

# Accelerating Detailed Routing Convergence through Offline Reinforcement Learning

Afsara Khan

*Electrical and Computer Engineering  
New York University  
Brooklyn, NY, USA  
atk331@nyu.edu*

Austin Rovinski

*Electrical and Computer Engineering  
New York University  
Brooklyn, NY, USA  
rovinski@nyu.edu*

**Abstract**—Detailed routing remains one of the most complex and time-consuming steps in modern physical design due to the challenges posed by shrinking feature sizes and stricter design rules. Prior detailed routers achieve state-of-the-art results by leveraging iterative pathfinding algorithms to route each net. However, runtimes are a major issue in detailed routers, as converging to a solution with zero design rule violations (DRVs) can be prohibitively expensive.

In this paper, we propose leveraging reinforcement learning (RL) to enable rapid convergence in detailed routing by learning from previous designs. We make the key observation that prior detailed routers *statically* schedule the cost weights used in their routing algorithms, meaning they do not change in response to the design or technology. By training a conservative Q-learning (CQL) model to dynamically select the routing cost weights to minimize the number of algorithm iterations, we find that our work completes the ISPD19 benchmarks with  $1.56\times$  average and up to  $3.01\times$  faster runtime than the baseline router while maintaining or improving the DRV count in all cases. We also find that this learning shows signs of generalization across technologies, meaning that learning designs in one technology can translate to improved outcomes in other technologies.

**Index Terms**—detailed routing, reinforcement learning

## I. INTRODUCTION

Detailed routing remains one of the most challenging aspects of modern physical design due to the increasing complexity of scaling feature sizes. As technology nodes advance, the number of design rules increases dramatically and leads to significantly increased runtime for detailed routing.

Prior work in detailed routing has leveraged techniques such as Lee’s Algorithm [1], A\* search [2], line search [3], and others to perform detailed routing. Most notably, Dr. CU [4] and OpenROAD [5], [6] are the leading open-source detailed routers capable of solving the ISPD18 and ISPD19 detailed routing benchmarks. In both cases, the general routing algorithms are the same: 1) A pathfinding algorithm such as A\* or Dijkstra’s algorithm is used to find the minimum cost path for each routed net, 2) a design rule check (DRC) is run to evaluate any DRVs, 3) violation costs are added to the corresponding nodes on the grid, and 4) the route is ripped up and rerouted to avoid violations. An “iteration” is defined as the number of times the algorithm must ripup and reroute nets to achieve a DRV-free solution, which can become prohibitively expensive for high-density designs.

Our major insight is that prior work only performs *static* costing of search weights, that is, the costs that are applied to violations do not change in response to the design. In the

case of Dr. CU, the violation costs are static during the entire algorithm. For OpenROAD, the costs are adjusted based solely on the current ripup iteration number.

One of the great challenges in trying to improve over the baseline implementations is that altering the violation costing can frequently lead the detailed router towards worse solutions which take longer to converge or do not converge to 0 DRVs. During our study, we find that random exploration of weights leads to a significantly higher number of worse solutions than better solutions, and weights which work well on one design may not translate to another design. Therefore, the challenge in improving weight selection relies on both 1) finding weights which enable the router to converge to 0 DRVs and 2) converging to 0 DRVs in fewer iterations than the baseline approach.

In this work, we propose using reinforcement learning (RL) to learn from prior design implementations and determine sequences of weights which will converge to 0 DRVs in the fewest number of iterations. We use perturbation data sampling [7], [8] to explore 350-400 weight sequences per design in our training set, and then we train a conservative Q-learning (CQL) model [9] on these samples to predict the best weights for the next iteration in order to minimize iterations to convergence. We find that on the ISPD19 benchmark, our work converges in 5% fewer iterations on average and up to 31%. This translates to a runtime speedup of  $1.56\times$  on average and up to  $3.01\times$ .

One of our key results is that we find that not only are the current weight schedules suboptimal, but that the conventional wisdom for weight selection is *dramatically different* from the weights that our RL model selects for improved results. We make the following contributions in this work:

- Development of a detailed routing training dataset using perturbation sampling and random exploration techniques
- Identification of critical state variables and reward functions suitable for RL models in detailed routing
- Implementation and evaluation of an offline RL approach using the conservative Q-learning (CQL) model to predict optimal cost weights for detailed routing.
- Demonstration of robustness across diverse designs
- Analysis of cost weights vs. routing performance to demonstrate that learned weights can achieve better performance by disregarding conventional practice.
- We aim to open-source all source code for training and inference along with publication of this paper.

The rest of the paper is organized as follows. Section II discusses related work, Section III discusses our methodology for data generation, Section IV discusses the model architecture and training, Section V discusses the experimental results, and Section VI provides our conclusions.

