Extended Abstract

Motivation Macro placement fixes the positions of large blocks (memories, IP cores, accelerators) on a chip canvas before the standard cells are placed. That arrangement sets how far signals travel, how crowded the routing becomes, and how the area is used, so it constrains the timing, power, and area of the finished design. The problem is hard for three reasons that compound. Macros may not overlap, the objectives pull against each other because short wires favor clustering while density and congestion favor spreading, and a single move can propagate through thousands of nets. Analytical placers such as RePlAce Cheng et al. [2019] and DREAMPlace Lin et al. [2019] define the current state of the art. A 2021 result argued that reinforcement learning agents building placements from scratch could match human experts Mirhoseini et al. [2021], but a later re-evaluation from UCSD found that a strong simulated annealing baseline and RePlAce still match or beat those agents on public benchmarks Cheng et al. [2024]. We ask a narrower question. Instead of building a placement from nothing, can a learned method improve one that an analytical placer has already produced?

Method We seed inference with a state-of-the-art analytical placement (RePlAce or DREAMPlace) and refine it with a learned local search adapted from the Lin-Kernighan family of heuristics for the traveling salesman problem Lin and Kernighan [1973], Helsgaun [2000]. Lin-Kernighan does not rebuild a tour. It improves an existing one through chains of dependent edge exchanges, where a single exchange may make the tour worse as long as the chain as a whole improves it. The same idea carries over to placement. Starting from the analytical solution, a learned policy selects a macro, proposes a small set of candidate positions, and either moves the macro or stops. Moving a macro onto a neighbor displaces that neighbor, which becomes the next macro in the chain, so one cascade reworks a connected group of macros at once. This lets the search make coordinated rearrangements the analytical placer cannot reach and escape the local optimum it would otherwise settle into.

Implementation Three learned components drive the loop. A graph neural network encodes the placement into per-macro embeddings using three rounds of message passing over the netlist, with macros as nodes and nets decomposed into weighted edges. A two-layer cost approximator reads the moved macro's embedding together with the proposed displacement and predicts the change in proxy cost, which lets the engine score thousands of candidate moves per second without calling the slow exact evaluator. A PPO actor-critic Schulman et al. [2017] scores the candidate positions and a stop action under one distribution, and decides where each chain starts, what it does at each step, and when it ends. The three components train together over several rounds. Each round collects labeled moves from deterministic chain cascades across all benchmarks on cloud CPUs, fits the cost approximator and encoder by regression on the true proxy change, trains the policy with PPO using the reduction in proxy cost measured after legalization as reward, and runs the full placer to feed new states into the next round.

Results We evaluate on the 17 ICCAD'04 IBM benchmarks used by the Partcl/HRT placement challenge Partcl and Hudson River Trading [2026], scored by the proxy cost $\text{HPWL} + 0.5\,(\text{congestion}) + 0.5\,(\text{density})$. The placer returns a legal, overlap-free result on all 17 designs within the one-hour budget. It improves on the simulated annealing baseline on every design and on RePlAce on 16 of the 17, the single exception being ibm01. Averaged over the suite it cuts the simulated annealing proxy cost by about 35 percent, from 2.125 to 1.371, and lands below the RePlAce target of 1.458.

Discussion The result shows that an analytical placement is not a fixed endpoint. A learned chain search can move past the local optimum the analytical placer settles into and reach layouts the placer cannot reach on its own. The learned components are deliberately small, which limits how closely the cost approximator fits the true proxy and how expressive the policy can be. Because the approximator is differentiable, its gradients also point to a continuous form of local refinement, using the cost model as an optimizer rather than only a move scorer.

Conclusion Seeded from RePlAce and refined by a learned Lin-Kernighan chain search, the system stays legal on all 17 benchmarks, beats simulated annealing on every design, and beats RePlAce on 16 of 17. To our knowledge, no published learning-based macro placer had previously surpassed RePlAce on the aggregate proxy cost for this benchmark suite.

