Optimizing Mixed-Signal Routing Using Deep Reinforcement Learning

Avikam Chauhan*
avikam.chauhan@berkeley.edu
University of California, Berkeley
Berkeley, USA

Ashwin Rammohan*
ashwin1501@berkeley.edu
University of California, Berkeley
Berkeley, USA

Abstract

Modern systems-on-a-chip (SoCs) typically contain several important mixed-signal circuits, such as analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and phase-locked-loops (PLLs), that rely on close integration of analog and digital circuits. Noise coupling of digital signals to analog signals can significantly degrade the performance of these circuits, and thus the generation of optimal signal routes in layout is an important step in mixed-signal design automation.

We present a reinforcement learning (RL) based framework to generate optimized analog routes that minimize parasitic noise coupling to nearby digital signals, and evaluate its performance on a variety of circuit topologies, from simple 2D and 3D examples to a modified layout of a commonly used circuit — the StrongARM comparator.

Keywords: analog mixed-signal (AMS) circuits, circuit routing, electronic design automation (EDA), computer-aided design (CAD), machine learning (ML), deep reinforcement learning (RL)

1 Introduction

Mixed-signal chip blocks, such as analog-to-digital converters (ADCs), digital-to-analog converters (DACs), and phase-locked-loops (PLLs) rely on close integration of analog and digital circuits. However, as technology nodes shrink and supply voltages are reduced, noise-coupling between analog and digital circuits becomes a critical challenge to solve for mixed-signal SoCs. Digital signal routes swing from rail to rail at high frequencies, and these large signal swings can interfere (through parasitic capacitive coupling) with sensitive analog routes carrying much smaller signal swings. Noise coupling can degrade the signal-to-noise-and-distortion ratio (SNDR) of mixed-signal circuits, forcing them to burn more power or accept poorer performance. Traditionally, noise coupling is solved using physical isolation techniques (such as guard rings and shielding wires) between analog and digital blocks. However, these techniques consume significant area and moreover, shielding cannot practically be applied to every route for critical analog signals.

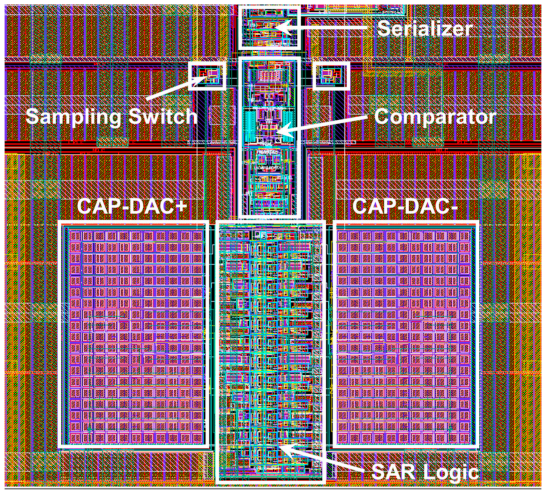

Figure 1. A layout of a 12-bit SAR ADC. Key blocks are labeled. [4]

Figure 1 shows an example of a mixed-signal circuit, a 12-bit successive-approximation-register (SAR) analog-to-digital converter (ADC) that is responsible for quantizing analog signals into 12-bit digital values. The ADC consists of blocks like the comparator and the capacitive digital-to-analog converter (CAP-DAC) that are sensitive to noise as well as blocks like the SAR logic that have hard-switching rail-to-rail digital signals. Isolating the sensitive blocks from the noise generated by the digital circuits is crucial to maintaining the high precision of the ADC.

We aim to develop an automated, reinforcement learning based tool to generate optimized routes that minimize noise coupling to nearby digital signal wires.

1.1 The Motivation for Machine Learning

Manual mixed-signal layout is a repetitive and iterative process that may not lead to an optimal result. Humans are not able to predict or visualize the effects of electromagnetic

*Both authors contributed equally to this research.

interactions between various wires and blocks, so they rely on an iterative cycle of using EDA tools to evaluate their layouts and then modify their routing as needed to improve performance.

We believe machine learning is an apt solution for such a problem because the design space for routing grows exponentially with the size of the circuit. We can train machine learning models to quickly iterate over different routing schemes and learn to place optimized routes that reduce noise coupling and other undesired effects. We propose that ML models can learn an optimal result by minimizing a linear combination of various cost parameters (e.g. parasitic wire capacitances, total wire length, total area, etc.), where the weights for each component could be user-defined.

1.2 The Motivation for Reinforcement Learning

Reinforcement learning is especially useful in these types of problems because there is a lack of sufficient datasets to train large-scale ML models (like neural networks, among others). The advantage of reinforcement learning is that we only need to define a few parameters: a state representation, state dynamics (how the agent interacts with the environment), and a reward function. Once these are defined, we can generate many arbitrary examples and run the RL agent on these cases, and train it to perform actions that optimize the reward.

We propose the concept of using reinforcement learning with human feedback (RLHF) in this application, to further solve the issue of limited training data. With RLHF, the reward function is a combination of the traditional RL reward function, along with a human-provided feedback (a score that rates the quality of the agent's output). This means that we can effectively learn better reward functions than just simply designing our own function manually, and we can have experts rank the agent's outputs for the same problem (but using different random seeds for the initial placement). As a result, the agent could learn to imitate behaviors that are more intuitive to humans (like clean layouts, proper alignment, etc.) which cannot be accurately described in a traditional reward function.

2 Related Work

Much of the prior work in mixed-signal design automation has focused on design and layout automation of the actual analog circuits, which is a massively time-consuming and iterative process for human designers. Despite this being an important problem to solve, the time and effort taken to perfect analog design automation will have gone to waste if the top-level layout and routing of digital and analog blocks is done poorly.

However, there has been some prior work on optimizing analog routing. Zhu, et al. [6] at UT Austin developed a nonlinear-programming-based global placement algorithm followed up by a wirelength-based compaction. Chen, at al. at NVIDIA also developed a reinforcement learning based routing scheme for analog/mixed-signal circuits. However, their routing scheme is optimized specifically for symmetry and matching of paths which ensures matched loads and parasitics in layout across pairs of signals [1]. Within the topic of optimizing noise-coupling, Lin et al. from National Chung Cheng University in Taiwan have developed a **deterministic** scheme to generate optimized mixed-signal layouts for minimal noise coupling [2]. Their approach uses slicing trees to organize analog and digital signals optimally on the chip.

We aim to synthesize the ideas in these approaches by leveraging reinforcement learning to place analog routes that optimize for noise coupling specifically.

To our knowledge, this work is the first of its kind, with the goal of using deep reinforcement learning to optimize analog mixed-signal routing with respect to capacitive noise coupling.

3 Methodology

In this section, we detail the specific implementation we designed and developed to utilize reinforcement learning to solve the mixed-signal routing problem. We describe the problem formulation (including the state space representation, state transition dynamics, and the reward function), as well as the model architecture and training process.

3.1 Deep Q Network (DQN) Algorithm

We used the Deep Q Network (DQN) algorithm for our reinforcement learning agent. DQNs are designed to learn the Q-value function for state-action pairs: $Q(s, a)$. Then, this learned function is used by the agent to select the actions that maximizes the Q-value at each timestep t: $a_t = \text{argmax}_a\, Q(s_t, a)$.

3.2 Markov Decision Process Formulation

We have formulated the problem as a Markov Decision Process (MDP), which is the standard format for problems being solved with reinforcement learning algorithms. An MDP is discrete-time stochastic control process, where an agent can interact with its environment by sampling state conditions, performing actions, and receiving reward signals at each timestep.

