgComp model.

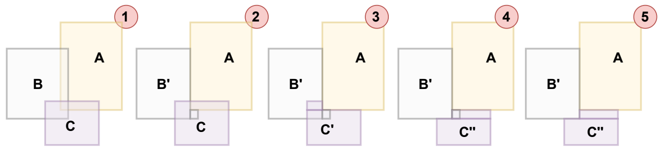


Fig. 7: Removal of routing blockage overlap.

training we use binary cross-entropy with logits loss.

$$loss_{DRVNet} = \log(1 + \exp(-y \cdot z)) \tag{4}$$

Initially, the routing blockages predicted by our DRVNet were not large enough to adequately cover the hotspots. Hence, we compute the mean (μ) and standard deviation (σ) of the negative values predicted by DRVNet, and use a threshold of $\mu + 3\sigma$ (instead of 0) to determine whether a Gcell is a hotspot. After obtaining all the Gcell predicted labels using this threshold, we use Algorithm 1 to generate layer-wise non-overlapping Blkg bounding boxes.

Label generation for DRVNet model. For label generation, we first capture the DRV count in each Gcell and find the DRV hotspots. For hotspot detection, we use DBSCAN [9] clustering over k-means because DBSCAN can identify clusters of varying shapes and does not require a predefined number of clusters. In Algorithm 1:

- **Line 3** executes DBSCAN clustering for the Gcells with DRV count greater than the input *threshold*, using the specified *eps* and *min_size* values. We use *eps* = 3 and *min_size* = 6 as it generates the intended clusters of the DRV markers for our experiments.
- **Lines 4-12** discard clusters that have fewer than 16 Gcells with DRVs, considering these as too small for a hotspot. Conversely, if $cluster_{util}$, the ratio of the number of Gcells in the cluster to the total number of Gcells in the rectangular bounding box of the cluster, is less than 0.05, then we break the cluster as most of the Gcells in the box do not have any DRVs.
- **Line 13** removes overlaps from cluster bounding boxes using the *removeOverlap* function.

The *removeOverlap* function breaks input overlapping boxes into non-overlapping boxes. It first sorts all boxes by area and checks for box overlaps. If two boxes overlap, it breaks the smaller box into one or more boxes that do not overlap with the larger box. *removeOverlap* recursively calls itself until no boxes overlap. Fig. 7 shows *removeOverlap*'s sequential removal of overlaps among blockages A, B and C. It first resolves overlap between A and B by breaking B into smaller boxes. It then removes overlap of A and C by breaking C. The function goes on to similarly eliminate overlaps between B' and other blockages. Fig. 8 illustrates how Algorithm 1 identifies DRV hotspots and creates Blkg outlines from DRV markers.

4.3 BlkgComp Model

We need an evaluator to assess the quality of BlkgSet. A vacuous evaluator would run detailed routing with BlkgSet and obtain a final DRV count. However, this is prohibitively time-consuming. Instead, we train a *BlkgComp* model that lets us evaluate BlkgSets without running the router. As shown in Fig. 6, for a given design and two BlkgSets, the BlkgComp model predicts the probability of which BlkgSet is better. Specifically, for BlkgSets e_1 and e_2 with DRV counts n_1 and n_2, we want our model to predict $p_1 = n_2/(n_1 + n_2)$ and $p_2 = n_1/(n_1 + n_2)$, where p_1 and p_2 indicate the relative quality of e_1 and e_2. Note that we model the comparison of two BlkgSets as a soft classification: the probability value provides more information about the relative quality of the BlkgSets instead of just indicating which one is better. Table 5 in Section 6.1 shows that using soft labels (i.e., continuous probability values) improves the performance of the BlkgComp model compared to using hard labels (i.e., 0 or 1).

We use ResNet-50 with ImageNet pre-trained weights as the base model for our BlkgComp. We update the number of channels in the input convolutional layer to match the number of features. During our model training, we use cross-entropy loss. Additionally, we add a term to the loss function to enforce model symmetry. Specifically, if the outputs corresponding to inputs e_1 and e_2 are o_{11} and o_{12} respectively, and the outputs for inputs e_2 and e_1 are o_{22} and o_{21}, then it is required that $o_{11} = o_{21}$ and $o_{12} = o_{22}$.

$$loss_{entropy} = \sum_{i=1}^{2} \text{crossentropy}(\{p_1, p_2\}, \{o_{i1}, o_{i2}\}) \tag{5}$$

$$loss_{asymmetry} = \sum_{i=1}^{2} MSE(o_{1i}, o_{2i}) \tag{6}$$

We compute $loss_{entropy}$ and $loss_{asymmetry}$ using Eqs. (5) and (6). The total loss for our model training is $loss_{BlkgComp} = loss_{entropy} + \alpha \cdot loss_{asymmetry}$. We found that for $\alpha = 0.1$, our BlkgComp model achieves the best accuracy and Kendall rank, and increasing α slows down the training speed and degrades the performance.

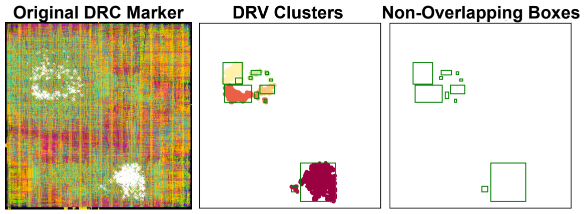


Fig. 8: DBSCAN clustering and Blkg outline creation processes.