## II. RELATED WORK

### A. Detailed Routers

The foundation of modern detailed routing traces back to Lee’s maze routing algorithm [1], a breadth-first search (BFS) that guaranteed minimum cost paths. Rapid notable improvements on this base algorithm included the A\* search using heuristics to guide the search [2], [10], along with techniques like line search [3] to speed up execution. Core strategies in modern routers also include the iterative rip-up and reroute and sometimes leverage multicommodity flow concepts [11]. TritonRoute [12] utilizes the prior findings to employ an iterative A\*-based search with partitions, enhanced by dynamic boundary adjustments. Similarly, Dr. CU [4] uses Dijkstra’s algorithm coupled with a sparse grid-graph and partitioning to efficiently route nets. As newer technology nodes arrive, routing research has shifted focus towards techniques such as gridless pin access [13], routing under complex patterning like SADP [14], [15], minimum area-sensitive path search [16], and ILP-based formulations [11].

In terms of academic routers, OpenROAD [6] provides state-of-the-art performance, achieving 0 DRVs on the ISPD18 [17] benchmark and 0 DRVs on all but one test case on ISPD19 [18]. OpenROAD’s router is derived from TritonRoute [12], [19], but several enhancements have been made and it is actively maintained to continue improving detailed routing results. In this work, we propose using dynamic weighting of costs which are learned from prior designs. Prior works use a static scheduling of weights which do not depend on the design. Additionally, we find that RL-generated weights tend to differ *significantly* from the expert-curated weights in the baseline implementations.

### B. Machine Learning Techniques in Routing

Several prior works have shown benefits from applying traditional machine learning to global and detailed routing. Many works have focused on congestion and DRV hotspot prediction at the routing level [20], [21], while others have focused on predicting and avoiding pin access violations [22]. These predictions typically required secondary mechanisms to influence routing behavior. More direct guidance has been explored via reinforcement learning. For example, Chen et al. [23] propose an online RL framework that combines graph neural networks (GNNs) with PPO policy learning.

Our work differs significantly from prior works as we propose layering an RL model on top of the iterative search in order to reach DRV convergence faster. Our technique is applicable to any router which uses violation costing for its search, and our technique is agnostic to technology node. To the best of our knowledge, no prior work has used RL to learn across detailed routing iterations to improve convergence.

## III. DATA GENERATION

As the first step in training our RL model, we first select the data which represents the state of the design at the end of an iteration. We captured the following values:

- **Input Weights:** the weights applied to violations during the current iteration. We use 4 distinct weights: **drvCost:** cost applied to potential DRVs. **markerCost:** penalty to nodes adjacent to DRVs. **fixedShapeCost:** cost applied nodes which violate fixed objects, such as macro pins or obstructions. **markerDecay:** the decay rate of DRV costs from previous routing iterations.
- **Partition DRVs:** the number of DRVs for each chip routing partition at the end of the iteration.
- **Maximum Partition DRV:** the maximum number of DRVs across all partitions at the end of the iteration.
- **Neighbor DRVs:** For each partition, the number of DRVs in the adjacent partitions
- **Total DRVs:** the total DRVs at the end of the iteration
- **Total Wirelength:** total wirelength in um at the end of the iteration

Several of the values are derived from the same source data, however we capture them individually in order to simplify model training and reduce model complexity. More details on these parameters are discussed in prior work [12].

After each iteration, we sample each of these values as part of the state  $s$ . A “sequence”  $S$  is formed by a series of states  $\{s_1, s_2, \dots, s_n\}$  that result in DRV convergence, where  $n$  is the number of iterations. Each sequence forms a data point which is used to train our RL model. Because states are deterministic, it is possible to represent a state simply by its sequence of input weight vectors. This is illustrated in Fig. 1 (top).

In addition to capturing the sequential history, we captured the following static design characteristics at the beginning of a design’s routing flow to help guide the initial iteration: die area, total number of pins, pin density, number of macros, instance density, number of nets, average pins per net, net density, and number of routing layers. These characteristics are design-dependent and do not change during detailed routing.

One challenging aspect of data generation is generating useful sequences to obtain a dataset with enough sequences that quickly converge to zero DRVs. We employed a balanced approach between exploitation and exploration. For roughly 60% of the dataset, we utilize perturbation-based sampling [8], beginning with weights near the baseline values established by domain experts and progressively exploring further from these values with each routing run. This process is illustrated in Fig. 1 (bottom). The remainder of the data set consisted of random sampling that leverage Sobol sequences [24], [25] to ensure efficient coverage of the solution space while minimizing redundant data points.

Using this methodology, we conducted 5,782 detailed routing runs across 17 designs to generate a training set. The set included 10 designs from the OpenROAD Design Suite [26], and 7 designs from the ISPD18 benchmarks to diversify

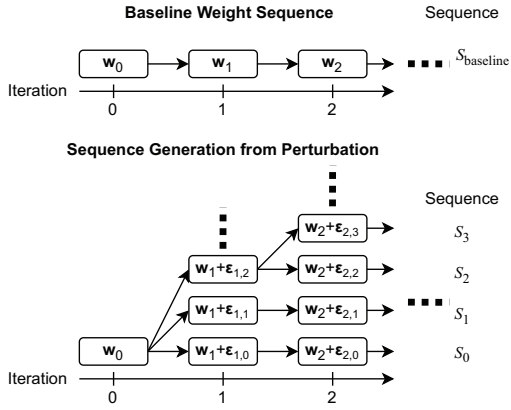


Fig. 1. Top: the baseline implementation uses a single sequence of weight vectors ( $\mathbf{w}$ ). Bottom: we use perturbation sampling to create many new sequences per design. Sequences start near the baseline values and then gradually diverge with more samples (increasing  $\epsilon$ ).