Hybrid Reinforcement Learning for Chip Macro Placement

Daniel Hagenlocker
Stanford University
dhagenlo@stanford.edu

Jack Herrmann
Stanford University
jackherr@stanford.edu

Moritz Schreyögg
Stanford University
mosch@stanford.edu

Abstract

Macro placement is an early step in chip physical design that strongly affects the wirelength, congestion, and density of the finished layout. Analytical placers such as RePlAce and DREAMPlace are the strongest available methods, and recent work has questioned whether reinforcement learning placers that build a layout from scratch can match them. We take a different position. Rather than build a placement from nothing, we start from a state-of-the-art analytical placement and refine it with a learned local search modeled on the Lin-Kernighan heuristic for the traveling salesman problem. A learned policy drives chains of dependent macro moves in which an individual move may worsen the local geometry as long as the chain as a whole lowers the cost. The loop is built from three learned components: a graph neural network that encodes the placement, a small network that predicts the proxy-cost change of a candidate move, and a PPO actor-critic that selects the seed macro, the move at each step, and the stopping point. Trained jointly over several rounds across the 17 ICCAD'04 IBM benchmarks, the placer stays legal on all designs within a one-hour budget, beats a strong simulated annealing baseline on every design, and beats RePlAce on 16 of 17, cutting the simulated annealing proxy cost by about 35 percent on average.

1 Introduction

A chip begins as a blank rectangle. Before any wiring exists, the large blocks on the design, the memories, IP cores, and accelerators known as macros, have to be placed on that rectangle. Their arrangement decides how far signals travel, measured by wirelength, how crowded the wiring becomes, measured by congestion, and how evenly area is used, measured by density. Because the standard cells are placed into whatever space the macros leave, a poor macro arrangement creates dead space and notch regions that the downstream cell placer cannot use well, which degrades timing and power. Macro placement therefore sits early in the flow and constrains everything after it.

The problem is hard in a way that resists local fixes. Macros may not overlap, so the feasible region is irregular. The objectives conflict, since wirelength rewards pulling connected macros together while density and congestion reward spreading them out. The search space is enormous: a single benchmark such as ibm01 has 246 hard macros on a continuous canvas, with size ratios as large as $33\times$ and area utilization above 40 percent, which leaves little slack Partcl and Hudson River Trading [2026]. Moves interact through the netlist, so shifting one macro to shorten a wire usually pushes it into a neighbor, which then has to move as well. The effect compounds across hundreds of macros and thousands of nets.

Two lines of work define the landscape. Analytical placers model placement as a smooth optimization problem and solve a continuous relaxation; RePlAce and DREAMPlace are the leading examples and represent the current state of the art Cheng et al. [2019], Lin et al. [2019]. Separately, Mirhoseini et al. Mirhoseini et al. [2021] posed macro placement as a reinforcement learning problem, training

a policy to place macros one at a time onto an empty canvas, and reported placements comparable to or better than those of human experts on Google's accelerators. That claim was contested. A re-evaluation from UCSD built a stronger simulated annealing baseline and an open, reproducible flow, and found that simulated annealing and RePlAce match or beat the reinforcement learning approach on the public benchmarks studied Cheng et al. [2024]. On the ICCAD'04 suite in particular, the open-source reproduction of the reinforcement learning method did not win a single proxy-cost comparison against RePlAce TILOS AI Institute [2023].

Almost every method in both lines, learned or classical, shares one habit: it constructs a placement from the ground up. Simulated annealing anneals from a random start, the reinforcement learning agent grows a layout macro by macro on an empty canvas, and the analytical placer solves a continuous relaxation from a quadratic guess. We start from the observation that an analytical placer already produces a strong layout, and that the open question is whether a learned method can improve on it rather than replace it.

This report studies that question. We seed inference with a state-of-the-art analytical placement and refine it with a learned local search adapted from the Lin-Kernighan heuristic for the traveling salesman problem Lin and Kernighan [1973], Helsgaun [2000]. Lin-Kernighan improves a tour through chains of dependent edge exchanges rather than rebuilding it, and tolerates a worsening single step as long as the chain improves the tour overall. We apply the same structure to macro moves. The contributions are the following:

1. A placement refinement loop that treats a Lin-Kernighan style chain cascade as the action structure for reinforcement learning, seeded from RePlAce or DREAMPlace rather than from scratch.
2. Three learned components that make the loop run at scale: a message-passing graph encoder, a learned proxy-cost approximator that replaces the slow exact evaluator inside the inner loop, and a PPO actor-critic that controls the full shape of each chain.
3. An evaluation on the 17 ICCAD'04 IBM benchmarks showing legal placements on every design within budget, an improvement over simulated annealing on all 17, and an improvement over RePlAce on 16 of 17.

