Adaptive Region-Aware Reinforcement Learning for Multi-Objective VLSI Placement Refinement

Foteini Oikonomou
PhD Candidate, Dept. of Electrical & Computer Engineering
University of Thessaly, Volos, Greece

13/07/2026 TWIN-RELECT – 2n Scientific Workshop Program

The problem: placement

Where every cell sits on the chip determines its power, performance and area

- Placement decides the physical position of every logic cell on the die.
- It is NP-hard. Decades of heuristics and analytical solvers have refined it.
- After global placement and legalization, a small amount of quality is still on the table.
- That last step is post-legalization refinement, the focus of this work.

Why reinforcement learning?

And why PPO with a graph neural network

Why RL at all

1. Placement is a sequence of decisions, a natural fit for RL.
2. RL learns a policy instead of following a hand-tuned heuristic.
3. Crucially: RL does not need a differentiable objective. It learns from reward. That matters later, when the objectives stop being smooth.

Why a GNN

1. A netlist is a graph. A GNN reads the connectivity directly.
2. So the agent sees which cells are electrically related, not just where they sit.

Why PPO

1. PPO: the standard policy-gradient algorithm, the same one used to fine-tune large language models.
2. It is a stable, well-understood on-policy algorithm.

If RL fails here, PPO is not the reason.

These were starting choices, not conclusions. Part of the work is testing whether they were the right ones.

Why standard cells, not macros?

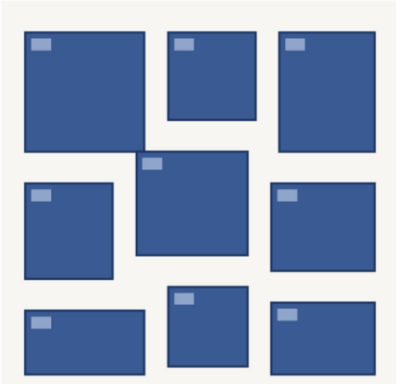
This is the choice that makes the problem hard and interesting

-Few, large, high-impact placements

- Each move visibly changes the reward

- Where RL first succeeded (AlphaChip)

Macro placement



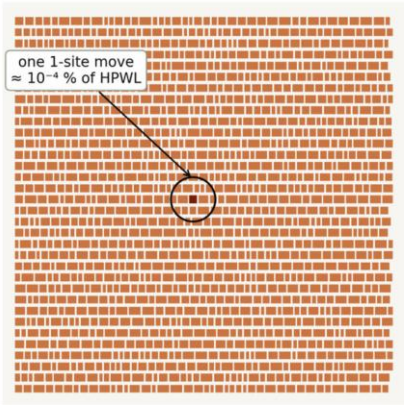
tens of decisions

- Many, tiny, individually negligible moves

- One move barely moves the reward at all

- Largely unexplored for RL

Standard-cell refinement



hundreds of thousands

Research direction : Does RL scale from decision-sparse to decision-dense problems?

RL refinement vs analytical placers

Where analytical placers win, and where they run out

- Analytical placers (OpenROAD, DREAMPlace) do gradient descent on a smooth wirelength function.
- They are fast, near-optimal, and hard to beat on wirelength.
  - But they need an objective they can differentiate.
  - Timing, congestion and reliability rules are approximated with hand-tuned weights.
- RL needs no gradient. It can, in principle, optimize the real objective directly.

The question is not whether RL beats them on wirelength. It is whether RL can go where gradients cannot.

What we did: a systematic exploration

GNN-PPO across 18 configurations, three ISPD2015 benchmarks

Architectures

GAT · GCN · GraphSAGE

Action spaces

1-site · 3-site · multi-site

Rewards

local · normalized · per-net · scaled

Training

full-graph vs. sub-region

Benchmarks: fft_1 (32K cells) · pci_bridge32 (30K) · matrix_mult (155K). Baseline: OpenROAD.

Promising findings

Good peaks during training. Collapse at inference.

Best training peak

5.85%

HPWL reduction (pci_bridge32)

Same agent, at inference

0.43%

the learned policy does not hold

- The reward curves show the agent is not converging on a stable policy. It stumbles onto good states rather than learning to reach them.
- So the limitation is not solution quality. It is whether anything is being learned at all.

Region-aware: the structure of the design matters

It is regular and irregular at the same time

- Today we classify a whole netlist with a single label. Real chips do not work that way.
- A datapath is regular. The control logic beside it is not. One training strategy cannot serve both.
- The next step is to make the classification adaptive, per region, not per design.

same netlist,
different needs

Adaptive, region-aware training, the first half of the title.

And wirelength is not the real goal

Timing is. That is the second half of the title.

- We optimize half-perimeter wirelength because it is cheap. But designers also care about timing.
- Adding a timing term is the natural next objective, and it raises a research question: how does an agent balance objectives that pull against each other?
- The cost is real. Full timing analysis per move is intractable, so an incremental or surrogate estimate is required.

now	next	later
wirelength	**wirelength + timing**	**+ physical reliability**

Summary

The agent is drowning in decisions that are too small to matter

- Every failure we found comes back to the same thing.
- The agent makes hundreds of thousands of decisions, and each one is almost invisible in the reward.
- So it cannot tell which move helped, and it difficult settles on a policy.

Our strategy: make the decisions fewer, and make each one matter.

Future work

- Move groups of cells, not one cell at a time. Fewer decisions, each with real impact.
- Allow swaps, so a single action does more work.
- Score regions, not whole designs, so the agent works on a piece it can actually see.
- Reward at the end of the episode, so the signal is not lost in the noise.
- Run after OpenROAD, not against it, and look for what it left behind.
- Add timing to the reward, so we optimize what designers actually care about.
- Apply standard ML engineering for stability: reward normalization, entropy regularization, learning rate schedules.

Goal: Show whether reinforcement learning can recover placement quality that analytical refinement leaves behind.