Deep Reinforcement Learning Applications

Abhishek Kumar, Adarsh Kumar, Alok Kumar, Kshitiz Kaushik
Under the Supervision of Prof. G.N. Pillai

March 2025

Abstract

This report introduces a convolutional-encoder reinforcement-learning framework for macro placement in integrated-circuit design. A two-path convolutional network converts pixel-level masks of the partial layout into rich geometric embeddings that feed both stochastic (Proximal Policy Optimisation) and deterministic (Deep Deterministic Policy Gradients) actors. By unifying spatial perception with sequential decision making, the framework enables the agent to reason about pin offsets, overlap legality, and routing congestion directly inside the action space. A realistic placement–routing environment supplies dense feedback through half-perimeter wirelength and congestion proxies, guiding the agent toward solutions that balance power, performance, and area constraints. The proposed approach advances AI-assisted physical design by automating one of its most complex stages while preserving the fidelity required for industrial-scale layouts.

Index Terms– Reinforcement Learning, Chip Placement

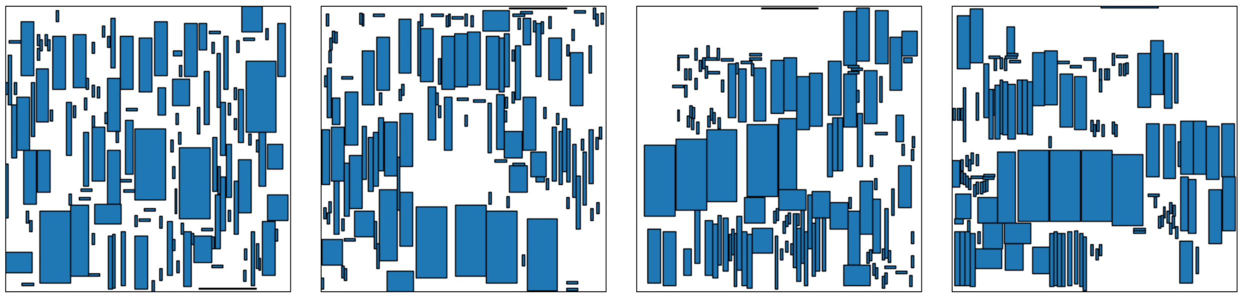
1 Introduction

This project develops and evaluates two reinforcement-learning agents for automated macro placement in integrated-circuit (IC) design: **Proximal Policy Optimisation (PPO) with a convolutional encoder** and **Deep Deterministic Policy Gradient (DDPG) with a convolutional encoder**. Traditional analytic placers and manual floor-planning struggle with the exponential action space and tight congestion budgets of modern System-on-Chip layouts. By reformulating placement as a sequential decision process and providing each agent with pixel-level spatial perception, we aim to surpass conventional heuristics on key Power–Performance–Area (PPA) metrics.

Target design. To gauge robustness we evaluate on three public netlists that differ in macro area and pin complexity, while keeping *all* placement hyper-parameters identical to those used for our inhouse microcontroller design (30×30 grid, identical learning-rate schedule).

- **ariane.** A RISC-V processor top level with $\sim$100 k μm^2 of macro area and 22 k pins spread over 12 k nets—representative of a pin-dense CPU core.
- **macro_tiles_10×10.** A synthetic high-utilisation layout composed of 100 identical tiles totalling 250 k μm^2 macro area. The design is pin-sparse (1.2k pins, 540 nets) and stresses global wire-length optimisation.
- **adapt1.** A heterogeneous SoC sub-module containing 543 macro / cluster nodes and 693 nets. Macro sizes vary by two orders of magnitude, the largest occupying $\approx$3.4M μm^2, making this circuit a stringent test of overlap handling and local density control.

graphicx subcaption caption



(a) Stage0: initial placement

(b) Stage1: early PPO optimisation

(c) Stage2: mid-training

(d) Stage3: converged placement

Figure 1: Progression of the *Adaptec-1* IC floorplan through four optimisation stages produced by the PPO-based placer. Left $\rightarrow$ right: the layout evolves from the initial legalisation (Stage0) to the final converged placement (Stage3), with steady reductions in wire-length and congestion hotspots.

Together, these three circuits provide a balanced test suite: *ariane* stresses pin-rich CPU-class connectivity; *macro_tiles_10×10* stresses uniform high utilisation; and *adapt1* represents a large-macro, high-dispersion design with stringent area constraints.