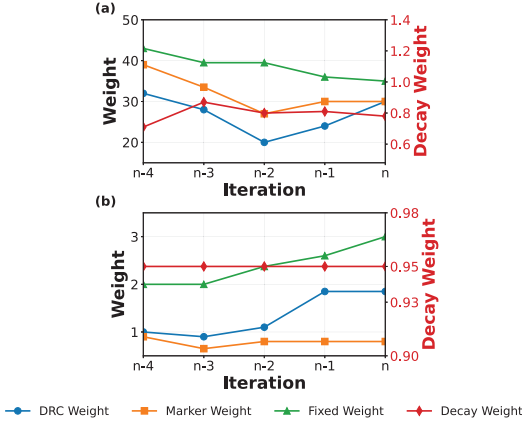


Fig. 2. a) Optimal vs b) Baseline weight trend for the last few iterations across all designs in the dataset. Optimal weight trends differ substantially from the baseline.

technology nodes and design characteristics. ISPD19 circuits were excluded from training data to be used as test data.

From this dataset, several trends become apparent. By taking the weight sequence for each design which resulted in the fastest convergence (minimum iterations), we observe how the weights differ from the baseline. Figure 2 (top) shows the average weights per iteration of these sequences. In this graph,  $n$  represents the last iteration before convergence for the given design. Therefore, the graph shows the average weight over the last 5 iterations before convergence across all designs. For designs which converge in fewer than 5 iterations, only those weights are averaged. Figure 2 (bottom) shows the corresponding weights used in the baseline. We note that the optimal weights differ substantially from the baseline weights, both in the trends and the absolute value. From these findings, we observe that there are several strong trends that can be exploited in order to improve detailed routing convergence. In the next section, we discuss our RL model architecture for inferring optimal weights.

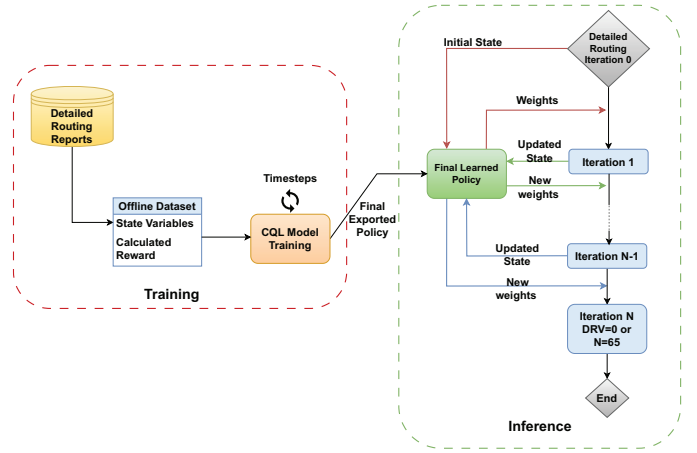


Fig. 3. Training and inference flow for conservative Q-Learning model

## IV. MODEL ARCHITECTURE

### A. Model

Our architecture leverages Conservative Q-Learning (CQL), an offline reinforcement learning algorithm that addresses Q-learning's overestimation bias through value function regularization [9]. CQL achieves this by incorporating a conservative penalty on out-of-distribution actions, effectively constraining the learned policy within the support of the training data. We found online exploration impractical in this context, as each episode can take minutes to hours to complete, which is dramatically slower than the baseline. The algorithm augments the standard Q-learning objective with a conservative term:

$$\begin{aligned} \min_Q \left( \alpha \mathbb{E}_{s \sim \mathcal{D}} \left[ \log \sum_a \exp(Q(s, a)) \right] \right. \\ \left. - \mathbb{E}_{a \sim \pi_\beta(a|s)} [Q(s, a)] \right) \\ + \frac{1}{2} \mathbb{E}_{(s, a, r, s') \sim \mathcal{D}} [(Q(s, a) - \hat{B}^\pi Q(s, a))^2] \end{aligned} \quad (1)$$

where  $\alpha$  is the conservative penalty coefficient,  $\mathcal{D}$  represents the offline dataset, and  $\pi_\beta$  represents the behavior policy [9]. A key advantage of CQL for inferring routing weights is its reduced inference time, as the algorithm's conservative regularization inherently calibrates Q-values during training and eliminates the need for additional normalization or post-processing steps during deployment [27]. Unlike conventional supervised learning approaches that require extensive datasets, CQL's architecture enables efficient learning from modest-sized datasets by leveraging complete state-action trajectories rather than isolated samples [28]. The model's ability to extract temporal dependencies and action-consequence relationships from each trajectory makes it particularly suitable for inferring routing weights as we attempt to optimize cost weights for every iteration by analyzing sequential trajectories from previous iterations. Additionally, CQL has a higher training stability due to prevention of Q-value overestimation in unexplored states while maintaining consistent performance across varying routing scenarios.