3.2.1 State Space.

The state space we chose is as follows:

- Current Grid: a 3D grid of shape $L \times W \times H$ which encodes out-of-bounds regions (consisting of the circuit block placements and previously routed analog wires) with ones at the corresponding coordinates, and zeros everywhere else

- Wire Grid: a 3D grid of shape $L \times W \times H$ which encodes digital "aggressor" wires (also considered out-of-bounds regions) with ones at the corresponding coordinates, and zeros everywhere else
- Current Position Grid: a 3D grid of shape $L \times W \times H$ which encodes the agent's current position in the layout with a one, and zeros everywhere else
- Target Position Grid: a 3D grid of shape $L \times W \times H$ which encodes the target position for the current route in the layout with a one, and zeros everywhere else

3.2.2 Action Space.

In this 3D space, the agent can take the following actions:

- Move 1 step forward
- Move 1 step backward
- Move 1 step left
- Move 1 step right
- Move 1 step up (via to the next metal layer)
- Move 1 step down (via to the previous metal layer)

3.2.3 Reward Function.

At each timestep, the agent is provided a reward for the action it takes. The reward function is defined as follows:

- -1 penalty for each timestep: this incentivizes the agent to prioritize shorter paths and complete the routes in fewer timesteps
- +8 reward for every unit decrease in the Manhattan distance to the current target: this incentivizes the agent to move towards the target position, on average
- -20 penalty for taking a via: this disincentivizes the agent from taking vias, unless they are absolutely necessary or reduce coupling capacitance significantly
- $-5 \times C_{\text{int}}$ penalty for intermediate coupling capacitance: this disincentivizes routes that are close and parallel to existing wires, and is calculated at each timestep — only if intermediate coupling capacitance calculations are enabled
- +75 reward every time the agent reaches a target position: this incentivizes the agent to finish routes
- +200 reward when the agent successfully completes all routes: this provides the agent a substantial reward if all the routes have been placed
- $-5 \times C_{\text{total}}$ penalty for total coupling capacitance: this disincentivizes routes that are close and parallel to existing wires, but is only calculated once at the end of each rollout — only if intermediate coupling capacitance calculations are disabled
- -100 penalty if the agent goes out of bounds, intersects with other wires/blocks, or takes more than 200 timesteps to complete routing: these are all failed rollouts, and we penalize the agent accordingly

We chose these reward values through manual hyperparameter tuning experiments, where we evaluated the tradeoff between different goals (e.g. minimizing routing distance and minimizing coupling capacitance). Ultimately, we chose values that allowed the agent to successfully reach its targets successfully while still weighing coupling capacitance, via penalties, etc.

3.3 Coupling Capacitance Calculations

We implemented 2 different methods to calculate the coupling capacitance: one that can be used to calculate the intermediate coupling capacitance at each timestep (done individually on the route that is currently being placed), and one that can be used to calculate the total coupling capacitance after the rollout is complete (done once all routes are successfuly placed).

Ultimately, we chose to use the intermediate coupling capacitance calculation method (as the agent produces better routes when trained with this reward function), but we demonstrate the difference in performance when using the total coupling capacitance calculation method on a test circuit in the next section.

3.3.1 Intermediate Coupling Capacitance Calculation.

The intermediate coupling capacitance is calculated by comparing the previous (x, y, z) coordinate with the current (x, y, z) coordinate. After determining if the segment lies in the x (or y) axis, we then loop through the row (or column) corresponding to the current coordinate, and add up the cells with ones (i.e. the ones that are occupied by wires), weighted inversely by their distance. For the sake of simplicity, we ignore the coupling capacitance between segments in the z axis, since the RL agent rarely takes vias (if at all), so the effects of coupling in this axis are relatively insignificant.

More formally, the coupling capacitance calculation for horizontal segments is:

$$C_{\text{int\_x}} = \sum_{x \in [0,L],\ x \neq \text{curr\_x}} \frac{\text{wire\_grid}[x, \text{curr\_y}, \text{curr\_z}]}{|x - \text{curr\_x}|}$$

and the calculation for vertical segments is:

$$C_{\text{int\_y}} = \sum_{y \in [0,W],\ y \neq \text{curr\_y}} \frac{\text{wire\_grid}[\text{curr\_x}, y, \text{curr\_z}]}{|y - \text{curr\_y}|}$$

These penalties are accumulated by the agent over the course of each rollout.

3.3.2 Total Coupling Capacitance Calculation.

The total coupling capacitance is calculated with a clever dot-product technique that determines the overlap between wires within the same metal layer, and across different metal layers.

Essentially, we take the dot-products of all combinations of rows within a layer, and dot-products of all combinations of columns within a layer to determine the coupling capacitance

for wires within the same metal layer. Then, we take the dot-produts of all combinations of layers within the design, and that allows us to determine the coupling capacitance for wires on separate metal layers. These intermediate results are then combined using a weighted sum to determine the total coupling capacitance.

More formally, the coupling capacitance calculation for horizontal segments is:

$$C_{\text{total\_x}} = \sum_{x_1 \in [0,L],\ x_2 \in [0,L],\ x_1 < x_2} \frac{\langle \text{row}_{x_1}, \text{row}_{x_2} \rangle}{|x_1 - x_2|}$$

and the calculation for vertical segments is:

$$C_{\text{total\_y}} = \sum_{y_1 \in [0,W],\ y_2 \in [0,W],\ y_1 < y_2} \frac{\langle \text{col}_{y_1}, \text{col}_{y_2} \rangle}{|y_1 - y_2|}$$

and the calculation for cross-layer capacitance is:

$$C_{\text{total\_z}} = \sum_{z_1 \in [0,H],\ z_2 \in [0,H],\ z_1 < z_2} \frac{\langle \text{layer}_{z_1}, \text{layer}_{z_2} \rangle}{|z_1 - z_2|}$$

The total coupling capacitance is calculated as follows:

$$C_{\text{total}} = \alpha_1 \ (C_{\text{total\_x}} + C_{\text{total\_y}}) + \alpha_2 \ C_{\text{total\_z}}$$

where α_1 and α_2 can be chosen to reflect the scaling factor between the wire pitch on the same metal layer versus the pitch and dielectric constant between successive metal layers (as defined by the technology node specifications). These scale factors can also be used to prioritize the reduction of coupling capacitance either within layers or across different layers. For the purpose of our experiments, we applied equal weights for α_1 and α_2.

3.4 Model Architecture and Training

Our RL agent can be trained to place routes for a variety of circuits, ranging from simple to complex. As expected, the training time increases with complexity of the circuit, given the number and length of routes, and the larger physical design space for the RL agent to explore.

3.4.1 Model Architecture.

The DQN model is a neural network that is trained to predict Q-values for state-action pairs: $Q(s, a)$. We define the Q-net model architecture as a sequential network:

1. Linear layer: $(L \times W \times H \times 4) \rightarrow 64$
2. ReLU activation layer
3. Linear layer: $64 \rightarrow 64$
4. ReLU activation layer
5. Linear layer: $64 \rightarrow 6$

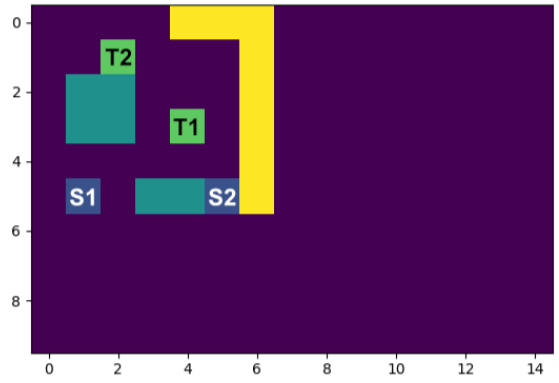

Figure 2. The starting state of the *Simple 2D Grid* circuit. Starting positions are labeled with a S and target positions are labeled with a T.

This architecture is duplicated for the Q-net target network, which prevents overfitting to local optima and reduces noise in gradients during the training process.

3.4.2 Optimizing the Model Training Process.

To improve the model performance, we performed manual hyperparameter sweeps over variables such as the total number of training timesteps, learning rate, reward discount factor, exploration factor, etc.

We also implemented a variety of tweaks to improve the model training process (both in terms of training time and overall performance). This included randomizing the order of routes (start-target pairs) after each rollout, to reduce overfitting on individual routes and instead learn to optimize all routes evenly. We also experimented with cycling through all the routes and taking one step for each route in parallel, instead of fully completing one route before moving onto the next one, but this was largely unsuccessful.