2 Related Work

Analytical placement The dominant academic placers cast placement as a continuous nonlinear optimization. ePlace and its successor RePlAce model the layout as an electrostatic system: cells become charges, the density penalty becomes electric potential energy derived from Poisson's equation, and the placement is spread out by following the gradient of that energy, solved with Nesterov's method Lu et al. [2015], Cheng et al. [2019]. RePlAce adds constraint-oriented local smoothing and dynamic step-size adaptation to improve quality and routability. DREAMPlace keeps the same formulation but recasts the optimization as a neural-network training problem on GPUs, reaching roughly an order of magnitude speedup over RePlAce without loss of quality Lin et al. [2019]. Both are strong, fast, and widely used, and we treat them as the seed for our refinement loop rather than as a competitor to be replaced.

Learned placement Mirhoseini et al. Mirhoseini et al. [2021] introduced a reinforcement learning approach in which an edge-based graph convolutional network feeds a policy that places macros sequentially onto a gridded canvas; standard cells are then placed by a force-directed method, and the reward is a weighted sum of approximate wirelength, congestion, and density given only at the end of the episode. The open-source reproduction is distributed as Circuit Training, later renamed AlphaChip. Subsequent learned placers explored other representations, such as MaskPlace, which learns a pixel-level visual representation and a dense reward to place macros on a high-resolution canvas Lai et al. [2022]. The reproducibility and the strength of the original claims were examined in detail by the UCSD study Cheng et al. [2024] and the associated MacroPlacement project TILOS AI Institute [2023], which reported that a stronger simulated annealing baseline and RePlAce are competitive with or better than the reinforcement learning method, and that the method benefits substantially from an initial placement supplied by a commercial tool. Our work differs from this line in its starting point: where these methods learn to construct a placement, we learn to refine one.

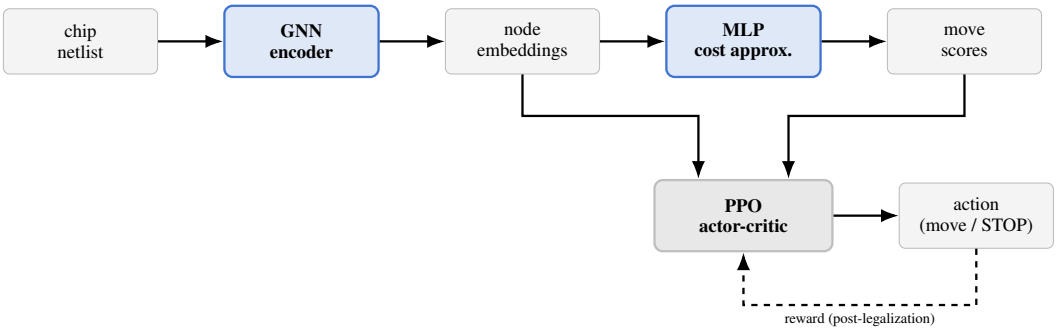

Figure 1: Overview of the refinement loop. From an analytical placement, a graph encoder produces per-macro embeddings. For the current macro, a learned cost approximator scores candidate displacements, and the PPO policy selects a move or a stop action under a single distribution. A move displaces a neighbor, which continues the chain. After a chain ends, the placement is legalized and the true proxy cost is measured to reward the policy.

Variable-depth local search The Lin-Kernighan heuristic is the basis of the strongest local-search solvers for the traveling salesman problem Lin and Kernighan [1973]. Rather than fix the number of edges it exchanges, it builds a chain of alternating edge removals and additions, accepts the chain only when the cumulative gain is positive, and lets a single step lose ground as long as the running gain stays favorable. Helsgaun's implementation, LKH, extends the basic move to deeper exchanges and uses a candidate set derived from minimum-spanning-tree sensitivity to direct and prune the search Helsgaun [2000]. The structure that matters for us is the chain: a sequence of dependent moves that reworks a connected part of the solution and is judged as a whole. We borrow that structure for macro placement, replacing the hand-designed gain criterion and candidate rules with learned components.

3 Method

3.1 Problem setup and objective

A design gives a set of macros with fixed shapes, a fixed canvas, and a netlist that connects macros to each other and to pre-placed standard-cell clusters. A placement assigns each macro a position. A placement is legal when no two macros overlap. Quality is measured by a proxy cost that combines half-perimeter wirelength (HPWL) with congestion and density penalties,

$$\text{proxy} \;=\; \text{HPWL} \;+\; 0.5\,(\text{congestion}) \;+\; 0.5\,(\text{density}),$$

computed by the public TILOS MacroPlacement evaluator TILOS AI Institute [2023]. Half-perimeter wirelength sums, over all nets, the half-perimeter of the bounding box of the net's pins, and is the standard proxy for routed wirelength. The two penalties stand in for routability and area pressure. Lower is better on all three terms, and the goal is a legal placement with the smallest proxy cost.