Algorithm 1: DRV Hotspot Detection.

```
Input: Indices, threshold = 0, eps = 3, sample_size = 6
Output: Boxes – List of non-overlapping regions
1  Function runDBSCAN(indices, threshold, eps, sample_size):
2      $boxes = []$;
3      $clusters = DBSCAN(indices[\text{'drv'}] > threshold)$;
4      for each cluster in clusters do
5          if Size(cluster) < 16 then
6              continue;
7          $cluster_{util}$ = len(cluster)/#(Gcells in the bounding box of cluster);
8          if $cluster_{util} < 0.05$ then
9              $avg\_drv = average(cluster[\text{'drv'}])$;
10             $boxes.extend(runDBSCAN(cluster, avg\_drv, eps, sample\_size + 1))$;
11             continue;
12         boxes.append(bounding box of cluster);
13     return removeOverlap(boxes);
```

Algorithm 2: Sorting BlkgSets using BlkgComp.

```
Input: $TrainedBlkgComp$, $BlkgSetList$, $Featuers$
Output: $SortedBlkgSetList$ – Sorted BlkgSets based on predicted
        hotspot mitigation capability.
1  Function SortBlkgSets(TrainedBlkgComp, BlkgSetList, Features):
2      $winCount = [0] * len(BlkgSetList)$;
3      for $i = 0; i < len(BlkgSetList) - 1; i++$ do
4          for $j = i + 1; j < len(BlkgSetList); j++$ do
5              $modelInput = \{Features, BlkgSetList[i, j]\}$;
6              $p1, p2 = TrainedBlkgComp(modelInput)$;
7              if $p1 < p2$ then
8                  $winCount[i] += 1$;
9              else
10                 $winCount[j] += 1$;
11     $SortedBlkgSets$ = Sort $BlkgSetList$ based on $winCount$;
```

Algorithm 2 uses the BlkgComp model to sort BlkgSets based on their predicted hotspot mitigation outcomes. In Algorithm 2:

- **Line 2** initializes $winCount = 0$ for all BlkgSets in $BlkgSetList$.
- **Lines 3-10** use the $TrainedBlkgComp$ model to compare all pairs of BlkgSets from $BlkgSetList$, and also keep track of the number of cases in which each BlkgSet has a better probability for congestion mitigation.
- **Line 11** sorts BlkgSets in $BlkgSetList$ in ascending order of the corresponding $winCount$, and returns the $SortedBlkgSetList$.

4.4 RL Model

We require efficient generation of layer-wise BlkgSets for a given design. As our BlkgComp model can only compare two BlkgSets for a given design, we develop an RL model that can efficiently choose an appropriate Blkg for each hotspot and generate BlkgSets to mitigate DRVs. Details of the RL environment are as follows.

- The *environment* includes a trained BlkgComp model, 30 randomly sampled BlkgSets as initial reference, design feature and DRVNet predicted hotspot markers.[4]
- The *state* consists target hotspot encodings predicted using DRVNet, the number of BlkgSets to be sampled, and enumeration of the current samples.
- The *action* space of the RL agent is fixed. The agent chooses a Blkg configuration from a predefined set of configurations that is used to train the BlkgComp model.
- To calculate the *reward*, the environment compares the BlkgSet generated by the RL-agent against reference BlkgSets. The *reward* is then determined based on the *rank* of the RL-generated BlkgSet using the equation: $reward = \frac{15-rank}{15}$.

We use an actor-critic RL framework, with detection transformer [2] as a base architecture. We first use ResNet-50 to extract a 2D representation, then flatten it and pass it through a transformer along with learned positional encodings. The output of the transformer decoder is passed through a shared feed forward network that predicts the Blkg configuration and the corresponding value.

Our RL agent generates a BlkgSet in each step, and a single episode consists of a fixed number of steps (for our experiment, it is five). If the sampled BlkgSet is not the worst one, the environment adds this new BlkgSet to its reference list and removes the lowest rank BlkgSet. At the end of the episode, the environment resets the reference BlkgSets to their initial state. We use the Monte Carlo reward update with $\gamma = 0.9$. If the agent samples the same BlkgSet, we penalize it by setting $reward = -2$ for each duplicate BlkgSet. For a given state s_t, actual return G_t, estimated value from the critic $v(s_t)$ and action a_t, actor and critic loss are defined in Eqs. (7) and (8), respectively.

$$L_{actor} = -\log(\pi(a_t|s_t)) \cdot (G_t - v(s_t)) - \lambda H \quad (7)$$

$$L_{critic} = (v(s_t) - G_t)^2 \quad (8)$$

Here λ is the entropy weight, and H is the entropy of the policy distribution as shown in Eq. (9). We add the regularization term λH to the actor loss to promote exploration by discouraging the policy from being deterministic.

$$H = -\sum_{a \in A} \pi(a|s_t) \log(\pi(a|s_t)) \quad (9)$$

During our model training, we reduce the entropy weight linearly from 1.0 to 0.01 to allow for initial exploration.