4 Experiments and Results

4.1 *Simple 2D Grid* Experiment

The first experiment we conducted to test the performance of the RL agent was a simple, single-layer circuit, consisting of 2 starts and 2 targets. The circuit had 2 pre-placed circuit blocks and 1 digital wire that should be avoided to reduce coupling capacitance. Figure 2 depicts the initial state of the *Simple 2D Grid* circuit.

After training the RL agent on this circuit, we plot the average reward earned by the agent in each episode in Figure A1. We also plot the Q-net model loss throughout the training process in Figure A2. Finally, we perform an evaluation rollout to visualize the routes drawn by the agent, depicted in Figure 3:

Although the routes drawn by the RL agent successfully connect the respective start and target pairs, the route on

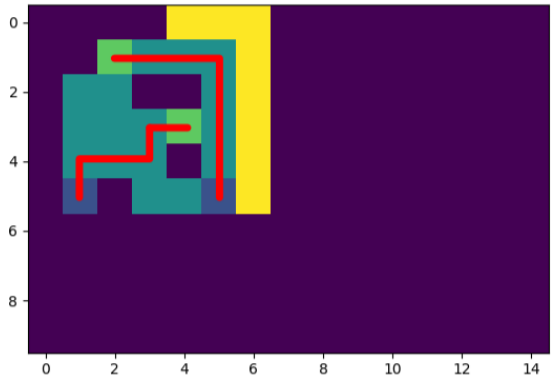

Figure 3. The final state of the *Simple 2D Grid* circuit. Routes (shown in red) are successfully drawn for all pairs of starts and targets.

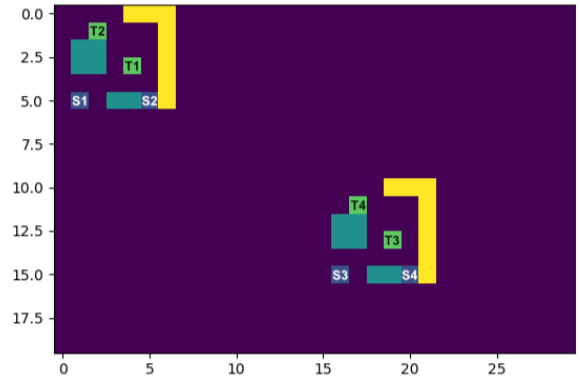

Figure 4. The starting state of the *Expanded 2D Grid* circuit. Starting positions are labeled with a S and target positions are labeled with a T.

the right is suboptimal because it is immediately adjacent to the digital wire (shown in yellow). Nevertheless, the left route is optimized because it is as far as it can possibly be from the digital wire.

4.2 *Expanded 2D Grid* Experiment

The second experiment we conducted was based off of the *Simple 2D Grid* circuit, but this time with more starts and targets. Essentially, we tiled the *Simple 2D Grid* circuit twice on a larger physical workspace to see how the RL agent would perform on a larger-scale circuit (with a bigger observation space and more freedom to draw complex routes). This is also relevant for real-world routing problems because there are often distinct circuit blocks that need many short wires to be routed between nodes, while having fewer long wires across the entire layout (which are also less likely to be analog signals by design). Figure 4 depicts the initial state of the *Expanded 2D Grid* circuit.

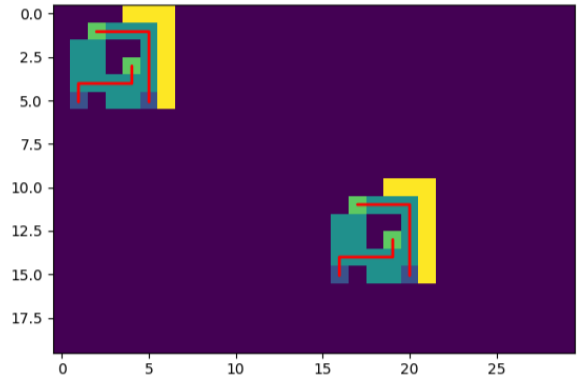

Figure 5. The final state of the *Expanded 2D Grid* circuit. Routes (shown in red) are successfully drawn for all pairs of starts and targets.

After training the RL agent on this circuit, we plot the average reward earned by the agent in each episode in Figure A3. We also plot the Q-net model loss throughout the training process in Figure A4. Finally, we perform an evaluation rollout to visualize the routes drawn by the agent, depicted in Figure 5.

Similar to the *Simple 2D Grid* experiment, the routes on the right of each "sub-circuit" are suboptimal because they are immediately adjacent to the digital wires (shown in yellow). Nevertheless, the left routes are optimized because they are as far as they can possibly be from the digital wire.

4.3 *Expanded 3D Grid* Experiment

We then extended the *Expanded 2D Grid* circuit by adding 2 new metal layers, one above and one below the existing layer from the previous design. This allows the RL agent to have more flexibility in the potential paths it can take, as it can use vias to move up and down between metal layers and potentially optimize the coupling capacitance even further.

To compare a scenario where the RL agent has the option to utilize vias (but is not required to, because the starts and targets are all in the same metal layer) with a scenario where the RL agent is required to take a via (because the starts and targets are not in the same metal layer), we create 2 versions of the *Expanded 3D Grid* circuit. This allows us to evaluate how the RL agent performs when it has this larger action space and determine whether it can optimize the locations where vias are taken in situations where it is necessary to move between metal layers.

4.3.1 Coplanar Starts and Targets.

Figure 6 depicts the initial state of the *Expanded 3D Grid* circuit with coplanar starts and targets.

After training the RL agent on this circuit, we plot the average reward earned by the agent in each episode in Figure

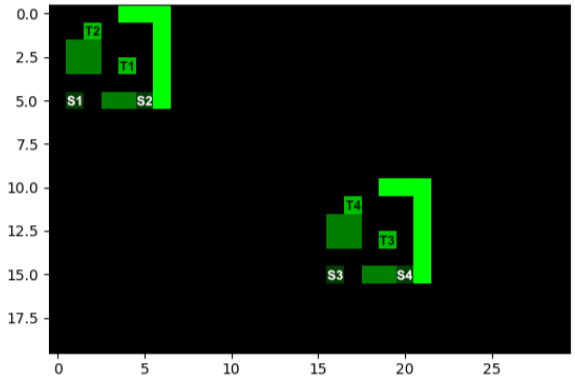

Figure 6. The starting state of the *Expanded 3D Grid* circuit with coplanar starts and targets. Starting positions are labeled with a S and target positions are labeled with a T.

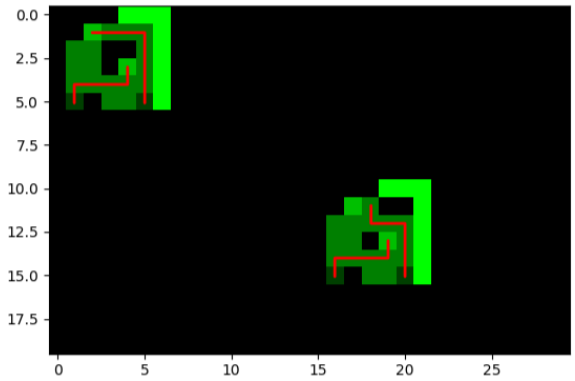

Figure 7. The final state of the *Expanded 3D Grid* circuit with coplanar starts and targets. Routes (shown in red) are successfully drawn for all pairs of starts and targets.

A5. We also plot the Q-net model loss throughout the training process in Figure A6. Finally, we perform an evaluation rollout to visualize the routes drawn by the agent, depicted in Figure 7.

The routes drawn by the RL agent successfully connect the respective start and target pairs, and most of them are optimally drawn because they generally avoid being immediately adjacent to the digital wires (shown in neon green) whenever possible, while also optimizing for the wire length of each route. The only exception to this is the wire on the right side of the left "sub-circuit", which is placed immediately adjacent to the digital wire.

4.3.2 Non-coplanar Starts and Targets.

Figure 8 depicts the initial state of the *Expanded 3D Grid* circuit with non-coplanar starts and targets.