3.2 Placement as a chain cascade

We treat refinement as a sequential decision process over an existing placement. The state is the current placement, encoded into per-macro embeddings by the graph network of Section 3.3. The policy acts in chains. The procedure for a single chain is the following.

1. **Encode.** Encode the current placement into per-macro embeddings.
2. **Start a chain.** The policy selects the seed macro, the macro expected to begin the most improving chain.
3. **Propose.** Generate a small set of candidate positions for the current macro.
4. **Score and choose.** Score every candidate together with a stop action under one distribution, and select the highest.

5. **Cascade.** Apply the chosen move. The macro it displaces becomes the current macro, and the chain continues from step 3. The chain extends until the policy stops or no macro is displaced.
6. **Commit or revert.** Evaluate the states visited along the chain by wirelength, density, congestion, and an overlap penalty, commit the best state seen, and revert to the start of the chain if no visited state improved on it.
7. **Legalize.** Legalize the placement so that it is overlap-free.
8. **Continue.** Start the next chain. After long stagnation, perturb the placement to escape a stalled region.

The commit-or-revert rule in step 6 is the placement analogue of the Lin-Kernighan acceptance criterion: a step inside a chain is allowed to raise the overlap count or the local wirelength, and the chain is kept only if some state along it lowers the overall cost. The cascade in step 5 is what lets the search make coordinated rearrangements. Moving one macro to shorten its wires pushes a neighbor out of place, and following the displaced macro turns that side effect into the next decision rather than an unresolved overlap. Perturbation after stagnation plays the role of the restarts in iterated Lin-Kernighan, giving the search a way out of a region where no chain improves.

3.3 Learned components

Three learned components encode the placement, score candidate moves, and decide what to move and when to stop.

State encoder (GNN) The encoder is a three-layer message-passing graph network over the netlist. Macros are nodes, and each net is decomposed into a weighted clique so that a multi-pin net becomes a set of weighted pairwise edges. Each node carries features for position, size, movability, and per-macro mask signals that describe its placement context. A message-passing layer Gilmer et al. [2017] updates each node from its neighbors,

$$h_v^{(\ell+1)} \;=\; \phi\Big(h_v^{(\ell)},\; \textstyle\sum_{u\in\mathcal{N}(v)} w_{uv}\,\psi\Big(h_u^{(\ell)}, h_v^{(\ell)}\Big)\Big),$$

where w_{uv} is the clique-decomposition weight and ϕ and ψ are learned. The encoder outputs a per-macro embedding that both other components read. It is first trained on its own by regression onto the proxy cost, which gives a stable representation, and is then updated jointly with the policy.

Cost approximator (MLP) The exact evaluator is far too slow to call for every candidate move inside the inner loop. In its place, a two-layer MLP takes the embedding of the macro being moved together with the proposed displacement and predicts the resulting change in proxy cost. This learned approximator is what lets the engine weigh thousands of candidate moves per second. Because it is differentiable, its gradient with respect to the displacement also gives a direction for continuous local refinement, a use we leave to future work and discuss in Section 6.

Chain policy (PPO actor-critic) The policy reads the encoder's per-macro embeddings and decides the full shape of each chain: the seed macro, the action at each step, and the stopping point. The candidate moves and the stop action are scored together under a single action distribution, so stopping competes directly with continuing. The policy and its value baseline are trained with proximal policy optimization Schulman et al. [2017], which maximizes the clipped surrogate objective

$$L^{\mathrm{CLIP}}(\theta) \;=\; \mathbb{E}_t\Big[\min\Big(r_t(\theta)\,\hat{A}_t,\; \mathrm{clip}(r_t(\theta),\,1-\epsilon,\,1+\epsilon)\,\hat{A}_t\Big)\Big], \qquad r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)},$$

with advantages $\hat{A}_t$ estimated by generalized advantage estimation Schulman et al. [2016]. The reward is the reduction in true proxy cost measured after the placement is legalized, so the policy is rewarded for chains that improve the real objective and not only the surrogate the approximator predicts.

3.4 Training loop

The three components are trained together in an iterative loop, repeated over several rounds.