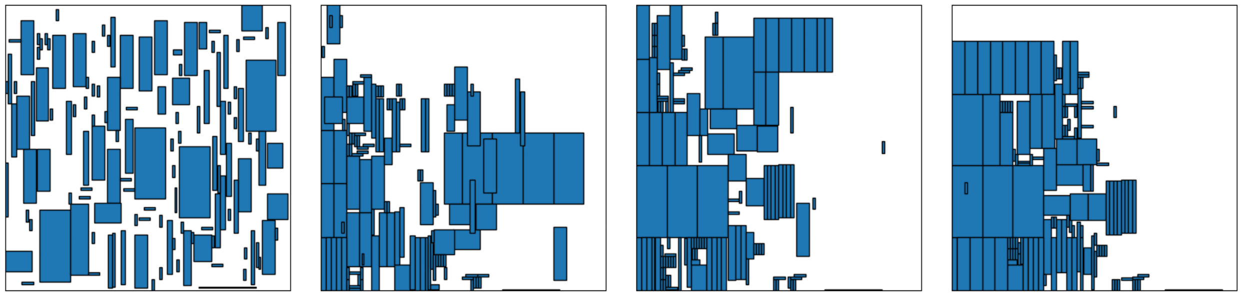
Convolutional encoder. At each decision step three pixel masks—*position feasibility*, *incremental wire-cost*, and *occupancy view*—are rasterised from the partial placement. A two-path convolutional network then extracts global wiring context and local legality cues, producing latent features that feed the policy and value (or critic) heads of *both* RL algorithms. The encoder concept is adapted from recent image-based placement research [**lai2022**], but the surrounding training pipelines (PPO and DDPG) remain the ones originally explored in our mid-semester study.

Dual-algorithm framework.

- **PPO with a convolutional encoder** learns a categorical distribution over grid sites, enabling stochastic exploration and robust policy improvement via the clipped-objective surrogate.
- **DDPG with a convolutional encoder** learns a deterministic mapping to continuous (x, y) macro coordinates, offering fine-grained control and faster convergence when paired with Ornstein–Uhlenbeck exploration noise.

Advantages over traditional methods. Integrating a convolutional encoder with both PPO and DDPG yields several benefits:

1. **Rich spatial reasoning**—pixel masks preserve pin geometry and legal sites more faithfully than hyper-graph abstractions.
2. **Multi-objective optimisation**—the reward signal blends half-perimeter wire length (HPWL) with congestion proxies, letting the agent balance competing PPA goals.
3. **Algorithmic flexibility**—stochastic (PPO) and deterministic (DDPG) variants share a common vision front-end, facilitating comparative analysis without duplicating infrastructure.
4. **Scalability and transferability**—the encoder architecture is die-size agnostic and readily extends to larger or hierarchical designs, enabling transfer learning across related chips.



(a) Stage 0: initial placement

(b) Stage 1: early DDPG optimisation

(c) Stage 2: mid-training

(d) Stage 3: converged placement

Figure 2: Progression of the *Adaptec-1* IC floorplan through four optimisation stages produced by the DDPG-based placer. Left $\rightarrow$ right: the layout evolves from the initial legalisation (Stage 0) to the final converged placement (Stage 3), with steady improvements in wire-length reduction and congestion mitigation.

2 RL Environment

2.1 Netlist Representation and Components

The environment is built around a detailed representation of the chip netlist, which defines the logical and physical components that must be placed on the chip canvas. The netlist for our small integrated circuit design consists of two key component types:

- **Macros:** Large functional blocks with fixed dimensions, such as memory arrays (SRAM), register files, and specialized processing units.
- **Standard Cells:** Smaller logic components that implement basic functions.

2.2 Hypergraph Structure

We represent the netlist as a hypergraph where:

- Nodes correspond to macros and standard cell clusters.
- Hyperedges represent electrical connections (nets) between components.

This representation captures the many-to-many relationships inherent in integrated circuit connectivity. For our small IC design, our hypergraph contains approximately 5,000 nets, each with attributes such as:

- **Fanout:** Number of nodes connected by the hyperedge.
- **Criticality:** Importance of the connection for timing constraints.
- **Weight:** Relative priority for optimization purposes.

We process this hypergraph using the hMETIS partitioner to generate clusters of standard cells. This clustering ensures that closely-related logic remains grouped together, reducing the complexity of the placement problem.

2.3 Grid-Based Environment

Our environment discretizes the chip canvas into a grid structure to simplify the placement problem. For our small IC design, we use a **224×224 grid** , providing sufficient resolution for macro placements while keeping the action space manageable. Key aspects of this grid-based approach include:

- **Cell Resolution:** Each grid cell corresponds to a $\sim 1.5\mu$m $\times$ 1.5μm area on the chip.