After training the RL agent on this circuit, we plot the average reward earned by the agent in each episode in Figure

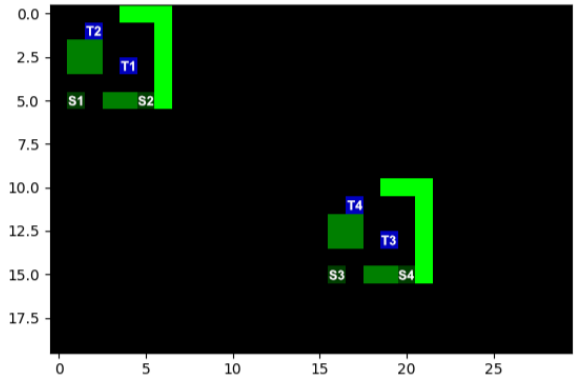

Figure 8. The starting state of the *Expanded 3D Grid* circuit with non-coplanar starts and targets. Starting positions are labeled with a S and target positions are labeled with a T.

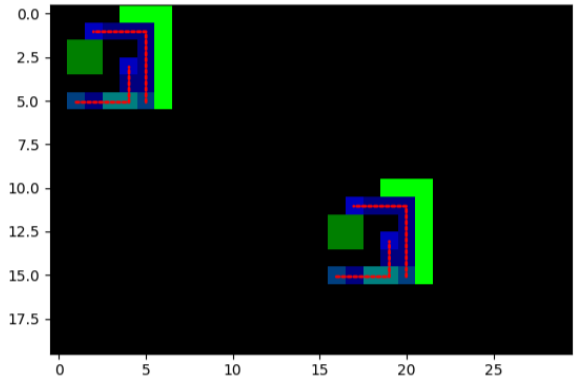

Figure 9. The final state of the *Expanded 3D Grid* circuit with non-coplanar starts and targets. Routes (shown in red) are successfully drawn for all pairs of starts and targets. Dotted lines correspond to wires on the topmost metal layer (which also contains the target positions).

A7. We also plot the Q-net model loss throughout the training process in Figure A8. Finally, we perform an evaluation rollout to visualize the routes drawn by the agent, depicted in Figure 9.

The routes drawn by the RL agent successfully connect the respective start and target pairs, and are all optimally drawn because they are far from the digital wires (shown in neon green), while also optimizing for the wire length of each route.

4.4 Evaluating Intermediate Coupling Capacitance vs. Total Coupling Capacitance Calculation

Before conducting a variety of experiments on different circuits, we compared the performance of our RL agent when calculating the intermediate coupling capacitance at each timestep versus calculating the total coupling capacitance at the end of each rollout.

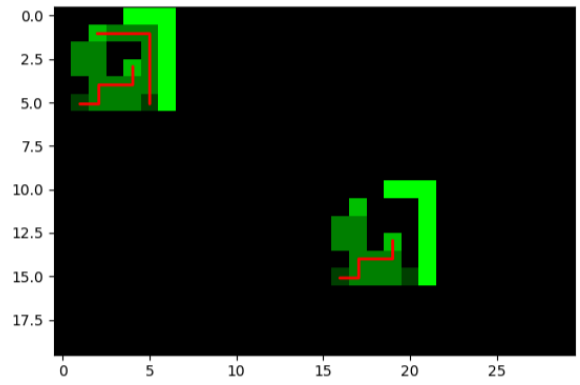

Figure 10. The routes (shown in red) placed by the RL agent when the total coupling capacitance is calculated at the end of each episode.

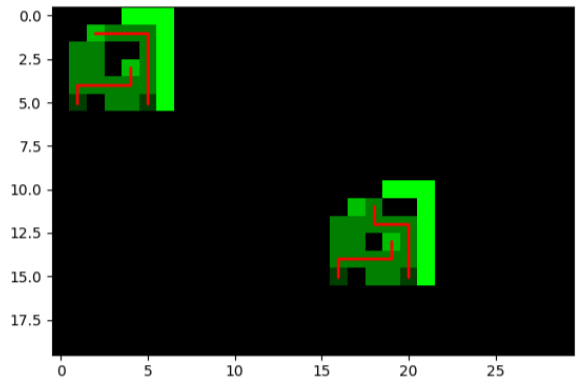

Figure 11. The routes (shown in red) placed by the RL agent when the intermediate coupling capacitance is calculated at each timestep.

In principle, we expected the intermediate coupling capacitance calculations to provide better results, since the agent is penalized for coupling capacitance at a much higher frequency, and thus should learn to better optimize for this cost. However, this method is more computationally intensive than the total coupling capacitance calculation, which is why we considered testing both methods. Although calculating the total coupling capacitance at the end violates the Markovian properties of the reward function, the episode lengths are relatively short, and thus we expected that it would not reduce the agent's final performance significantly.

We tested the different coupling capacitance calculations on the *Expanded 3D Grid* circuit with coplanar starts and targets (as described in section 4.3.1). Figure A9 depicts the average reward earned by the agent using each of the coupling capacitance calculation methods, and Figure A10 depicts the Q-net model loss using each of the coupling capacitance calculation methods.

In the average reward plot, we see the intermediate coupling capacitance calculations enable the agent to earn higher rewards, and produce better routes overall. This is further confirmed by consistently lower loss values for the respective DQN model, and by the actual routes the agent generates using each of the coupling capacitance calculation methods.

Figure 10 depicts the routes created by the RL agent when coupling capacitance is calculated just once at the end of each episode. In this scenario, the agent only successfully places 3 out of the 4 routes, and the routes do not minimize the coupling capacitance.

Figure 11 depicts the routes created by the RL agent when the intermediate coupling capacitance is calculated at each timestep. In this scenario, the agent only successfully completes all 4 routes, and most of the routes are optimized for lower coupling capacitance.

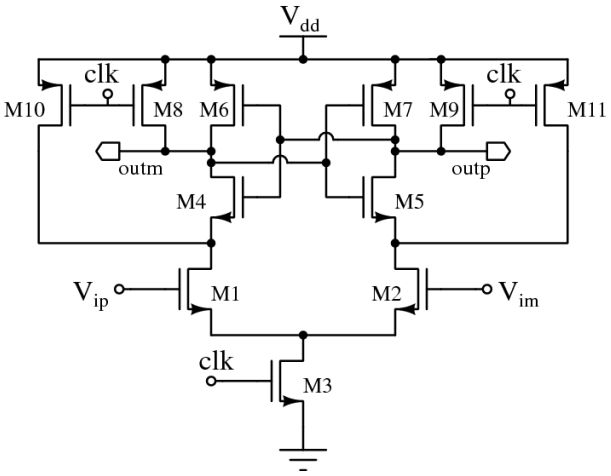

Figure 12. The StrongARM comparator schematic [3].

Clearly, given both the average reward earned by the agent, and the physical placement of wires generated by the RL agent, we conclude that calculating the intermediate coupling capacitance is more effective. **Thus, this is the method we used for all experiments described in this section.**

4.5 StrongARM Comparator Case Study

Finally, we chose to test the performance of our RL agent on a StrongARM comparator, a circuit used commonly in analog to digital converters (ADCs) to compare two input voltages and determine which is larger. Comparators are designed for good noise robustness to distinguish between close analog voltages. Therefore, minimizing noise coupling from digital signals is important for these circuits.

4.5.1 StrongARM Comparator Schematic and Layout.

The StrongARM comparator circuit schematic is shown in Figure 12.

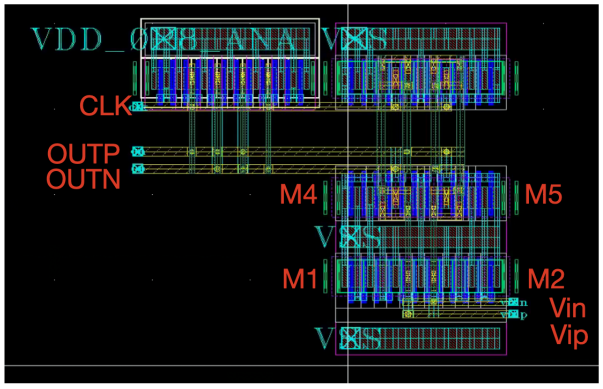

Figure 13. The StrongARM comparator layout with key signals labeled in red.