TABLE I  
CQL HYPERPARAMETER VALUES

| Hyperparameter       | Value                    |
|----------------------|--------------------------|
| Actor Learning Rate  | $1.0 \times 10^{-3}$     |
| Critic Learning Rate | $4.0 \times 10^{-3}$     |
| Conservative Weight  | $8.0 \times 10^{-1}$     |
| Batch Size           | 128                      |
| Initial Temperature  | $1.43298 \times 10^{-1}$ |
| Temperature LR       | 0.0                      |
| Tau (Polyak $\tau$ ) | $1.97555 \times 10^{-3}$ |

### B. State Representations and Reward Functions

For the reinforcement learning model to effectively train and predict optimal weights for iteration  $n$ , a state representation from iteration  $n - 1$  should convey meaningful design characteristics and dynamic routing progress information to the model. In addition to the variables collected in Section III, the variables derived from the data that were included in the state representation are:

- Average density of DRVs per routing region
- Rate of DRV reduction across iterations

Feature vectors were normalized using standard scaler from training data statistics. The reward function was clipped to 1st-99th percentiles of training data and normalized using tanh before training to help maintain stability with higher learning rates during training. The variables in the reward function in the order of importance are:

- DRV Improvement Bonus (highest importance)
- Iteration Penalty (highest importance)
- Convergence Bonus (highest importance)
- Stuck Penalty (low importance)

We deliberately exclude fine-grained spatial coordinates of violations to keep the state representation simple and generalizable. Our proposal aims at balancing model complexity while establishing a novel framework for applying reinforcement learning to a highly capable router.

### C. Model Tuning

Our model was tuned to provide faster convergence in good weight configurations without succumbing to overestimation or out-of-distribution shifts. Using the hyperparameters suggested by CQL [9] and optimizing with d3rlpy [29], we arrived at the values shown in Table I. To prevent policy collapse during training, we implement two early stopping mechanisms monitored every epoch:

**Critic Loss Explosion Detection:** The training pipeline monitors critic loss for exponential growth, which indicates Q-function approximation failure. When the critic network fails to track the actor’s rapidly changing policy updates, cascading instability occurs where the actor receives increasingly unreliable gradient signals. We implement a threshold of 100x increase to accommodate the transient overshooting behavior that can occur during early training when actor and critic networks bootstrap off each other (critic’s Bellman targets) before stabilizing for the remainder of the training.

**Action Difference Monitoring:** The d3rlpy framework’s action\_diff metric computes  $\text{action\_diff} = \frac{1}{N} \sum_{i=1}^N \|\pi_{\theta}(s_i) -$

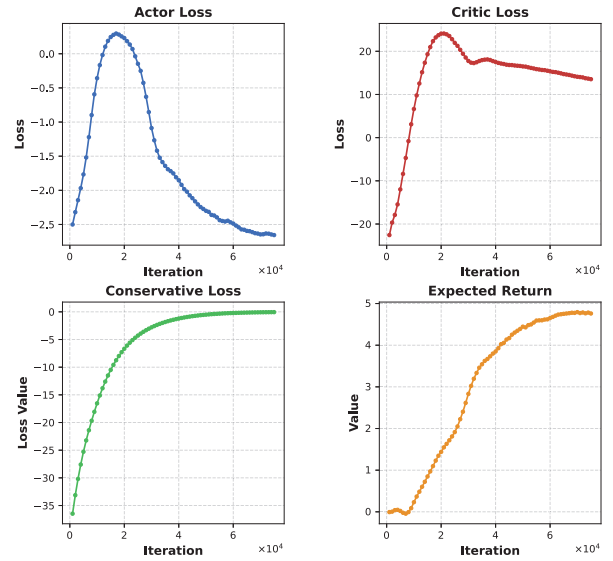


Fig. 4. CQL Training Progress

$a_i\|^2$ , where  $\pi_{\theta}(s_i)$  represents the learned policy’s action and  $a_i$  represents the action from the dataset for state  $s_i$ . While some deviation is expected for policy improvement, excessive divergence ( $>1.0$ ) indicates out-of-distribution behavior where generalization fails. Since actions are normalized to  $[0,1]$  during training, action\_diff is particularly sensitive and represents a substantial deviation in the original action space.

## V. EXPERIMENTAL RESULTS

### A. Methodology

For our experimental evaluation, we implement a comprehensive pipeline to train, deploy, and validate our reinforcement learning approach. For deployment, we developed a real-time inference pipeline using TorchScript and LibTorch to integrate the saved feature scaler and trained policy directly into OpenROAD’s C++ detailed router. This implementation extracts the relevant state variables at the end of each routing iteration and feeds them into the trained RL policy to predict optimal weight configurations for the subsequent iteration. To evaluate performance, we conducted comparative experiments between the baseline (default) weight configurations and RL-predicted weights across 29 standard and benchmark circuits, with 10 being unseen test benchmarks from ISPD19. Our benchmarks were run on an AMD EPYC 9275F CPU @ 4.1 GHz with 768 GB of DDR5 RAM. All training was performed on this platform, as the data collection time far outweighed the model training time. The runtime for each benchmark was averaged over 10 runs for each configuration (default and RL-guided). Runtimes for our approach include policy inference overhead, which was measured to be about 2s each run.