[4]For the given hotspots, we use 30 randomly sampled BlkgSets as a reference to compute the rank of the RL-generated routing blockage. Here, if we increase the number of BlkgSets in our reference list, then the rank predicted using Algorithm 2 will be more reliable, but the runtime for rank computation will increase. We observe that for 30 BlkgSets, the computed rank is reliable while the runtime remains low.

5 EXPERIMENTAL SETUP

Our experimental setup spans data generation, model training, baseline setup, and evaluation flow, all of which are public in [32].

5.1 Data Generation

We use the PROBE method [8, 15] to generate standard cell-based testcases with DRV violations, and modified Artificial Netlist Generator (ANG) [16, 33] code to create small design datasets with macros.

Table 1: PROBE [8, 15] parameters for data generation.

Parameters	Values	Description
$Size$	100, 150, 200	Size of the square mesh
$Size_{sub}$	10, 15, 20, 25, 30	Size of the square subregion
$\#Subregion$	[0-4]	Number of the subregion
$\#locations$	[4, 8]	Number of location combination
K_1	[0-5]	Normalized swap count for the whole region
K_2	[3, 5, 8, 10, . . , 20]	Normalized swap count for the subregion

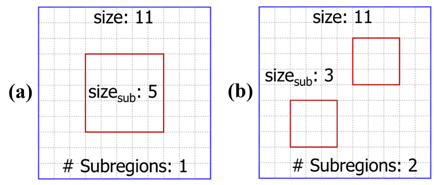
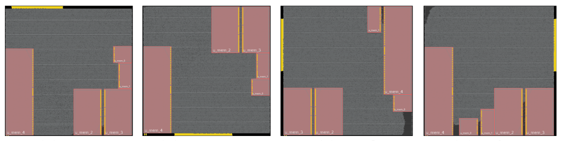


Fig. 9: Visualization of subregions according to parameter settings. (a) #Subregion=1, $size_{sub}$=5; (b) #Subregion=2, $size_{sub}$=3.

PROBE. We apply PROBE [8] with a 3-input base cell and knight's tour-based topology, using parameters in Table 1. We create routing (DRV) hotspots with *subregion swapping*, i.e., additional random neighbor swaps in designated subregions created according to Table 1 parameters, as illustrated in Fig. 9. As documented in [32], training data uses $size_{sub} = \{10, 20, 30\}$, and testing data uses $size_{sub} = \{15, 25\}$. When $\#Subregion = 1$, we use $\#locations = 4$. When $\#Subregion > 1$ and there are multiple (randomly chosen) $size_{sub}$ values, we use $\#locations = 8$.

ANG. Our studies in NG45/GF12 below use Arm CA53 to reflect real designs with macros. As CA53 is too large for training data generation, we augment ANG [16] to create artificial netlists with macros that are 20% the size of CA53, while maintaining the same combinational cell ratio and distributions of net bounding box, path depth, and net degree. As detailed in [32], for a prescribed selection of macros, we (i) calculate total macro input (SUM_{input}) and output (SUM_{output}) pins; (ii) add SUM_{input} (SUM_{output}) to the block's target output (input) pin count; (iii) generate an artificial netlist by



(a) Top (b) Bottom (c) Left (d) Right

Fig. 10: Macro locations change with IO pin locations.

running ANG; and (iv) connect the netlist's output (input) pins to macro input (output) pins. Cases with DRVs are obtained by varying design aspect ratio, utilization, and IO pin locations. Fig. 10 shows how macro locations change with IO pin locations.

Data for BlkgComp model training. BlkgComp modeling requires data with Blkg-induced DRV labels. We add Blkg impact data to PROBE-based data used for DRVNet modeling, via four steps. (i) From the PROBE-based data used for DRVNet, we sample 400 data points (300 for training, 100 for testing) with DRV count between 800 and 3,000. (ii) We use DBSCAN to create Blkg outlines and extract layer information (l_i) from DRV markers. (iii) For BlkgComp data generation, the Blkg space comprises six fBlkgs with partial densities ranging from 70 to 95 with step 5, and eight cBlkgs with configurations {(80, 98, 10, l_i), (80, 99, 20, l_i), (70, 97, 10, l_i), (70, 98, 15, l_i), (98, 90, 10, l_i), (99, 80, 20, l_i), (97, 70, 10, l_i), and (98, 70, 15, l_i)}. (iv) We limit the number of sampled BlkgSets for data point i as $\#BlkgSet_i = min(20, \#hotspot_i \cdot 5)$. (iv) We generate data for BlkgComp from the sampled BlkgSets (4,500 for training, 1,500 for testing). For GF12 modeling, we apply transfer learning from NG45 to GF12 and fine-tune the model using GF12 data. With this strategy, (training, testing) dataset sizes in GF12 are (75, 30) for PROBE-based data, and (820, 280) for sampled BlkgSets.

5.2 Model Training

We implement our models using *PyTorch* 2.0 and train them on a server with 2.25 GHz AMD EPYC 7742 processor, 512 GB RAM, and four NVIDIA A100 80GB GPUs. Details of the training are as follows.