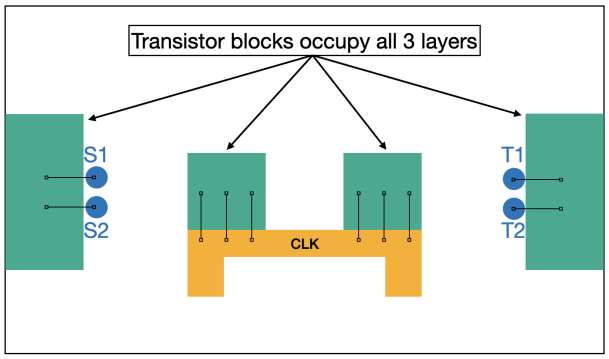

Figure 14. The *Modified StrongARM Layout* with starts and targets labeled as S1 and S2, T1 and T2.

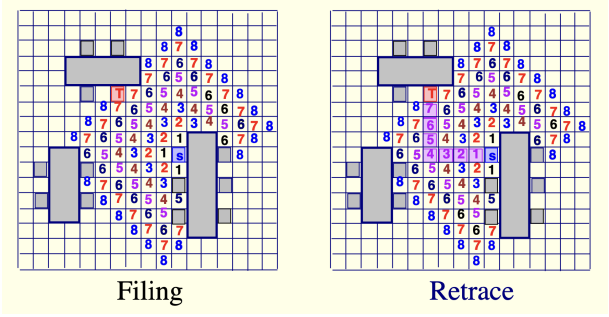

Figure 15. A visualization of the Lee Algorithm [5].

An example of a StrongARM layout (performed manually by Aviral Pandey in Professor Rikky Muller's research group at UC Berkeley for a research tapeout) is shown in Figure 13. As seen in the layout, the input signals are kept far away from the rail-to-rail switching clock signal.

We used this human-generated layout as a baseline for how a human expert would go about routing these signals. However, the routing for the output signals and the input signals here is fairly trivial.

4.5.2 *Modified StrongARM Layout* for Experiment.

To make this problem more challenging and feasible for the RL agent, we altered this layout into the *Modified StrongARM Layout* as shown in Figure 14. The grid occupies 3 metal layers in the z axis. The connections from the input and output transistors are abstracted away into starts S1 and S2 and targets T1 and T2 (depicted as blue-colored dots). The teal-colored regions represent transistors connecting to the input, output, and clock signals, and they occupy all three metal layers in the z axis. The yellow-colored region in the middle represents the U-shaped wire for the clock signal. For the purpose of this modified experiment, we added the U-shape extension to mimic what the shape would be like if the clock wire were to include vias coming down from a higher metal layer. For this circuit, the RL agent needs to successfully route from S1 to T1 and S2 to T2 while maximizing the reward function.

4.5.3 Baseline — Lee Algorithm.

To benchmark the performance of our RL-based routing agent for the *Modified StrongARM Layout*, we first examine the performance of the Lee algorithm [5], an established routing algorithm that does not account for noise coupling. The Lee algorithm operates in two stages: Filling and Retracing, as shown in Figure 15.

The start and target locations are marked on the grid in Figure 15 as **S** and **T** respectively. During the filling stage, a breadth-first-search (BFS) style "wave propagation" approach is used to find the Manhattan distances from **S** to each point on the grid until the target is reached. Then during the retracing stage, the algorithm starts at **T** and retraces the path to **S** by choosing the next closest point to the start at each step.

The Lee algorithm suffers from a large time and space complexity of $O(MN)$ for a grid of size $M \times N$. Nevertheless, we use the Lee algorithm as a baseline because it is guaranteed to find the shortest route from **S** to **T** and is completely agnostic of any coupling or interference from adjacent wires.

In our adaptation of the Lee algorithm, we extend it to 3D (allowing the algorithm to take vias and move from one metal layer to another) and also account for the "out-of-bounds" regions where it cannot place routes (because of pre-defined block placements and wires that have already been routed).

The results of running the Lee algorithm on the *Modified StrongARM Layout* are shown in Figure 16. The algorithm begins at S1, predictably routing around the transistor region in the middle to reach T1. For S2, the algorithm decides to traverse to the clock region; then it takes a via up to the next metal layer to cross the clock region before it takes a via back down to reach T2. The route on the upper metal layer is marked as a dashed red line, rather than a solid red

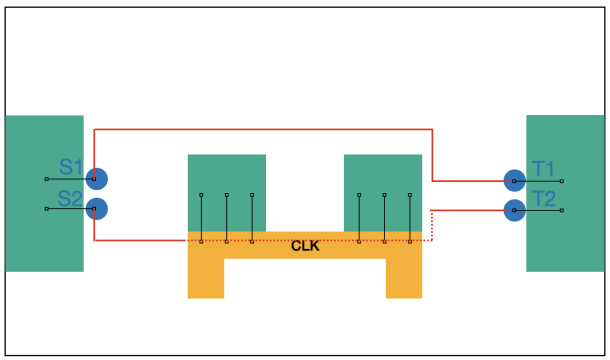

Figure 16. The routes generated by the Lee algorithm for the *Modified StrongARM Layout*.

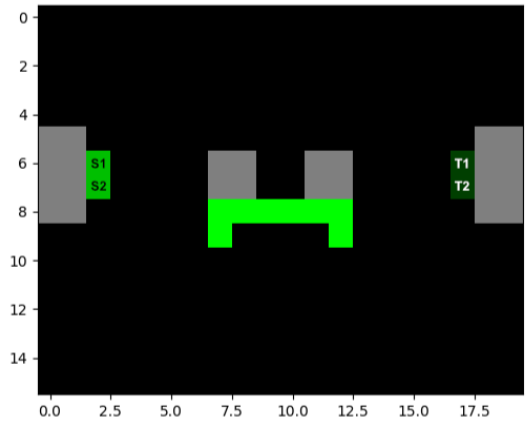

Figure 17. The starting state of the *Modified StrongARM Layout* circuit with coplanar starts and targets. Starting positions are labeled with a S and target positions are labeled with a T.

line. This S2 → T2 route is indeed the shortest route from a Manhattan distance perspective, although it clearly suffers from a large coupling capacitance when traveling directly over the clock signal route.

As a preliminary baseline, the routes generated using the Lee algorithm prove to be useful in analyzing performance of our RL agent's performance on real-world circuit designs.

4.5.4 RL Agent Performance with Coplanar Starts and Targets.

Figure 17 depicts the initial state of the *Modified StrongARM Layout* circuit with coplanar starts and targets.

After training the RL agent on this circuit, we plot the average reward earned by the agent in each episode in Figure A11. We also plot the Q-net model loss throughout the training process in Figure A12. Finally, we perform an evaluation rollout to visualize the routes drawn by the agent, depicted in Figure 18:

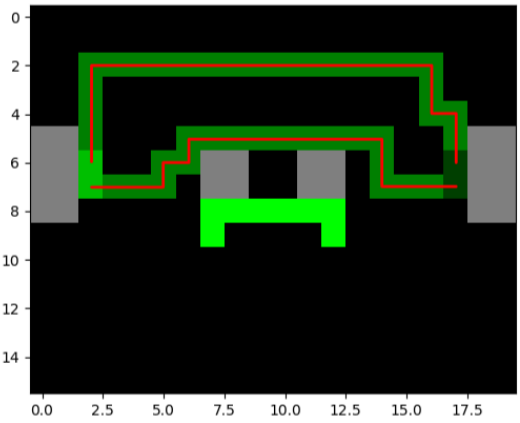

Figure 18. The final state of the *Modified StrongARM Layout* circuit with coplanar starts and targets. Routes (shown in red) are successfully drawn for all pairs of starts and targets.