### B. Model Training

Fig. 4 shows four standard training metrics from the CQL model. In our RL model, both the critic and actor curves overshoot before settling to lower values. This initial spike validates our high threshold for early stopping, allowing

model to recover from transient instabilities. Early on, the Q-network chases shifting targets and the policy chases the evolving Q-estimates. With continued training, both losses steadily descend due to the convergence of both the actor and critic networks. Similarly, the conservative penalty starts very negative to strongly discourage out-of-distribution actions, but gradually rises to zero as the policy learns align with the training data [9]. The Expected Return plot tracks the agent’s estimate of the cumulative reward from the initial Q (first routing state), thus the smooth monotonic rise confirms that the learned policy is maximizing the expected return [30]. We monitor the Initial State Value Estimation metric to determine convergence and allow the training to continue until this metric plateaus with variance  $<0.01$  over 3 epochs, indicating stable value function approximation. The training duration shown in Fig. 4 corresponds to the point where the expected return was observed to consistently approach convergence across multiple training runs without triggering the early stopping criteria.

### C. Benchmark Performance

Our experimental results are summarized in Tables II and III. Table II shows the results across 1) the baseline OpenROAD detailed router, 2) our work using the RL model, and 3) the fastest (min.) converging model seen in the generated data. Each of the designs in this set are designs which the model has trained on, and therefore shows how well the model is able to perform on previously seen data. Table II shows the results across 1) the baseline router and 2) our RL model. Because the training data included ISPD18 benchmarks, we report here only the ISPD19 benchmarks to represent completely unseen designs (including a new technology node). Across both seen and unseen data, our model is successful in reducing the number of iterations. Our model finishes in the same or fewer iterations as the baseline. For runtime, our model is successful in decreasing the average runtime across both sets of data as well. It should be noted that in several cases, the number of iterations does not decrease, but the runtime does. This typically occurs when the policy reduces DRV’s earlier in the trajectory, which lowers the amount of work required in later iterations. Figure 5 shows DRV traces for our model and the baseline and illustrates this effect.

The policy impact is also benchmark dependent. Speedups are most pronounced on larger or more congested designs where each iteration is expensive and early DRV reduction compounds over time. In contrast, several smaller designs such as `gcd` and `ibex` already converge quickly under the baseline. In these cases, there is limited routing work to reduce, so the inference overhead per iteration can dominate the total runtime which yields negligible gain or a slight slowdown. We include the inference overhead in all reported runtimes.

This effect is visible on `ispd19_test9`, where runtime increases despite similar convergence behavior. Upon inspection, the routing difficulty of this benchmark appears relatively low for the baseline and the remaining runtime is dominated by processing a large number of simple nets rather than resolving difficult violations. Since the policy does not substantially

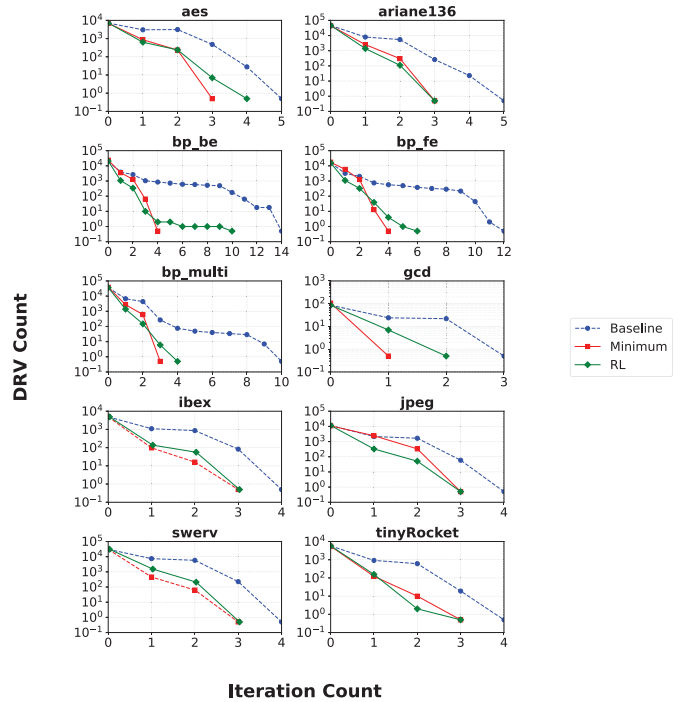


Fig. 5. DRV convergence comparison for the OpenROAD Design Suite

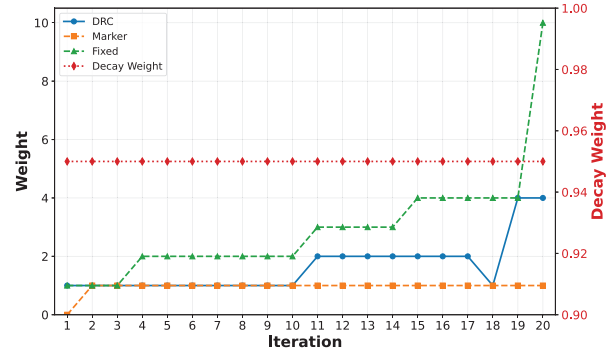


Fig. 6. Baseline weight configuration for all designs. Truncated after iteration 20 for brevity. Designs will stop early if number of DRV’s goes to 0.