DRVNet. For training the DRVNet model, we employ the Adam optimizer with a learning rate of 0.0001 and a batch size of 64, utilizing binary cross-entropy with logits loss. The model is trained over 90 epochs, and we save the version that achieves the highest F1-score on the validation dataset. The total training time for our model is approximately 120 minutes for NG45. Additionally, we employ a decaying learning rate strategy with a decay factor of 0.5, monitoring the loss on the validation dataset. For GF12, we begin with the DRVNet model initially trained on the NG45 dataset and then fine-tune it using the GF12 dataset.

BlkgComp. For the BlkgComp model, the training setup mirrors that of DRVNet with a few adjustments: we use a learning rate of 0.001 and a batch size of 128. The model is saved based on its performance as measured by the Kendall rank correlation coefficient on the validation dataset. The training time is approximately 120 minutes on NG45 dataset. As with DRVNet, we fine-tune the BlkgComp model initially trained on the NG45 dataset, using the GF12 dataset.

RL Model. For the RL model, we perform online training using three PROBE designs, and employ the Adam optimizer with a learning rate of 0.0001. We use a linearly decaying entropy weight, starting at 0.5 and decreasing to 0.01, to train the model over 90 episodes, which takes approximately 90 minutes. We save the weights that deliver the highest combined reward across the three designs. For design-specific fine-tuning, we train the RL model for an additional 10 episodes, which takes around 5 minutes.

5.3 Baseline Setup and Evaluation Flow

We now provide details of the designs and enablements, baselines, and the evaluation flow used to assess our proposed framework.

Designs and enablements. We evaluate our framework on two technology platforms: the open-source PDK NG45 and the industry PDK GF12. We test BlkgSets across three designs (#instance, #macro): NOVA (140k, 0), LDPC (50k, 0) and CA53 (300k, 25).

Baseline data setup. To evaluate the output of our framework, we use two baselines: the default tool flow (without Blkg) and a human baseline. For the human baseline, a human expert determines the Blkg outline and layer based on post-route DRV markers from the default tool flow. Several BlkgSets are created by varying the sizes of Blkgs according to the DRV distribution (examples of human expert-generated BlkgSets are shown in Fig. 13). We then evaluate all the human-created BlkgSets and set the BlkgSet with the fewest DRVs as our human baseline. Table 2 shows example data from layer-wise BlkgSet tests, with 'O' representing associated layers.

Table 2: Human baseline layer-wise Blkg data (NG45-CA53).

M2	M3	M4	M5	M6	M7	M8	M9	M10	#DRV
O	O	O	O	O	O	O	O	O	1,175
	O		O		O		O		1,038
O		O		O		O		O	863
			O	O					1,612
O	O								750
O	O	O	O						701
O	O	O	O	O					646
O	O		O	O					**519**

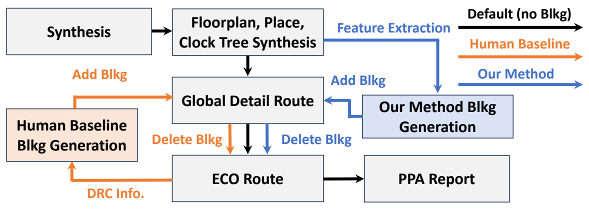


Fig. 11: Assessing default, human baseline, and our methods.

Evaluation flow. Fig. 11 shows the evaluation flow for three methods: default, human baseline, and our framework. Note that the human expert uses the post-ECO-route DRV markers (location and layer) from the default flow to generate the BlkgSets, whereas our framework generates BlkgSets based on the post-CTS information.

6 EXPERIMENTAL RESULTS

We now present the performance of our DRVNet, BlkgComp, and RL models across various design-specific choices, followed by results from our framework, ablation studies, and a robustness test.

6.1 ML Model-Specific Choices

In this subsection, we discuss for DRVNet (i) the application of cross-entropy class weight and (ii) the base model selection. For BlkgComp, we examine (iii) the effectiveness of soft classification, (iv) the base model selection, and (v) the top-k accuracy of the BlkgComp model. We then present (vi) the training results of the RL model.

Cross-entropy class weight. The training dataset contains a small fraction of Gcells in hotspots, leading to an imbalance. To address this, Gcells in hotspots are given additional weight during training. However, excessively high weights increase the false positive rate (FPR), while too low weights reduce the true positive rate (TPR). Therefore, finding the optimal weight is crucial. In Table 3, the best results in terms of TPR and F1-score are observed for a weight of 5. We use this weight for our DRVNet model training.

Choice of base model for DRVNet. For the DRVNet, we have trained three base models: U-Net, RouteNet, and Detection Transformer (DETR) [2]. Since U-Net achieves the best results over RouteNet and DETR, as shown in Table 4, we use U-Net as the base

Table 3: Class weight effect on DRVNet model.

Enablement	Weight	TPR	FPR	F1	Accuracy
NG45	1	0.79	0.01	**0.80**	0.98
	2	0.82	0.01	0.79	0.98
	5	**0.83**	0.01	0.79	0.98
	10	0.81	0.01	0.78	0.98
GF12	1	0.69	0.00	0.73	0.99
	2	0.72	0.00	0.73	0.99
	5	0.75	0.00	**0.74**	0.99
	10	**0.77**	0.00	0.73	0.99

model for DRVNet. For GF12 enablement, when we train U-Net from scratch, we observe that the final F1 score is 0.72. This suggests that transfer learning from NG45 helps achieve better results.

Table 4: Performance of different DRVNet base model.