The routes drawn by the RL agent successfully connect the respective start and target pairs, and are optimally drawn because they are far from the digital wires (shown in neon green), while also optimizing for the wire length of each route and the total number of vias taken. Clearly, this result is much better than the routing generated by the Lee algorithm, since the wires placed by the RL agent have significantly lower coupling capacitance with the clock wire.

4.5.5 RL Agent Performance with Non-coplanar Starts and Targets.

We also changed the *Modified StrongARM Layout* by moving the target coordinates up one metal layer, so that the agent is forced to take a via in each route. This would allow us to understand where the agent decides to take a via if it is necessary to reach the target position. Figure 19 depicts the initial state of the *Modified StrongARM Layout* circuit with non-coplanar starts and targets.

After training the RL agent on this circuit, we plot the average reward earned by the agent in each episode in Figure A13. We also plot the Q-net model loss throughout the training process in Figure A14. Finally, we perform an evaluation rollout to visualize the routes drawn by the agent, depicted in Figure 20.

The routes drawn by the RL agent successfully connect the respective start and target pairs, and are all optimally drawn because they are far from the digital wires (shown in neon green), while also optimizing for the wire length of each route.

Because we did not extend our intermediate coupling capacitance calculation to 3D, the RL agent ends up drawing a route directly above the clock wire. In reality, this route would suffer from the undesired coupling capacitance in the z axis to the clock signal wire. To solve this, we propose to

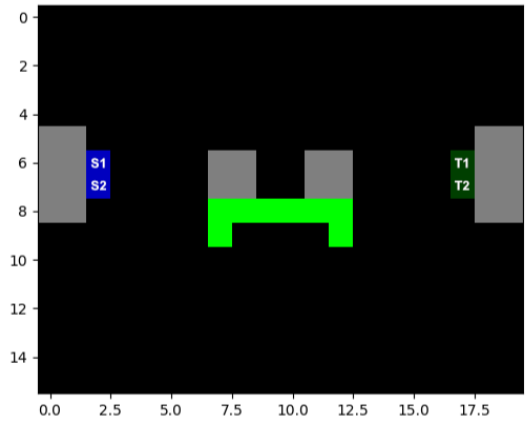

Figure 19. The starting state of the *Modified StrongARM Layout* circuit with non-coplanar starts and targets. Starting positions are labeled with a S and target positions are labeled with a T.

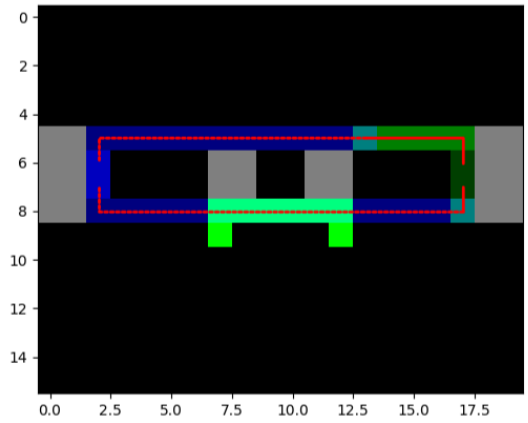

Figure 20. The final state of the *Modified StrongARM Layout* circuit with non-coplanar starts and targets. Routes (shown in red) are successfully drawn for all pairs of starts and targets. Dotted lines correspond to wires on the topmost metal layer (which also contains the target positions).

expand our intermediate coupling capacitance calculation to 3D and also choose $\alpha_1 \neq \alpha_2$ to account for different wire pitches and dielectrics between successive metal layers.

4.6 Runtime and Performance Analysis

The RL agent can be trained to draw routes for simple circuits in as little as a few hours running only on CPU and a couple million environment steps during the training process.

Table 1 lists the total time (in hours) and number of environment steps taken to train the RL agent for different circuits, and its performance in each environment. We calculate the score by dividing the average reward by the number of routes in the circuit.

Env Name	Steps	Time	Score
Grid2D	1M	0h 5m	127
Expanded Grid2D	2.5M	0h 26m	56
Grid3D Coplanar	5M	2h 16m	68
Grid3D Non-coplanar	5M	1h 59m	9
StrongARM Coplanar	15M	4h 27m	20
StrongARM Non-coplanar	15M	4h 39m	20

Table 1. The training time and number of environment steps taken, along with agent performance for a variety of circuits. The score is calculated by dividing the average reward by the number of routes in the circuit.

5 Conclusion

5.1 Contributions

In summary, we have developed an RL-based framework to generate analog routes that minimize parasitic noise coupling to nearby digital signals. This noise coupling can significantly degrade the performance of mixed-signal circuits like ADCs and DACs, and thus the generation of optimized routes is an important step in mixed-signal design automation.

Based on an action space of possible directions to route and an observation space of existing analog wires, digital wires, and pre-placed blocks, the RL agent optimally chooses a direction to move in for each step of the route. We also developed two custom coupling capacitance calculation methods to evaluate the severity of noise coupling, and we incorporated this capacitance as a penalty term into our reward function. We tuned the hyperparameters of the reward function to yield a balanced tradeoff between minimal wirelength and minimal noise coupling to achieve best performance.

Finally, we demonstrate the efficacy of our RL framework on a variety of circuit topologies, from simple 2D and 3D examples to a modified layout of a commonly used circuit — the StrongARM comparator.

5.2 Analysis

5.2.1 RL Agent vs. Human Experts.

For real-world circuit routing, most of the times analog design engineers will manually place blocks, manually draw routes, and then iteratively run simulations to evaluate their layout and make tweaks to optimize the performance. This process is very slow and requires the involvement of a human expert (thus making it impossible to simulate). As a result, it is difficult to compare the performance of the RL agent against that of an expert engineer, but we believe that the RL agent produces routes similar to ones a human expert would draw (considering the pre-placement of circuit blocks on the layout). The RL agent demonstrates an ability to produce optimized routes much faster than human engineers, which

is a useful step in the machine-learning based automation of circuit layout.

5.2.2 Relevance to Optimal Placement, Bootstrapping.

Our RL agent framework is designed to optimize routing for a given placement of blocks and routes. However, this work also provides useful insights into the placement of circuit blocks before routing is performed. Specifically, our RL agent can be used to evaluate the "quality" of a given placement, by evaluating (or approximating) the total routing cost of all wires in the circuit, given the initial placement. Furthermore, this can be incorporated into the cost function of a block placement engine, allowing it to generate better placements that reduce the total routing cost.

This can also be extended further with bootstrapping to generate random coarse block placements, make fine adjustments in the block positions, and prune the placements that will not yield satisfactory routing results. With this method, block placement engines will be able to explore the design space more efficiently, and also output better final layouts. This would further reduce the gap between the RL agent and an expert human designer because our framework can "sweep" through placement options and predict the corresponding routing cost to optimally select both placement and routing for AMS designs.

5.3 Future Work

This is just the beginning of circuit routing automation, and there is still a lot of work to be done in this space. Below, we detail scope for improvement within our work and further extensions of this project.

5.3.1 Training with Pre-Trained Agents.

Ideally, we think it would be useful to evaluate the ability for a pre-trained DQN model to be fine-tuned for a specific new circuit problem, and evaluate whether it can produce optimized routes with minimal fine-tuning.

This would require a more generic DQN model architecture, which is agnostic of the circuit layout dimensions and includes modules to extract various features from the layout that are relevant to the current route being drawn. This could be done using convolutional neural networks (CNNs) with varying kernel sizes, among other types of models.

5.3.2 Testing Generalizability of RL Agent.

Furthermore, we would like to benchmark the generalizability of the RL agent, and understand its single-shot performance on new circuits outside of the training distribution. Of course, to do this, the DQN model must be trained on a larger range of test circuit problems, which is certainly a challenge for research in this domain.