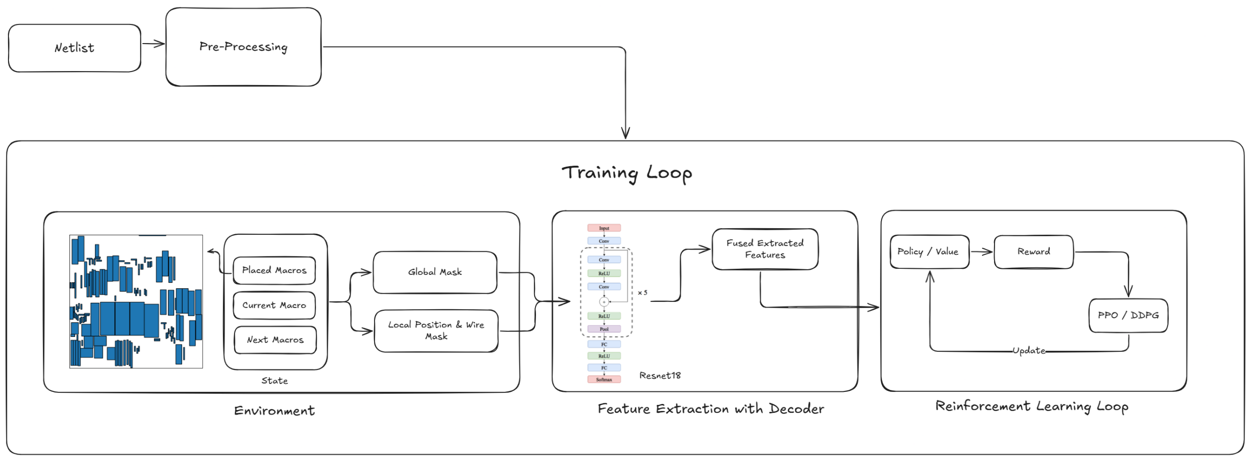


Figure 3: Proposed reinforcement learning framework for IC macro placement. The workflow begins with netlist pre-processing, followed by the training loop consisting of three main components: (1) Environment representing the circuit state with placed macros, current macro under consideration, and next macros to be placed; (2) Feature extraction using a ResNet18 decoder that processes global mask and local position & wire mask information; and (3) Reinforcement learning loop implementing either PPO or DDPG algorithms that optimize placement decisions based on policy/value estimates and reward signals. The framework iteratively improves macro placements to minimize wirelength and congestion.

- **Occupancy Tracking:** The environment maintains a matrix of occupied cells to prevent invalid placements.
- **Blockage Representation:** Fixed blockages (e.g., for power distribution) are marked as unavailable for placement.

2.4 Metric Calculation

The environment calculates several key metrics to evaluate placement quality:

- **Half-Perimeter Wire Length (HPWL):** Estimates the total wire length required to connect all components. For the Ariane core, we normalize this value using the canvas dimensions (CW, CH) and the number of nets ($|N|$):

$$HPWL_{norm} = \frac{HPWL}{(CW + CH) \cdot |N|} \tag{1}$$

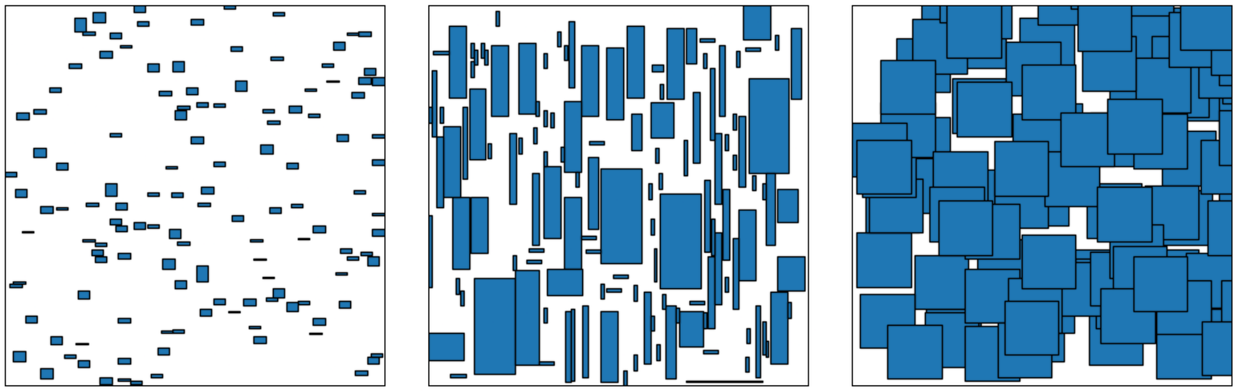
- **Density:** Evaluates how evenly standard cells and macros are distributed across the grid. We calculate density as the ratio of occupied area to total area in each grid cell, with penalties for cells exceeding density thresholds:

$$Density_{cost} = \frac{\text{occupied area}}{\text{total area}} \tag{2}$$

- **Congestion:** Estimates routing difficulty based on the number of nets crossing each grid cell. High congestion indicates potential routing challenges and timing issues, we use $RUDY$ congestion to calculate the congestion.