Enablement	Base	TPR	FPR	F1	Accuracy
NG45	U-Net	0.83	0.01	**0.79**	0.98
	RouteNet	0.96	0.04	0.65	0.97
	DETR	0.94	0.03	0.71	0.97
GF12	U-Net	0.75	0.00	**0.74**	0.99
	RouteNet	0.81	0.01	0.55	0.94
	DETR	0.73	0.00	0.72	0.99

Effectiveness of soft classification for BlkgComp. We have trained the BlkgComp model using hard labels and soft labels. Results in Table 5 indicate that using soft labels leads to better results in terms of accuracy and Kendall rank.

Table 5: BlkgComp model evaluation for different labels.

Enablement	Labels	Accuracy	Kendall Rank
NG45	Soft	0.73	**0.53**
	Hard	0.70	0.49
GF12	Soft	0.68	**0.32**
	Hard	0.68	0.31

Choice of base model for BlkgComp. We study each of ResNet-50, MobileNet-V2, and Vision Transformer (ViT) as a candidate base model and use pretrained weights from ImageNet. Since we achieve better accuracy and Kendall rank with the ResNet-50 base model (Table 6), we use ResNet-50 for all of our reported experiments.

Table 6: Performance of different base model for BlkgComp.

Enablement	Labels	Accuracy	Kendall Rank
NG45	ResNet-50	0.73	**0.53**
	MobileNet-V2	0.73	0.51
	ViT	0.73	0.42
GF12	ResNet-50	0.68	**0.32**
	MobileNet-V2	0.67	0.31
	ViT	0.68	0.31

Top-5 Accuracy of BlkgComp. Top-5 accuracy indicates the probability of finding the best BlkgSets within the top-5 BlkgSets predicted using the BlkgComp model. To find the top-5 accuracy, we run a Monte Carlo simulation with 10,000 samples. For 30 BlkgSets, our simulation shows a top-5 accuracy of 0.91 at 0.73 accuracy, and 0.85 at 0.68 accuracy. This indicates that the reward computed using the BlkgComp model is sufficient for the RL model.

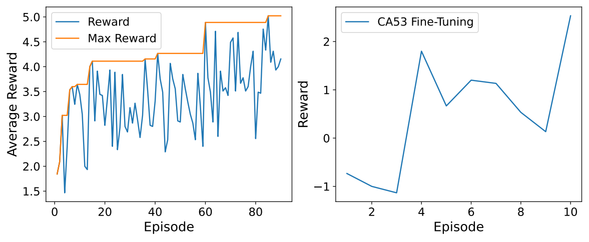


Fig. 12: RL pre-training (left) and fine-tuning (right) curves.

Performance of RL model. Fig. 12 shows the training curve of the RL model on PROBE (NG45) designs and the fine-tuning curve for the CA53 (NG45) design. Here, a human expert has generated Blkg boundaries and layer information for CA53. For the PROBE designs, the best reward of the model is around five, suggesting that almost all the five sampled BlkgSets are predicted to be better than the reference BlkgSets. For episode 0, the CA53 design shows negative rewards; however, after fine-tuning, two out of five RL-generated BlkgSets outperform all the reference BlkgSets.

6.2 Performance of Our Framework

We report model performance for NOVA, LDPC and CA53 designs in NG45 and GF12. For each design, we fine-tune the RL-model for 10 episodes and select the top-5 BlkgSets generated by the RL-agent using our BlkgComp model. As shown in Fig. 11, we then run *globalDetailedRoute* with each BlkgSet, remove the Blkgs, run *ecoRoute -fix_drc* and report the final DRV and PPA. Table 7 details our model performance as compared to the default tool flow (i.e., when no Blkgs are used) and human-generated BlkgSets. We highlight the best results with blue font. For GF12, wirelength (WL) and total power are normalized with respect to the default tool flow, and worst negative slack (WNS) and total negative slack (TNS) are normalized to the target clock period.[5] Fig. 13 presents DRV markers and Blkg outlines for CA53 (NG45) and LDPC (GF12).

- In NG45, our framework-generated BlkgSet reduces DRV up to 88% (avg 59%) compared to the default tool flow and up to 21% (avg 6%) compared to the human baseline.
- In GF12, our framework-generated BlkgSet reduces DRV up to 80% (avg 68%) compared to the default tool flow and up to 4% (avg -2%) compared to the human baseline.
- In GF12, adding routing blockages improves wirelength and total power. Our BlkgSet achieves up to 1.0% (avg 0.9%) and 2.0% (avg 1.3%) reductions in wirelength and total power, respectively, when compared to the default tool flow. Compared to human baselines, our BlkgSet yields similar wirelength and total power values.
- We observe that the blockage boundaries generated by DRVNet do not adequately cover the hotspots for LDPC (NG45, GF12) and NOVA (GF12) designs due to the highly scattered DRVs, preventing our framework from outperforming the human baseline.

We emphasize that for human baselines, expert humans generate routing blockages *by analyzing final post-route DRV location and layer information* from the default tool flow, and then creating a BlkgSet for each hotspot. In contrast, our approach generates BlkgSet based on post-CTS features. We reduce turnaround time while producing BlkgSets for DRV mitigation that are similar to or better than those created manually, by experts who run the detailed router.