5.3.3 Fine-tuning RL Agent with Human Feedback.

Another way we think the RL agent could be improved is using reinforcement learning with human feedback (RLHF). With RLHF, the reward function is a combination of the traditional reward function, along with a human-provided feedback (a score that rates the quality of the agent's output). RLHF would allow for a more flexible reward function that can better represent the "intuitive" and "subjective" differences in routes (e.g. routes that are clean, evenly spaced, symmetric, etc.). This would allow the RL agent to learn to optimize for a reward function that is more effective than a manually designed reward function.

5.3.4 Automated Hyperparameter Tuning.

To further optimize the reward function we manually designed, we propose the use of automated hyperparameter tuning, in which a set of training processes can be run in parallel, with different learning parameters and reward function weights. Although we manually optimized these parameters, we were not able to extensively sweep over wide ranges of values, and this is something that could improve the RL agent's performance further. It would also be effective to utilize cloud compute servers to speed up and parallelize the training process.

5.3.5 Testing with More Real-World Circuits.

To test the scalability of our RL framework, we propose testing on a wider collection of circuits — both in terms of functional diversity and layout size. We believe functional diversity is important because our framework may work better on certain types of circuits that lend themselves to routing-based optimizations. For example, clocked circuits that are susceptible to interference from the hard switching clock signals may prove to be good candidates. At the same time, we would like to test our framework on larger layouts to understand how successfully our agent can explore larger observation spaces and reach the desired targets.

6 Attributions

This project was a group project, between students Avikam Chauhan and Ashwin Rammohan. We contributed equally to the programming work and writing the report as well as discussions about reward function tweaks and coupling capacitance calculation methods. Avikam led the development of the RL agent software and configurable circuit description framework, while Ashwin led the design and development of various test circuit problems and the implementation of the Lee routing algorithm.

Acknowledgments

We would like to thank the UC Berkeley CS 294-256 course staff (Professor John Wawrzynek and Josh Kang) and the UC Berkeley CS 285 course staff (Professor Sergey Levine, Kyle Stachowicz, Vivek Myers, Joey Hong, and Kevin Black) for their feedback on this work.

References

[1] Hao Chen, Kai-Chieh Hsu, Walker J. Turner, Po-Hsuan Wei, Keren Zhu, David Z. Pan, and Haoxing Ren. 2023. Reinforcement Learning Guided Detailed Routing for Custom Circuits. In *Proceedings of the 2023 International Symposium on Physical Design.*

[2] Mark Po-Hung Lin, Po-Hsun Chang, Shuenn-Yuh Lee, and Helmut E. Graeb. 2016. DeMixGen: Deterministic Mixed-Signal Layout Generation With Separated Analog and Digital Signal Paths. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 8 (2016), 1229–1242. https://doi.org/10.1109/TCAD.2015.2501295

[3] Sreenivasulu Polineni, M. S. Bhat, and S. Rekha. 2020. A Switched Capacitor-Based SAR ADC Employing a Passive Reference Charge Sharing and Charge Accumulation Technique. *Circuits, Systems, and Signal Processing* 39, 11 (2020), 5352–5370. https://doi.org/10.1007/s00034-020-01437-3

[4] F. Schembari, G. Bellotti, and C. Fiorini. 2016. A 12-bit SAR ADC integrated on a multichannel silicon drift detector readout IC. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 824 (2016), 353–355. https://doi.org/10.1016/j.nima.2015.08.036 Frontier Detectors for Frontier Physics: Proceedings of the 13th Pisa Meeting on Advanced Detectors.

[5] Hai Zhou. 2014. *Detailed routing: shortest path and maze search.* http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf

[6] Keren Zhu. 2023. *Fully-automated Layout Synthesis for Analog and Mixed-Signal Integrated Circuits.* Ph. D. Dissertation. University of Texas at Austin, Austin, TX.

A Training Curves

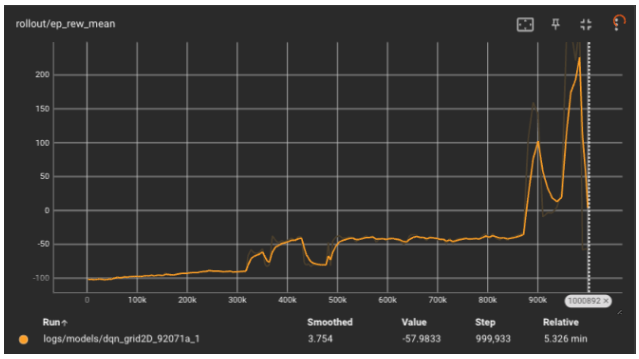

Figure A1. The average reward gained by the RL agent in each episode throughout the training process for the *Simple 2D Grid* circuit.

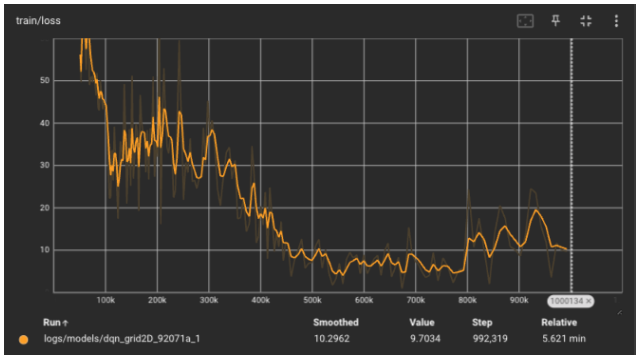

Figure A2. The Q-net model loss throughout the training process for the *Simple 2D Grid* circuit.

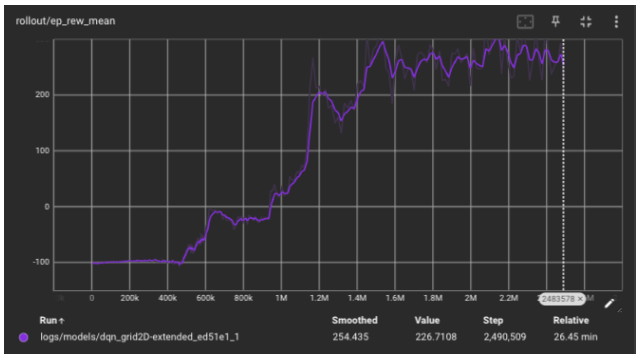

Figure A3. The average reward gained by the RL agent in each episode throughout the training process for the *Expanded 2D Grid* circuit.

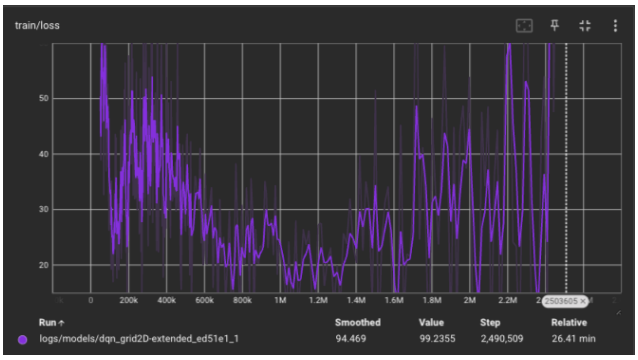

Figure A4. The Q-net model loss throughout the training process for the *Expanded 2D Grid* circuit.

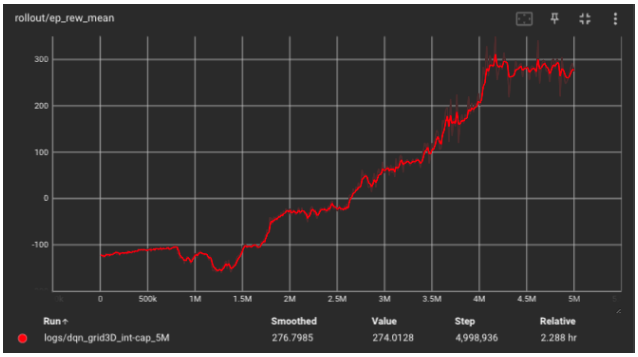

Figure A5. The average reward gained by the RL agent in each episode throughout the training process for the *Expanded 3D Grid* circuit with coplanar starts and targets.