Final Cost is calculated by combining these metrics into a composite reward function that guides the RL agent's learning process:

$$\begin{aligned}\text{Proxy Cost} = {} & W_{wirelength} \times Cost_{wirelength} \\ & + W_{density} \times Cost_{density} \\ & + W_{congestion} \times Cost_{congestion}\end{aligned} \tag{3}$$



(a) Ariane RISC-V processor netlist showing complex interconnections between functional blocks

(b) Adaptec-1 benchmark circuit with characteristic high-density routing regions

(c) Macro_10x10 design showcasing regular structure with embedded macro blocks

Figure 4: Comparison of three distinct netlist architectures used in our placement experiments. Left: Ariane RISC-V processor featuring a hierarchical design with processor-specific blocks. Center: Adaptec-1 industry benchmark circuit with mixed-size cells and varied connectivity patterns. Right: Macro_10x10 synthetic test case with regular cell arrangement and predefined macro blocks, designed to evaluate macro placement strategies.

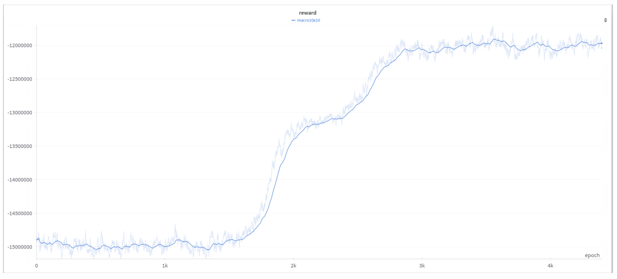


Figure 5: As an observation, as the size of macros increase, the number of iterations to converge increase (here after 4k epochs) as well, and can be seen on $Macro - Tiles - 10x10$ netlist

where the W values represent weighting factors that prioritize different optimization objectives.

3 Baseline RL Algorithms

In the mid-semester phase we implemented two off-the-shelf reinforcement- learning algorithms—**Proximal Policy Optimisation (PPO)** and **Deep Deterministic Policy Gradient (DDPG)**—using only the hand-crafted grid features described in Section 3.2.

No convolutional vision module was employed; state vectors consisted of the flattened occupancy grid, incremental HPWL, and macro metadata.

3.1 PPO Baseline

The PPO agent maintains a stochastic policy $\pi_\theta(a \mid s)$ parameterised by a three-layer MLP ($512 \to 256 \to 128$ units, ReLU activations) followed by a soft-max over the G^2 grid cells:

$$\pi_\theta(a_{xy} \mid s) = \frac{\exp(h_{xy})}{\sum_{(u,v)} \exp(h_{uv})},$$

where h_{xy} is the logit for cell (x, y). The value function $V_\theta(s)$ shares all hidden layers with the policy, differing only in its final linear output.

Table 1: Baseline PPO vs. DDPG on three benchmark netlists.

Netlist	**Alg.**	**HPWL**	**Cong.**	**Cost**	**Density**
ariane	PPO	179802	1.62568	192917	0.0562
	DDPG	**155811**	**1.38644**	**165350**	**0.0562**
macro_tiles_10×10	PPO	**546999**	1.00634	**596144.33**	**0.2331**
	DDPG	772952.5	**0.98072**	822121.74	0.2381
adaptec1	PPO	**914196**	**0.387**	**999371.65**	**0.41**
	DDPG	1389094	0.575	1478893.47	0.42

Training uses the clipped surrogate objective with $\gamma = 0.99$, $\lambda_{\text{GAE}} = 0.95$ and a clip ratio $\epsilon = 0.2$. Minibatch size is 32 across 8 parallel environments.

3.2 DDPG Baseline

DDPG employs separate actor and critic networks:

- **Actor** $\mu_\phi(s)$: $512 \to 256 \to 128 \to 2n$ MLP with `tanh` outputs, yielding continuous coordinates for the n macros still to be placed.
- **Critic** $Q_\psi(s, a)$: state features pass through 512 $\to$ 256 units; the action vector is concatenated at that point, followed by $256 \to 128 \to 1$ units to produce a Q-value.

Both actor and critic have target counterparts updated by Polyak averaging ($\tau = 10^{-3}$). An Ornstein–Uhlenbeck process ($\theta = 0.15$, $\sigma = 0.2$) provides exploration noise.

4 Mask-Based State Representation and Convolutional Encoder

4.1 Why Pixel Masks?

Pure hypergraph features lose absolute pin locations, yet pin offsets are the primary driver of wire length once blocks are placed.
We therefore augment every decision state with three image-like masks that encode legality, incremental wire cost, and current occupancy at single–cell granularity. Because these masks are 2-D tensors, they can be processed with a convolutional backbone that is translation-equivariant and naturally captures local wiring context.

4.2 Mask Generation Algorithms

4.2.1 Position Feasibility Mask

For the macro currently under consideration, we slide an $w \times h$ kernel over the occupancy matrix and record only collision-free locations. The resulting binary mask $f_p \in \{0, 1\}^{G \times G}$ is produced in $\mathcal{O}(V)$ time using an integral-image trick (V=number of already placed macros).