Table 7: Evaluation of our framework.

Tech	Design	Method	#DRV	WL (mm)	WNS (ns)	TNS (ns)	Power (mW)
NG45	NOVA	No Blkg	2,003	**3,874**	**-0.634**	**-419**	**257.3**
		Human	898	4,016	-0.752	-512	259.8
		Ours	**814**	3,931	-0.688	-435	258.1
	LDPC	No Blkg	830	**1,739**	**-0.209**	-256	**240.1**
		Human	**516**	1,741	-0.277	**-245**	240.3
		Ours	583	1,739	-0.215	-261	**240.1**
	CA53	No Blkg	3,308	**11,344**	**-0.311**	**-690**	**484.1**
		Human	519	11,512	-0.571	-699	485.7
		Ours	**409**	11,418	-0.480	-719	484.9
GF12	NOVA	No Blkg	1,242	1.000	-0.562	-782	1.000
		Human	**403**	0.994	-0.655	-707	**0.994**
		Ours	442	**0.990**	**-0.475**	**-186**	**0.994**
	LDPC	No Blkg	1,739	1.000	-0.583	-632	1.000
		Human	**345**	**0.990**	-0.472	**-604**	**0.980**
		Ours	346	**0.990**	**-0.438**	-717	**0.980**
	CA53	No Blkg	3,316	1.000	**-0.369**	-1,495	1.000
		Human	942	0.992	-0.615	-783	**0.986**
		Ours	**902**	**0.991**	-0.454	**-696**	**0.986**

[5]Our WNS values are high because we do not run post-route optimization, as we evaluate on a fixed placement.

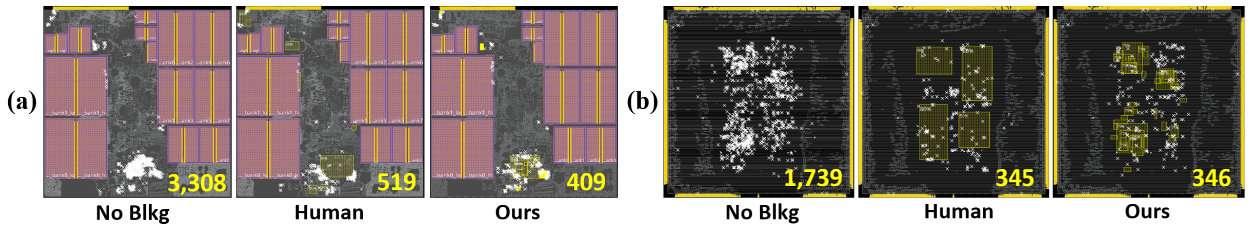


Fig. 13: Comparison of DRV markers (white), DRV counts, and Blkg outlines (yellow). (a) CA53 in NG45; (b) LDPC in GF12.

6.3 Ablation Studies

In this subsection, we conduct ablation studies to show the importance of various features in our model training and to assess the performance of our RL model for zero-shot inference and fine-tuning.

Feature importance. To examine feature importance, we systematically remove one feature at a time and observe the impact on the performance of the DRVNet model. The tomography feature includes both vertical and horizontal layer usage when the available routing resource for eGR is below 100%. The layer-wise feature includes details specific to each layer. The eGR all layer feature includes information on layer-wise routing resources in scenarios where there is no reduction in routing resources. Additional features evaluated include pin density and cell density. Table 8 shows that layer-wise information is crucial, and tomography features offer additional benefits for DRVNet. However, eGR across all layers, pin, and cell density have minimal impact, likely because tomography features and layer-wise information serve as proxies for these attributes.

Table 8: Ablation study of DRVNet feature importance (NG45).

Tomography	✓		✓	✓	✓	✓
Cell Density	✓	✓		✓	✓	✓
Pin Density	✓	✓	✓		✓	✓
Layer-wise	✓	✓	✓	✓		✓
eGR All Layer	✓	✓	✓	✓	✓	
F1	0.79	0.77	0.79	0.79	0.68	0.79
Accuracy	0.98	0.98	0.98	0.98	0.96	0.98
TPR	0.83	0.88	0.83	0.83	0.96	0.83
FPR	0.01	0.02	0.01	0.01	0.04	0.01

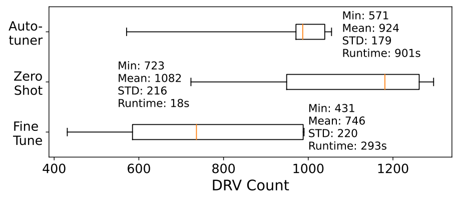


Fig. 14: DRV counts of solutions generated using autotuner, RL-zero shot, and RL-fine-tuning for CA53 in NG45.

Autotuner vs. RL zero-shot vs. RL fine-tuning. We compare the performance of three different routing blockage generation methods – autotuner, RL zero-shot, and RL fine-tuning – using CA53 in NG45 with human-generated Blkg boundary and layer information. For autotuner, we employ HyperOpt [1] search to sample different BlkgSets for the given layer-wise hotspots, conducting 150 iterations that take approximately 15 minutes. For RL, we use the base model pre-trained for 90 minutes on the PROBE design. The RL-zero shot takes 18 seconds to generate five BlkgSets for our testcase. We then fine-tune it over 10 episodes, which takes about 5 minutes. For all three methods, we select the top five sampled BlkgSets and run our evaluation flow. Fig. 14 shows that the DRV violation counts for the best solutions from autotuner, RL zero-shot, and RL fine-tuning are 571, 723, and 431, respectively. Despite having an initial training cost, RL fine-tuning outperforms autotuner in terms of both DRV count and runtime.