change the routing effort in this regime, the constant inference overhead can outweigh the small savings. On the other hand, `ispd19_test10` exhibits higher congestion and benefits substantially from our method. The terminal DRV count is also reduced in this case, although both our model and the baseline reach the maximum iteration cap and finish with non-zero DRV’s. For wirelength, on the seen data there is a small penalty of  $<1\%$  on average and therefore the impact is minimal. We did not include wirelength in the reward function, and therefore it was not considered as an optimization target when selecting the weights.

### D. Weight Analysis

Fig. 6 shows the full schedule of weights for the baseline implementation up until iteration 20 (truncated for brevity), and Fig. 7 shows the weights for both the ones inferred at runtime by our RL model and the weights for the fastest converging sequence in the training data. The weight trajectories in Fig. 6 indicate that our model learns generalized strategies by exploring its own path and not simply replicating

TABLE II  
PERFORMANCE ON OPENROAD DESIGN SUITE (SEEN DATA)

| Design       | Iterations |      |      | Runtime (s) |      |         | DRVs |      | Wirelength (um) |          |        |
|--------------|------------|------|------|-------------|------|---------|------|------|-----------------|----------|--------|
|              | Base       | Ours | Min. | Base        | Ours | Diff    | Base | Ours | Base            | Ours     | Diff   |
| aes          | 6          | 5    | 4    | 55          | 46   | 16.36%  | 0    | 0    | 309750          | 312744   | -0.97% |
| arianel36    | 6          | 4    | 4    | 147         | 143  | 2.72%   | 0    | 0    | 8017226         | 8038308  | -0.26% |
| bp_be        | 15         | 11   | 5    | 115         | 56   | 51.30%  | 0    | 0    | 3028121         | 3037392  | -0.31% |
| bp_fe        | 13         | 7    | 5    | 80          | 43   | 46.25%  | 0    | 0    | 2352172         | 2359893  | -0.33% |
| bp_multi     | 11         | 5    | 4    | 132         | 110  | 16.67%  | 0    | 0    | 4711286         | 4729301  | -0.38% |
| gcd          | 4          | 3    | 2    | 3           | 4    | -33.33% | 0    | 0    | 4786            | 4827     | -0.86% |
| ibex         | 5          | 4    | 4    | 27          | 28   | -3.70%  | 0    | 0    | 330625          | 332567   | -0.59% |
| jpeg         | 5          | 4    | 4    | 39          | 37   | 5.13%   | 0    | 0    | 1163854         | 1169491  | -0.48% |
| swerv        | 5          | 4    | 4    | 94          | 96   | -2.13%  | 0    | 0    | 2959953         | 2973861  | -0.47% |
| tinyRocket   | 5          | 4    | 4    | 21          | 20   | 4.76%   | 0    | 0    | 723782          | 726170   | -0.33% |
| <b>Total</b> | –          | –    | –    | 713         | 583  | 18.23%  | 0    | 0    | 23601555        | 23684554 | -0.35% |

TABLE III  
PERFORMANCE ON ISPD19 (UNSEEN TEST DATA)

| Design        | Iterations |      |      | Runtime (s) |        |      | DRVs |           | Wirelength (μm) |        |      |
|---------------|------------|------|------|-------------|--------|------|------|-----------|-----------------|--------|------|
|               | Base       | Ours | Diff | Base        | Ours   | Diff | Base | Ours      | Base            | Ours   | Diff |
| ispd19_test1  | 5          | 5    | 43   | 46          | -6.98% | 0    | 0    | 63293     | 63694           | -0.63% |      |
| ispd19_test2  | 7          | 7    | 255  | 155         | 39.22% | 0    | 0    | 2480245   | 2473440         | 0.27%  |      |
| ispd19_test3  | 6          | 6    | 106  | 104         | 1.89%  | 0    | 0    | 82647     | 82683           | -0.04% |      |
| ispd19_test4  | 16         | 11   | 1336 | 443         | 66.84% | 0    | 0    | 6000839   | 6013707         | -0.21% |      |
| ispd19_test6  | 6          | 5    | 289  | 262         | 9.34%  | 0    | 0    | 6536852   | 6542048         | -0.08% |      |
| ispd19_test7  | 5          | 5    | 315  | 317         | -0.63% | 0    | 0    | 12181108  | 12189586        | -0.07% |      |
| ispd19_test8  | 5          | 5    | 458  | 455         | 0.66%  | 0    | 0    | 18733926  | 18733268        | 0.00%  |      |
| ispd19_test9  | 5          | 5    | 758  | 772         | -1.85% | 0    | 0    | 28347604  | 28354901        | -0.03% |      |
| ispd19_test10 | 65         | 65   | 4538 | 2736        | 39.71% | 24   | 5    | 28032422  | 28052084        | -0.07% |      |
| <b>Total</b>  | –          | –    | 8098 | 5200        | 35.79% | 24   | 5    | 102458936 | 102504963       | -0.04% |      |

the optima from the training data, thus balancing the bias-variance tradeoff for robustness on unseen designs. This also suggests there may be more than one weight trajectory leading to identical results since cost functions work in relativity.