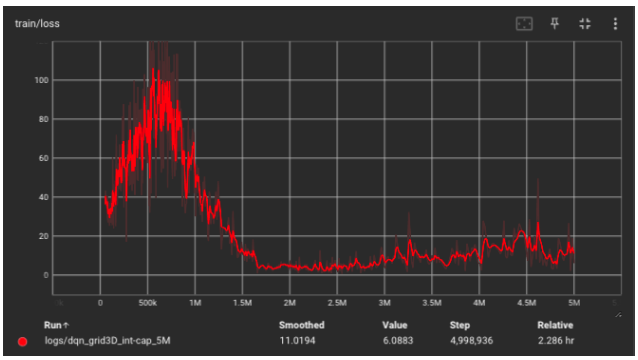

Figure A6. The Q-net model loss throughout the training process for the *Expanded 3D Grid* circuit with coplanar starts and targets.

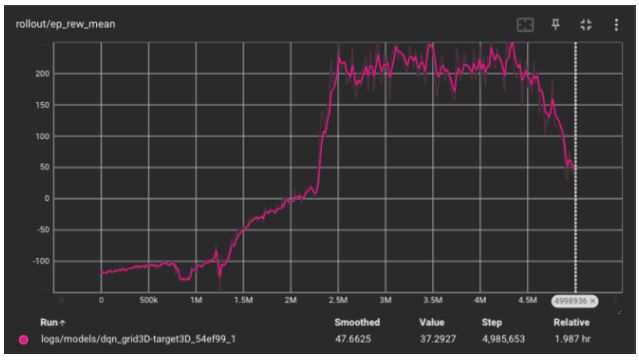

Figure A7. The average reward gained by the RL agent in each episode throughout the training process for the *Expanded 3D Grid* circuit with non-coplanar starts and targets.

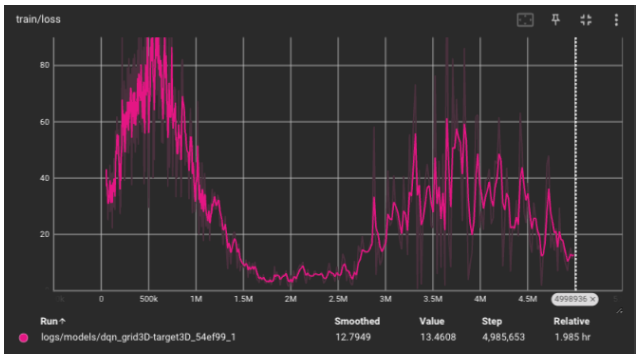

Figure A8. The Q-net model loss throughout the training process for the *Expanded 3D Grid* circuit with non-coplanar starts and targets.

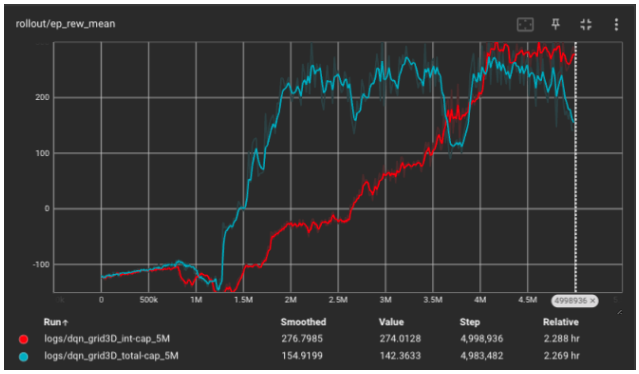

Figure A9. The average reward gained by the RL agent in each rollout throughout the training process. The red line corresponds to the intermediate coupling capacitance calculations, and the blue line corresponds to the total coupling capacitance calculations.

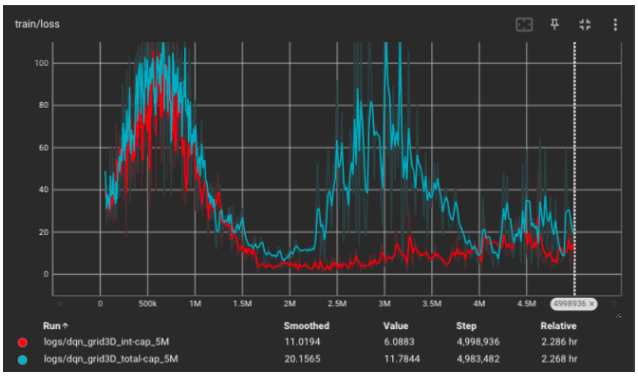

Figure A10. The Q-net model loss throughout the training process. The red line corresponds to the intermediate coupling capacitance calculations, and the blue line corresponds to the total coupling capacitance calculations.

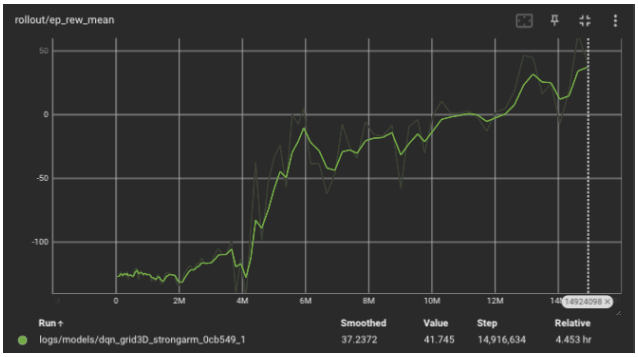

Figure A11. The average reward gained by the RL agent in each episode throughout the training process for the *Modified StrongARM Layout* circuit with coplanar starts and targets.

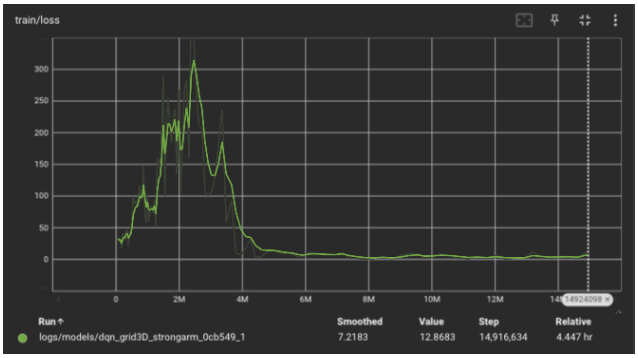

Figure A12. The Q-net model loss throughout the training process for the *Modified StrongARM Layout* circuit with coplanar starts and targets.

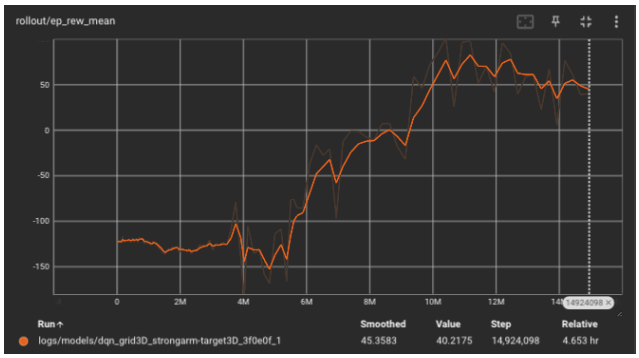

Figure A13. The average reward gained by the RL agent in each episode throughout the training process for the *Modified StrongARM Layout* circuit with non-coplanar starts and targets.

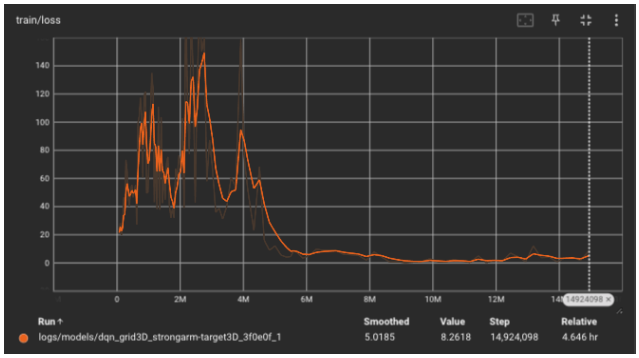

Figure A14. The Q-net model loss throughout the training process for the *Modified StrongARM Layout* circuit with non-coplanar starts and targets.