6.4 Robustness Tests

Our robustness tests perturb BlkgSets and baseline design settings.

Perturbation on BlkgSets. We assess robustness of our generated BlkgSets by applying the following perturbations selected uniformly at random from the corresponding intervals: (i) shift the predicted Blkgs both horizontally and vertically by $[-2, 2]$ Gcells; (ii) change Blkg height and width by $[-2, 2]$ Gcells; and (iii) vary partial density by $[-2, 2]\%$. Across five randomly perturbed BlkgSets, we observe (min, avg, max) DRV counts of (491, 558, 632) for CA53 in NG45 (ref: 409 DRVs), and (400, 423, 475) for LDPC in GF12 (ref: 346 DRVs).

Perturbation on design settings. We apply our Blkg generation framework to different design settings. We perturb the floorplan aspect ratio (AR) and utilization of the design by small amounts (1% in AR, 0.1% in utilization), generate different post-CTS solutions, and then apply our framework. Table 9 shows that our framework obtains similar reductions in DRV count (min 78%, max 92%) compared to baselines for both CA53 in NG45 and LDPC in GF12.

Table 9: Performance on different design versions.

Design	Method	DRV Count				
		Ref.	AR_1	AR_2	$Util_1$	$Util_2$
CA53 (NG45)	Baselines	3,308	1,880	2,070	3,522	6,344
	Ours	409	336	294	427	1,262
	%Reduction	88%	82%	86%	88%	80%
LDPC (GF12)	Baselines	1,739	1,765	6,187	1,401	1,155
	Ours	346	371	522	267	251
	%Reduction	80%	79%	92%	81%	78%

7 CONCLUSION

We have introduced a *placement tomography*-based Blkg generation framework, which generates BlkgSets after the CTS stage. Our framework comprises three main models. (i) DRVNet generates blockage outline and layer information based on hotspot predictions. (ii) BlkgComp assesses the quality of BlkgSets by comparing two BlkgSets and predicting the probability of one BlkgSet winning over the other. (iii) Our RL model efficiently chooses an appropriate Blkg for each hotspot and generates BlkgSets to be compared using BlkgComp. The DRVNet model achieves an F1 score of up to 0.79 and accuracy up to 0.99. The BlkgComp model achieves accuracy up to 0.74, top-5 accuracy up to 0.91 and a Kendall rank of up to 0.52. Overall, our framework generates BlkgSets that reduce DRVs by up to 88% compared to the no-blockage tool default, and up to 21% compared to an expert human baseline that accesses post-route DRV information. Ongoing research seeks improved ML-based Blkg outline and layer prediction in DRVNet, for improved BlkgSet generation. Enhancing BlkgComp so that it can compare BlkgSets with different outlines will also enhance our mitigation of post-route DRVs. We also seek to confirm effectiveness of our approach using the OpenROAD tool [36].

ACKNOWLEDGMENTS

This work is supported by Apple; we thank Varun Garg, Dr. Abdelrahman Hosny, Dr. Yi-Chen Lu, and Ala Qumsieh for their constructive feedback. Our research on ML EDA is also broadly supported by Samsung Advanced Institute of Technology.

REFERENCES

[1] J. Bergstra, D. Yamins and D. Cox, "Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures", *Proc. ICML*, 2013.

[2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov and S. Zagoruyko, "End-to-End Object Detection with Transformers", *Proc. ECCV*, 2020, pp. 213-229.

[3] C.-C. Chang, J. Pan, T. Zhang, Z. Xie et al., "Automatic Routability Predictor Development Using Neural Architecture Search", *Proc. ICCAD*, 2021.

[4] W.-T. J. Chan, P.-H. Ho, A. B. Kahng and P. Saxena, "Routability Optimization for Industrial Designs at Sub-14nm Process Nodes Using Machine Learning", *Proc. ISPD*, 2017, pp. 15-21.

[5] X. Chen, Z. Di, W. Wu, Q. Wu, J. Shi and Q. Fang, "Detailed Routing Short Violation Prediction Using Graph-Based Deep Learning Model", *IEEE TCAS* 69(2) (2022).

[6] R. Chen, W. Zhong, H. Yang, H. Geng, X. Zeng and B. Yu, "Faster Region-Based Hotspot Detection", *Proc. DAC*, 2019.

[7] J. Chen, J. Kuang, G. Zhou, D. J.-H. Huang and E. F. Y. Young, "PROS: A Plug-in for Routability Optimization Applied in the State-of-the-Art Commercial EDA Tool Using Deep Learning", *Proc. ICCAD*, 2020.

[8] C.-K. Cheng, A. B. Kahng, H. Kim, M. Kim, D. Lee, D. Park and M. Woo, "PROBE2.0: A Systematic Framework for Routability Assessment from Technology to Design in Advanced Nodes", *IEEE TCAD* 41(5) (2022), pp. 1495-1508.