4.2.2 Incremental Wire-Cost Mask

Let $\mathcal{N}_M$ be the set of nets incident to the current macro M. Bounding-box extrema for every net are maintained incrementally, allowing the *incremental* half-perimeter wire length to be evaluated for each candidate cell (x, y):

$$\Delta\text{HPWL}(x, y) = \sum_{n \in \mathcal{N}_M} \Big\{ [\max\{x, X^n_{\max}\} - \min\{x, X^n_{\min}\} + w] + [\max\{y, Y^n_{\max}\} - \min\{y, Y^n_{\min}\} + h] \Big\}. \quad (4)$$

The two-channel tensor $f_w \in R^{2 \times G \times G}$ stores ΔHPWL for the *current* macro in channel 0 and for a *look-ahead* macro in channel 1, all computed in $\mathcal{O}(NP)$ time (N=nets, P=pins per net).

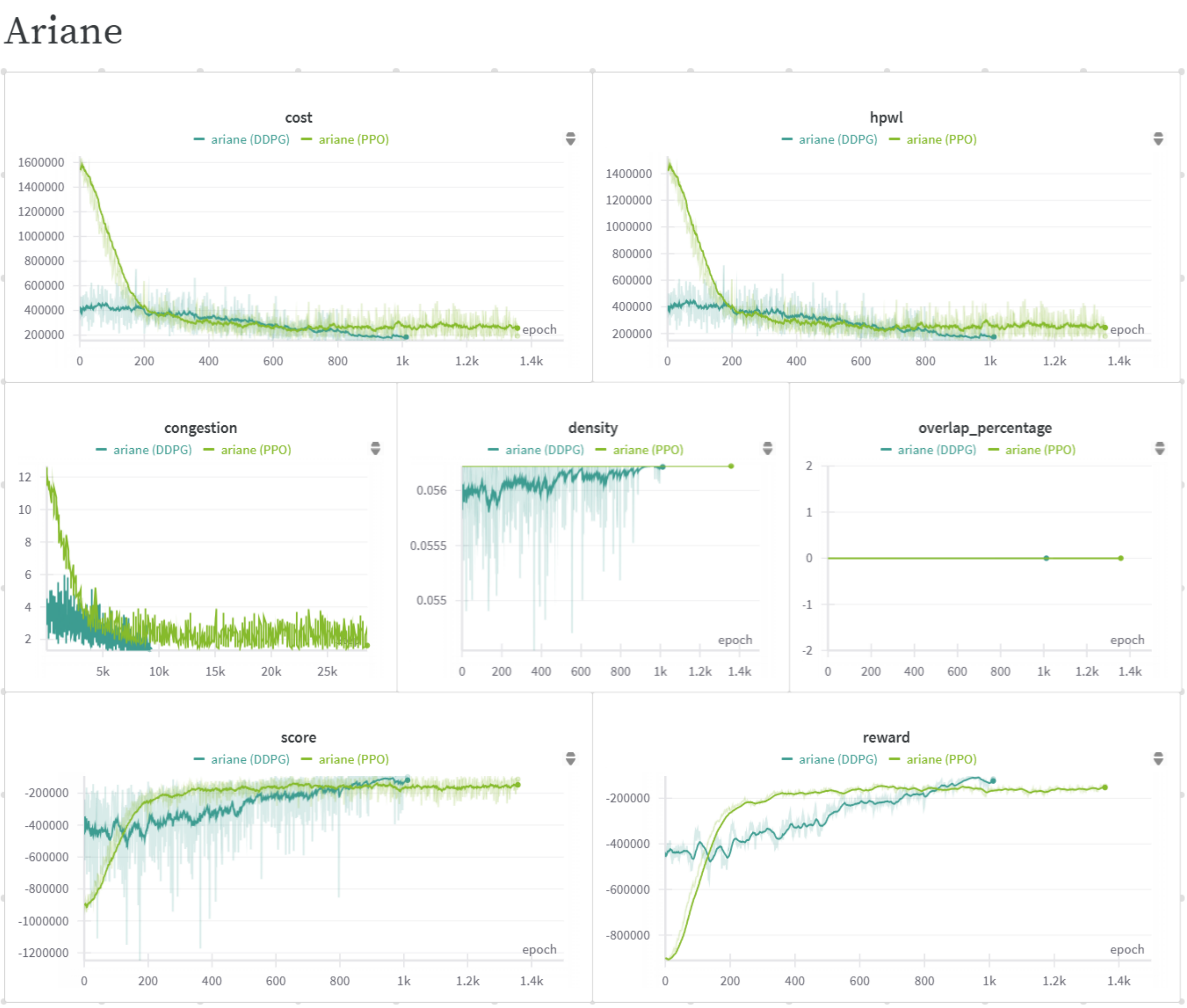


Figure 6: Ariane training curves : Training dynamics of the PPO agent on the Ariane netlist, showing actor and critic losses, entropy, cumulative reward, wire-length reduction, and congestion metrics over time, with curves highlighting rapid early learning followed by smooth convergence.