1. **Collect.** Sample candidate moves from deterministic Lin-Kernighan style cascades and label each with its true proxy-cost change from the exact evaluator. Collection runs in parallel across all 17 benchmarks on cloud CPUs.
2. **Approximate.** Fit the cost approximator, and the encoder beneath it, to those labels by regression.
3. **PPO.** Train the chain policy by running many chains and rewarding the reduction in proxy cost measured after legalization, warm-started from the previous round.
4. **Evaluate.** Run the full placer on every benchmark and feed the visited states back into the next round's collection set.

Pretraining the approximator before policy optimization keeps the early rounds stable, because the policy is scoring moves against a target that already correlates with the true cost. Warm-starting the policy each round and feeding back the states the placer actually visits keeps the training data close to the distribution the policy induces at inference.

4 Experimental Setup

Benchmarks We use the ICCAD'04 IBM-MSwPins mixed-size benchmarks Adya et al. [2004], the standard academic suite with established baselines. The suite has 18 designs, ibm01 through ibm18; ibm05 contains no macros and is excluded, which leaves 17. Each design has hundreds of hard macros with large size variation. ibm01, for example, has 246 hard macros ranging from 0.8 to $27\ \mu\mathrm{m}^2$, $7{,}269$ nets, 894 pre-placed standard-cell clusters, and about 43 percent area utilization Partcl and Hudson River Trading [2026]. These designs match the Tier 1 proxy-cost track of the Partcl/HRT Macro Placement Challenge, which is the setting our work targets.

Objective and evaluator All placements are scored by the proxy cost $\mathrm{HPWL} + 0.5\,(\text{congestion}) + 0.5\,(\text{density})$, computed by the TILOS MacroPlacement evaluator and used without modification TILOS AI Institute [2023], Partcl and Hudson River Trading [2026]. We report the average proxy cost over the 17 designs. A submission must be overlap-free, and the macro-placement run must finish within one hour per design.

Baselines and seed The two baselines are simulated annealing and RePlAce. The simulated annealing baseline follows the challenge configuration, which uses the go-with-the-winners metaheuristic and a multi-threaded implementation, the same strengthened baseline used in the UCSD study Cheng et al. [2024]. The RePlAce baseline is the OpenROAD implementation. Our method seeds inference with an analytical placement; the runs reported here seed from RePlAce, and the loop also accepts a DREAMPlace seed.

Candidate moves and chain length For the current macro the engine proposes about 16 candidate positions, drawn from three sources: the centroids of connected macros, which pull toward wirelength-reducing locations; local grid moves, which make small adjustments; and random canvas jumps, which provide exploration. A chain runs to a maximum depth of 8 or stops earlier when the policy emits the stop action or a step leaves no macro displaced.

Models and compute The encoder is a three-layer message-passing network and the cost approximator is a two-layer MLP, both deliberately small so that scoring stays fast inside the inner loop. Data collection runs in parallel across the benchmarks on cloud CPUs, and the per-benchmark evaluation harness measures the true proxy cost after legalization. Appendix B lists the remaining settings.

5 Results

5.1 Quantitative Evaluation

The placer returns a legal, overlap-free placement on all 17 designs within the one-hour budget. It improves on the simulated annealing baseline on 17 of 17 designs and on RePlAce on 16 of 17, the only exception being ibm01. Table 1 reports the average proxy cost over the suite. The refinement loop lowers the simulated annealing average from 2.125 to 1.371, a reduction of about 35 percent, and lands below the RePlAce target of 1.458. Because the loop seeds from RePlAce and either

commits an improving state or reverts to the seed, the per-design result tracks RePlAce closely on the hardest designs and pulls well below it where chains find room to improve.

Table 1: Average proxy cost over the 17 ICCAD'04 IBM benchmarks (lower is better). The right column is the change relative to the simulated annealing baseline.

Method	Avg. proxy cost	vs. SA
Simulated annealing	2.125	-
RePlAce (target)	1.458	-31%
Ours (LK chain + RL)	**1.371**	$\mathbf{-35\%}$