Comparing the weight trajectories as a whole, it is clear that the weights required to minimize the iteration count can vary largely based on the design. Our model was able to identify features that enabled it to dynamically select better weights than the baseline version.

## VI. CONCLUSION

This paper presented a detailed router solution which leverages reinforcement learning to enable rapid convergence in detailed routing. By training a conservative Q-learning model to infer A\* search weights and minimize iterations, we were able to reduce the iterations by 5% on average and up to 31% on unseen designs in the ISPD19 benchmark. This translates to a runtime speedup of  $1.56\times$  on average and up to  $3.01\times$ . We found that this learning generalizes to other technologies, as ISPD19 has technology nodes that the model has not seen before, yet it still demonstrated significant improvement.

## ACKNOWLEDGMENTS

We thank Eugene Vinitzky for helpful early conversations and Saik Anam Siam for early guidance on practical machine learning directions for detailed routing. This work was partially funded by a Google charitable gift.

## REFERENCES

- [1] C. Y. Lee, "An Algorithm for Path Connections and Its Applications," *IRE Transactions on Electronic Computers*, vol. EC-10, no. 3, pp. 346–365, 1961.

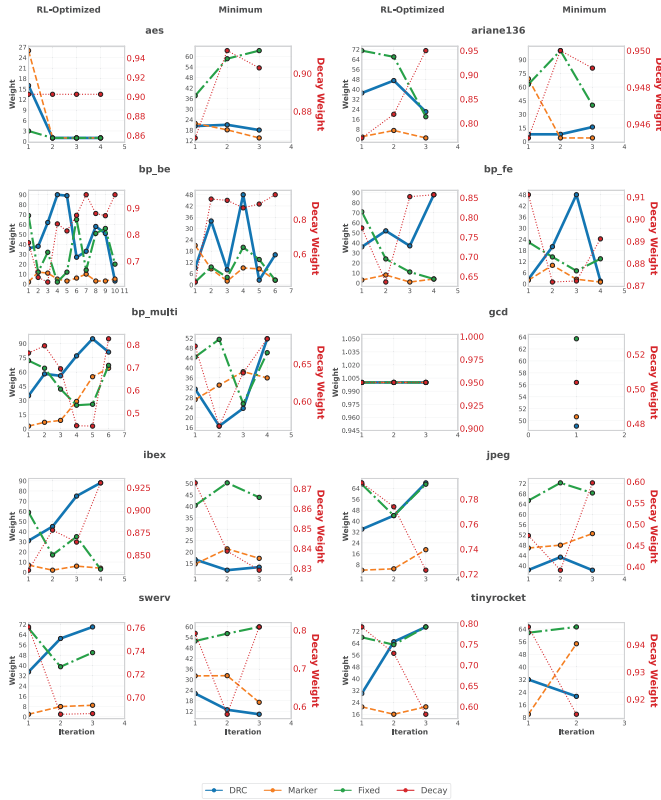


Fig. 7. Weight selections by our RL model vs. weights used in the fastest converging sequence in the training data.