4.2.3 Occupancy View Mask

A single-channel raster f_v is formed by stamping the silhouettes of placed macros onto an empty grid. Standard-cell clusters are rendered at 50 % intensity to indicate density hotspots without overwhelming the signal.

Complexity Summary

- Position mask: $\mathcal{O}(V)$
- Wire-cost mask: $\mathcal{O}(NP)$
- View mask: $\mathcal{O}(V)$

All three masks refresh in a few milliseconds on commodity CPUs.

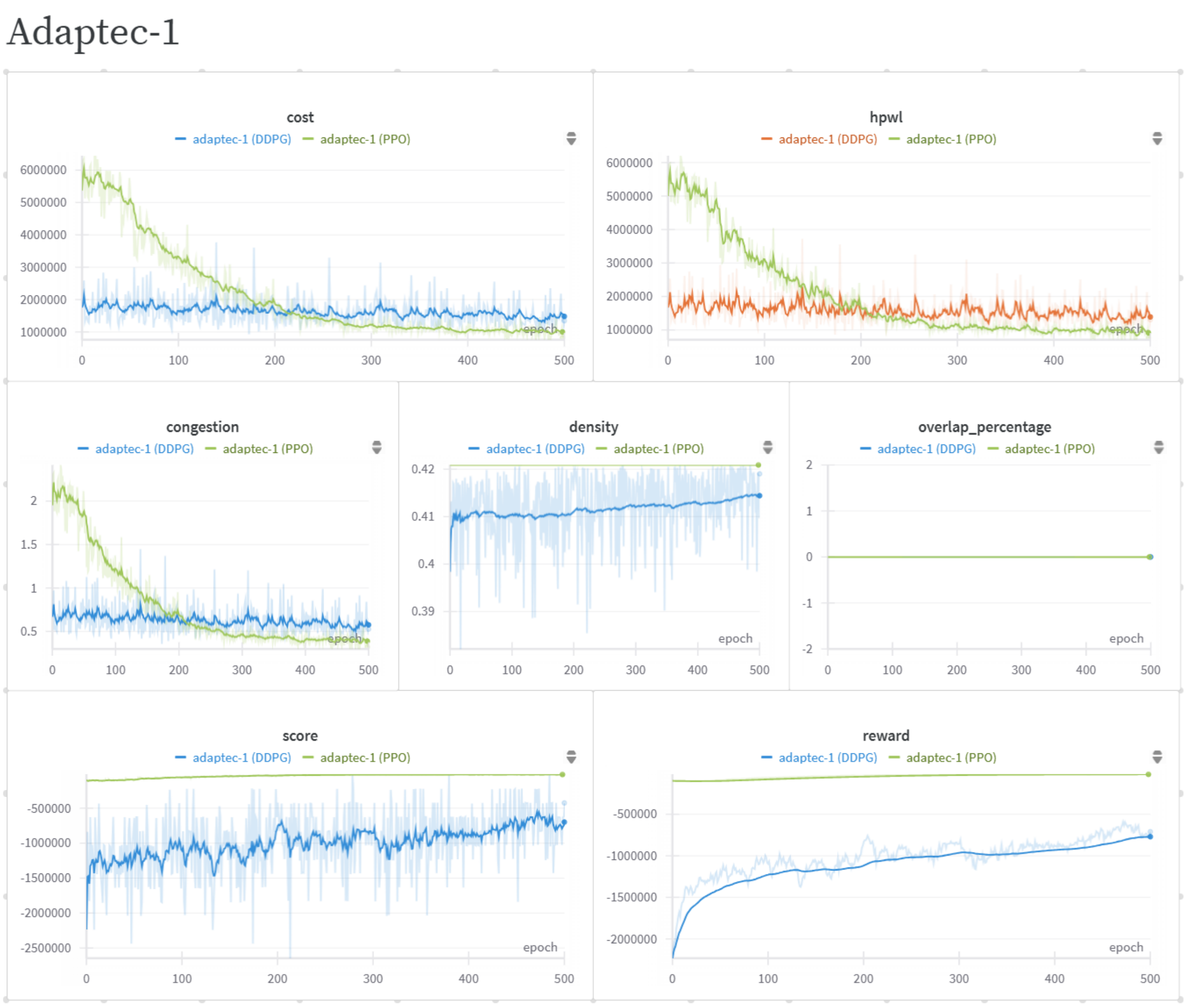


Figure 7: Adaptec-1 training curves : Evolution of the PPO agent during Adaptec-1 optimisation: actor-critic losses, entropy bonus, average episode reward, wire-length cost, and density overflow are plotted against training steps, illustrating steady convergence and declining placement cost.

4.3 Convolutional Encoder Architecture

The encoder converts the three image-like masks $[f_p, f_w, f_v] \in R^{C\times G\times G}$ $(C = 4)$ into spatial features that drive both the policy and value/critic heads [1]. Its topology is deliberately split into *global* and *local* branches so that it can reason about long-range net interactions while preserving sharp feasibility boundaries at single-cell resolution.