[9] M. Ester, H. P. Kriegel, J. Sander and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", *Proc. KDD*, 1996, pp. 226-231.

[10] A. Ghose, V. Zhang, Y. Zhang, D. Li, W. Liu and M. Coates, "Generalizable Cross-Graph Embedding for GNN-Based Congestion Prediciton", *Proc. ICCAD*, 2021.

[11] W.-T. Hung, J.-Y. Huang, Y.-C. Chou, C.-H. Tsai and M. Chao, "Transforming Global Routing Report into DRC Violation Map with Convolutional Neural Network", *Proc. ISPD*, 2020.

[12] T. Jindal, C. J. Alpert, J. Hu, Z. Li, G. J. Nam and C. B. Winn, "Detecting Tangled Logic Structures in VLSI Netlists", *Proc. DAC*, 2010.

[13] A. B. Kahng, "Solvers, Engines, Tools and Flows: The Next Wave for AI/ML in Physical Design", *Proc. ISPD*, 2024.

[14] A. B. Kahng and X. Xu, "Accurate Pseudo-Constructive Wirelength and Congestion Estimation", *Proc. SLIP*, 2003, pp. 61-68.

[15] A. Kahng, A. Kahng, H. Lee and J. Li, "PROBE: A Placement, Routing, Back-End-of-Line Measurement Utility", *IEEE TCAD* 37(7) (2017), pp. 1459-1472.

[16] D. Kim, S.-Y. Lee, K. Min and S. Kang, "Construction of Realistic Place-and-Route Benchmarks for Machine Learning Applications", *IEEE TCAD* 42(6) (2022), pp. 2030-2042.

[17] Z.-H. Lee, C.-H. Lu, H.-H. Pan, T.-C. Wang et al., "A Robust Routing Guide Generation Approach for Mixed-Size Designs", *Proc. MLCAD*, 2023.

[18] R. Liang, H. Xiang, D. Pandey, L. Reddy, S. Ramji, G.-J. Nam and J. Hu, "DRC Hotspot Prediction at Sub-10nm Process Nodes Using Customized Convolutional Network", *Proc. ISPD*, 2020.

[19] L. Li, Y. Cai and Q. Zhou, "An Efficient Approach for DRC Hotspot Prediction with Convolutional Neural Network", *Proc. ISCAS*, 2021.

[20] J. Liu, C.-W. Pui, F. Wang and E. F. Y. Young, "CUGR: Detailed-Routability-Driven 3D Global Routing with Probabilistic Resource Model", *Proc. DAC*, 2020.

[21] B. Liu, C. Qiao, N. Xu, X. Geng, Z. Zhu and J. Yang, "Variational Label-Correlation Enhancement for Congestion Prediction", *Proc. ASP-DAC*, 2024, pp 466-471.

[22] J. Lou, S. Krishnamoorthy and H. S. Sheng, "Estimating Routing Congestion Using Probabilistic Analysis", *Proc. ISPD*, 2001.

[23] K. Min, S. Kwon, S.-Y. Lee, D. Kim, S. Park and S. Kang, "ClusterNet: Routing Congestion Prediction and Optimization Using Netlist Clustering and Graph Neural Networks", *Proc. ICCAD*, 2023, pp 1-8.

[24] S. Park, D. Kim, S. Kwon and S. Kang, "Routability Prediction and Optimization Using Explainable AI", *Proc. ICCAD*, 2023, pp 1-8.

[25] O. Ronneberger, P. Fischer and T. Box, "U-Net: Convolutional Networks for Biomedical Image Segmentation", *Proc. MICCAI*, 2015, pp. 234-241.

[26] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov and L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks", *Proc. CVPR*, 2018, pp. 4510-4520.

[27] P. Spindler and F. M. Johannes, "Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement", *Proc. DATE*, 2007.

[28] M. Su, H. Ding, S. Weng, C. Zou, Z. Zhou, Y. Chen, J. Chen and Y.-W. Chang, "High-Correlation 3D Routability Estimation for Congestion-Guided Global Routing", *Proc. ASPDAC*, 2022.

[29] B. Wang, G. Shen, D. Li, J. Hao et al., "LHNN: Lattice Hypergraph Neural Network for VLSI Congestion Prediction", *Proc. DAC*, 2022.

[30] Z. Xie, Y.-H. Huang, G.-Q. Fang, H. Ren et al., "RouteNet: Routability Prediction for Mixed-Size Designs Using Convolutional Neural Network", *Proc. ICCAD*, 2018.

[31] S. Zheng, L. Zou, P. Xu, S. Liu, B. Yu and M. Wong, "Lay-Net: Grafting Netlist Knowledge on Layout-Based Congestion Prediction", *Proc. ICCAD*, 2023, pp. 1-8.

[32] Tomography GitHub. https://github.com/ABKGroup/Tomography

[33] Artificial Netlist Generator. https://github.com/daeyeon22/artificial_netlist_generator

[34] Cadence Innovus v21.1. https://cadence.com

[35] Cadence Innovus User Guide. https://cadence.com

[36] The OpenROAD Project (GitHub). https://theopenroadproject.org and https://github.com/The-OpenROAD-Project/OpenROAD

[37] Nangate45 PDK. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts/tree/master/flow/platforms/nangate45