- [2] M. H. Arnold and W. S. Scott, "An Interactive Maze Router with Hints," in *Proceedings of the 25th ACM/IEEE Design Automation Conference*, ser. DAC '88. Washington, DC, USA: IEEE Computer Society Press, 1988, p. 672–676.
- [3] D. W. Hightower, "A Solution to Line-Routing Problems on the Continuous Plane," in *Proceedings of the 6th Annual Design Automation Conference*, ser. DAC '69. New York, NY, USA: Association for Computing Machinery, 1969, p. 1–24. [Online]. Available: <https://doi.org/10.1145/800260.809014>
- [4] G. Chen, C.-W. Pui, H. Li, J. Chen, B. Jiang, and E. F. Y. Young, "Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search," in *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, ser. ASPDAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 754–760. [Online]. Available: <https://doi.org/10.1145/3287624.3287678>
- [5] T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, S. Dobre, R. G. Dreslinski, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Li, Z. Liang, U. Mallappa, P. Penzes, G. Pradipta, S. Reda, A. Rovinski, K. Samadi, S. S. Sapatnekar, L. Saul, C. Sechen, V. Srinivas, W. Swartz, D. Sylvester, D. Urquhart, L. Wang, M. Woo, and B. Xu, "OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain," in *Proc. GOMATECH*, 2019.
- [6] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. 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, "INVITED: Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project," in *Proc. DAC*, 2019.
- [7] C. Shi, W. Xiong, C. Shen, and J. Yang, "Provably Efficient Offline Reinforcement Learning with Perturbed Data Sources," in *International Conference on Machine Learning*. PMLR, 2023, pp. 31 353–31 388.
- [8] Q. Liu, Y. Kuang, and J. Wang, "Robust Deep Reinforcement Learning with Adaptive Adversarial Perturbations in Action Space," in *2024 International Joint Conference on Neural Networks (IJCNN)*, 2024, pp. 1–8.
- [9] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative Q-Learning for Offline Reinforcement Learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1179–1191.
- [10] H. Kaindl and G. Kainz, "Bidirectional Heuristic Search Reconsidered," *J. Artif. Int. Res.*, vol. 7, no. 1, p. 283–317, Dec. 1997.
- [11] K. Han, A. B. Kahng, and H. Lee, "Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-Based Detailed Router," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: Association for Computing Machinery, 2015.
- [12] A. B. Kahng, L. Wang, and B. Xu, "TritonRoute: The Open-Source Detailed Router," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 3, pp. 547–559, 2020.
- [13] T. Nieberg, "Gridless pin access in detailed routing," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 170–175.
- [14] Y. Ding, C. Chu, and W.-K. Mak, "Self-Aligned Double Patterning Lithography Aware Detailed Routing With Color Preassignment," vol. 36, no. 8, p. 1381–1394, Aug. 2017.
- [15] I.-J. Liu, S.-Y. Fang, and Y.-W. Chang, "Overlay-Aware Detailed Routing for Self-Aligned Double Patterning Lithography Using the Cut Process," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2014, pp. 1–6.
- [16] M. Ahrens, M. Gester, N. Klewinghaus, D. Muller, S. Peyer, C. Schulte, and G. Tellez, "Detailed Routing Algorithms for Advanced Technology Nodes," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 34, no. 4, p. 563–576, Apr. 2015.
- [17] S. Mantik, G. Posser, W.-K. Chow, Y. Ding, and W.-H. Liu, "ISPD 2018 Initial Detailed Routing Contest and Benchmarks," in *Proceedings of the 2018 International Symposium on Physical Design*, ser. ISPD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 140–143. [Online]. Available: <https://doi.org/10.1145/3177540.3177562>
- [18] W.-H. Liu, S. Mantik, W.-K. Chow, Y. Ding, A. Farshidi, and G. Posser, "ISPD 2019 Initial Detailed Routing Contest and Benchmark with Advanced Routing Rules," in *Proceedings of the 2019 International Symposium on Physical Design*, ser. ISPD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 147–151. [Online]. Available: <https://doi.org/10.1145/3299902.3311067>
- [19] A. B. Kahng, L. Wang, and B. Xu, "TritonRoute: An Initial Detailed Router for Advanced VLSI Technologies," in *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2018, pp. 1–8.
- [20] W. Zeng, A. Davoodi, and R. O. Topaloglu, "Explainable DRC Hotspot Prediction with Random Forest and SHAP Tree Explainer," in *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2020, pp. 1151–1156.
- [21] H. Park, K. Baek, S. Kim, K. Choi, and T. Kim, "Pin Accessibility and Routing Congestion Aware DRC Hotspot Prediction for Designs in Advanced Technology Nodes With Consolidated Practical Applicability and Sustainability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 12, pp. 4786–4799, 2024.
- [22] 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," in *Proceedings of the 2020 International Symposium on Physical Design*. New York, NY, USA: Association for Computing Machinery, 2020, p. 135–142.
- [23] H. Chen, K.-C. Hsu, W. J. Turner, P.-H. Wei, K. Zhu, D. Z. Pan, and H. Ren, "Reinforcement Learning Guided Detailed Routing for Custom Circuits," in *Proceedings of the 2023 International Symposium on Physical Design*, ser. ISPD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 26–34.
- [24] S. Mishra and T. K. Rusch, "Enhancing Accuracy of Deep Learning Algorithms by Training with Low-Discrepancy Sequences," *SIAM Journal on Numerical Analysis*, vol. 59, no. 3, pp. 1811–1834, 2021.
- [25] S. Burhenne, D. Jacob, and G. P. Henze, "Sampling based on sobol' sequences for monte carlo techniques applied to building simulations," in *Proceedings of Building Simulation 2011: 12th Conference of IBPSA*, ser. Building Simulation, vol. 12. Sydney, Australia: IBPSA, November 2011, pp. 1816–1823. [Online]. Available: [https://publications.ibpsa.org/conference/paper/?id=bs2011\\_1590](https://publications.ibpsa.org/conference/paper/?id=bs2011_1590)
- [26] A. Rovinski, T. Ajayi, M. Kim, G. Wang, and M. Saligane, "Bridging Academic Open-Source EDA to Real-World Usability," in *Proc. ICCAD*, November 2020, pp. 1–7.
- [27] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems," 2020.
- [28] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 1054–1054, 1998.
- [29] T. Seno and M. Imai, "d3rlpy: An Offline Deep Reinforcement Learning Library," 2022. [Online]. Available: <https://arxiv.org/abs/2111.03788>
- [30] T. L. Paine, C. Paduraru, A. Michi, C. Gulchere, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas, "Hyperparameter Selection for Offline Reinforcement Learning," 2020. [Online]. Available: <https://arxiv.org/abs/2007.09055>