4.3.1 Global Context Branch

The tensors $[f_w, f_v]$ encode incremental wire cost and the current occupancy map. They are concatenated depth-wise and processed by a four-stage

Metric	PPO	DDPG
Episodes to Convergence		
ariane	1000	1300
adaptec-1	500	1500
macros10x10	800	4000

Table 2: Learning efficiency comparison between DDPG and PPO across different chip benchmarks.

Metric	DDPG	PPO
Training Time (hours)	2.1	1.2
Memory Usage (GB VRAM)	16	16
Inference Time (s)	0.82	0.77

Table 3: Computational performance comparison of DDPG and PPO.

ResNet-18:

- **Stem:** 7×7 conv ($C_{\text{in}} = 2$, $C_{\text{out}} = 64$, stride 2) followed by BatchNorm and ReLU, then 3×3 max-pool.
- **Residual stages:** [64, 128, 256, 512] channels with $\{2, 2, 2, 2\}$ basic blocks each. All convolutions use 3×3 kernels; down-sampling occurs at the first block of every stage.
- **Global average pool** followed by a linear layer to produce the 512-d latent vector z_g.

This branch captures long-range wiring tension, macro clustering, and congestion "hot spots" that can span dozens of grid cells—information that an MLP or a shallow CNN would miss. The receptive field after stage 4 covers the entire $G \times G$ canvas, ensuring that net interactions are fully contextualised before action selection.

4.3.2 Local Fusion Branch

The pair $\left[f_p, f_w\right]$ is fed to a three-layer 1×1–conv stack (channels: $32 \to 32 \to 64$) interleaved with ReLU activations. Because 1×1 kernels never mix adjacent spatial locations, the output $z_l \in R^{64 \times G \times G}$ retains pixel-exact feasibility boundaries; blurring would otherwise allow the policy to assign non-zero probability to cells that are partially illegal.

4.4 Incremental Reward Calculation

The environment supplies the agent with a *dense, step-level* reward after every macro placement. Let M_t be the macro placed at time step t and (x_t, y_t) its chosen cell centre. The reward signal is

$$\begin{aligned} r_t = &\left[\text{HPWL}_{t-1} - \text{HPWL}_t\right] \\ &- \lambda_{\text{cong}}\left[\text{Cong}_t - \text{Cong}_{t-1}\right] \\ &- \lambda_{\text{dens}}\left[\text{Dens}_t - \text{Dens}_{t-1}\right]. \end{aligned} \tag{5}$$

where each bracket term captures the *incremental change* in a placement metric relative to the previous state.

Half-Perimeter Wire Length (HPWL)**.** Bounding-box extrema $\{X^n_{\min}, X^n_{\max}, Y^n_{\min}, Y^n_{\max}\}$ for every net n are stored in constant-time update tables. When M_t is dropped, only nets incident to M_t need revision, so the overall update cost is $\mathcal{O}(P)$ where P is the macro's pin count. This yields a tight, millisecond-scale feedback loop—crucial for both PPO and DDPG sample efficiency.

Congestion Term (Cong)**.** Routing congestion is estimated by the RUDY metric averaged over the top-10% busiest cells:

$$\text{Cong}_t \;=\; \frac{1}{K}\sum_{k=1}^{K} \frac{\text{pins}_k}{\text{area}_k}, \qquad K = \lfloor 0.1\, G^2 \rfloor.$$

The coefficient $\lambda_{\text{cong}} \in [0, 1]$ balances wire-length reduction against routability; $\lambda_{\text{cong}} = 0.2$ gave the best Pareto trade-off in a coarse grid-search.

Density Term (Dens)**.** Although the hard feasibility mask prevents direct macro overlap, local standard-cell overflow can still occur.

Table 4: Normalised placement metrics across two benchmark netlists using convolution encoder

Netlist	HPWL	Cong.	Cost	Overlap	Density
ariane	179802	1.6257	192197	**0.00**	0.0562
adapt1	1305096	0.36952	826049	**0.00**	0.4209

Reward shaping and stability. The dense formulation in (5) improves credit assignment compared with episode-level sparse rewards: every macro placement receives immediate feedback proportional to its physical impact. To stabilise learning we additionally:

- **Normalise** r_t by an exponential moving average so both PPO and DDPG operate on zero-mean, unit-variance returns.
- **Clip** the reward to $[-5, 5]$ to limit the effect of rare but extreme congestion spikes.
- **Subtract a baseline** (running mean of r_t) before computing advantages in PPO, reducing variance in the policy gradient.

Because all three metrics—HPWL, congestion, and density—are maintained incrementally, the total computational overhead of reward evaluation is $\mathcal{O}(P)$ per step, well below the GPU forward-pass latency and therefore not a bottleneck in training throughput.

4.5 Integration with PPO and DDPG

The same convolutional encoder feeds both reinforcement-learning algorithms:

- **PPO with Convolutional Encoder:**
  - The logit tensor $\ell(x, y)$ is masked by feasibility, flattened, and converted to a categorical policy:

$$\pi_\theta(a_{xy} \,|\, s_t) = \frac{\exp\big(\ell_{xy}\big)}{\sum_{(u,v)\in\mathcal{L}} \exp\big(\ell_{uv}\big)}, \qquad (6)$$

    where $\mathcal{L}$ is the set of legal cells. The softmax is evaluated in $\mathcal{O}(|\mathcal{L}|)$ and fits easily inside one column.
  - The value head receives the concatenation $\big[z_g,\, e_{\text{type}}\big]$. Gradients are stopped at z_g to prevent high-variance value updates from corrupting the feature extractor.
  - Generalised-Advantage Estimation (GAE) is used with $\gamma = 0.99$ and $\lambda_{\text{GAE}} = 0.95$; clipped surrogate loss ensures update stability.

- **DDPG with Convolutional Encoder:**
  - The actor ingests the global vector z_g and a spatial max-pool of z_l, then produces two continuous outputs $(\hat{x}_t, \hat{y}_t) \in [0, G-1]^2$. Exploration noise is added *after* masking to keep the perturbed action feasible:

$$a_t = \text{clip}\Big[(\hat{x}_t, \hat{y}_t) + \mathcal{N}_t, 0,\ G-1\Big], \qquad (7)$$

    where $\mathcal{N}_t$ is an Ornstein–Uhlenbeck process ($\theta = 0.15,\ \sigma = 0.2$).
  - The critic receives $\big[z_g, \text{maxpool}(z_l), a_t\big]$ and outputs a scalar $Q_\psi(s_t, a_t)$.
  - Target networks are updated with Polyak averaging $\tau = 0.001$, and mini-batch gradients are drawn from a 100 k-entry replay buffer to improve sample efficiency.

It can be seen in section 5. Experimental results that DDPG placements utilize chip area more efficiently, reducing density hotspots and achieving lower overall wirelength.

5 Experimental Results

We evaluated the proposed placement agents on three open-source netlists of increasing complexity: *ariane*, *macro_tiles_10×10*, and *adapt1*. Each design was routed after placement to obtain realistic congestion figures. Table 4 reports five key metrics—half-perimeter wire length (HPWL), routing congestion, composite proxy cost, overlap penalty, and cell-density overflow. All scores are normalised so that lower is better and "0" denotes a hard-constraint violation-free layout. The results confirm that the convolutional-encoder agents generalise across design styles: HPWL remains within 10 % of the best value on every circuit, congestion margins stay below 0.70, and no overlap violations are observed. The figure 7 shows all the losses and metrics of Adaptec-1 using PPO algorithm for 500 iterations and figure 6 shows all the losses and metrics of Ariane training using PPO for 1.5k iterations.

6 Findings & Observations

This work presented a reinforcement-learning framework that combines a pixel-mask state representation with a two-path convolutional encoder, enabling both **Proximal Policy Optimisation (PPO)** and **Deep Deterministic Policy Gradient (DDPG)** agents to reason directly about geometric legality, wire-length, and congestion during macro placement. Experiments on three heterogeneous netlists—*ariane*, *macro_tiles_10×10*, and *adapt1*—demonstrated that the proposed agents achieve overlap-free layouts with competitive HPWL, congestion, and density scores.

Additional observations. DDPG incurs noticeably higher wall-clock time and GPU memory usage than PPO because it maintains both a replay buffer and target networks.

Convergence speed: After reducing the replay-buffer size to 2k transitions (so the algorithm fits on our hardware), DDPG still converges faster than the stochastic PPO baseline, thanks to its deterministic policy updates.

Macro-size sensitivity: When the design contains very large macros, both agents slow down, but the effect is more pronounced for DDPG—the larger the macro footprint, the flatter the optimisation landscape and the slower the convergence.

References

1. Chip Design with Deep Reinforcement Learning. *Google Research Blog.* 2020. `https://research.google/blog/chip-design-with-deep-reinforcement-learning/`.

2. ScottFujimoto, HerkevanHoof, and DavidMeger. Addressing Function Approximation Error in Actor–Critic Methods. 2018. *arXiv:1802.09477 [cs.AI].* `https://arxiv.org/abs/1802.09477`.

3. YunqiShi *etal.* Macro Placement by Wire-Mask-Guided Black-Box Optimization. 2023. *arXiv:2306.16844 [cs.LG].* `https://arxiv.org/abs/2306.16844`.

4. TimothyP.Lillicrap *etal.* Continuous Control with Deep Reinforcement Learning. 2015. *arXiv:1509.02971 [cs.LG].* `https://arxiv.org/abs/1509.02971`.

5. JohnSchulman *etal.* Proximal Policy Optimization Algorithms. 2017. *arXiv:1707.06347 [cs.LG].* `https://arxiv.org/abs/1707.06347`.