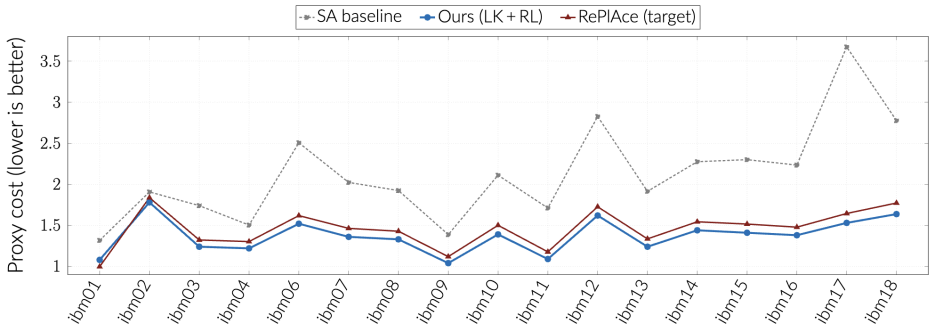

Figure 2: Per-design proxy cost for simulated annealing, our method, and the RePlAce target. Our curve sits below RePlAce on every design except ibm01 and far below simulated annealing throughout. All 17 placements are legal.

5.2 Qualitative Analysis

Figure 3 shows the effect of refinement on ibm13. The left panel is the legalized analytical seed and the right panel is the refined output, with arrows tracing how each macro moved. Both layouts are overlap-free, and the proxy cost falls over the run. The displacement pattern is not a uniform shift; clusters of connected macros move together, which is the signature of the chain cascade reworking connected groups rather than relocating macros one at a time.

A single chain makes the mechanism concrete. Figure 4 follows one chain on ibm01. A macro moves to shorten its wires and displaces a neighbor, which the chain then takes up, and the cascade continues from there. The overlap count rises by one in the middle of the chain and settles back as the wirelength drops, which is exactly the behavior the commit-or-revert rule is meant to allow: a temporary worsening that the chain pays back by the time it ends.

6 Discussion

The headline result is that an analytical placement is a starting point, not a fixed endpoint. RePlAce settles into a local optimum of its continuous relaxation, and a learned chain search built on Lin-Kernighan moves can leave that optimum and reach lower-cost legal layouts. Seeding from a strong placement and refining it sidesteps the part of the problem that learned-from-scratch placers struggle with, namely producing a globally sensible arrangement, and spends the learned capacity where it helps, on coordinated local rearrangement.

The approach has clear limits. The encoder and the cost approximator are small, so the approximator fits the true proxy only approximately; during development we observed that the approximator's rank ordering of candidate moves was weaker than its overall correlation (Appendix A), and because the chain selects moves by argmax, the quality of that ranking bounds the quality of the search. ibm01 is the one design where refinement did not improve on the seed, which suggests that for some layouts the reachable neighborhood under the current move set and chain depth does not contain a better legal arrangement. The evaluation here covers the Tier 1 proxy-cost track only; whether the gains

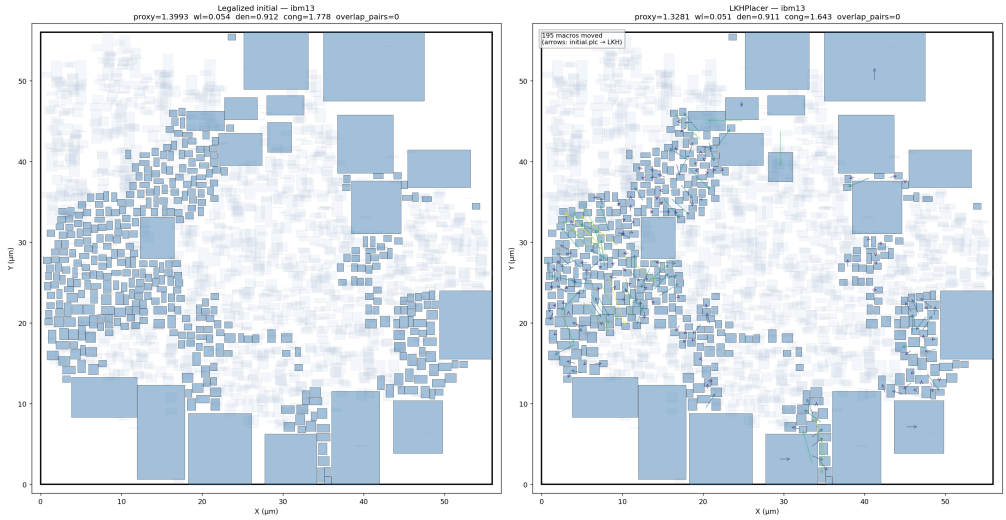

Figure 3: ibm13, legalized analytical seed (left) versus refined output (right). Arrows show macro displacement, with color proportional to move distance. Both placements are overlap-free and the proxy cost improves over the run. The values shown are from a development run and differ slightly from the final per-design results in Figure 2.

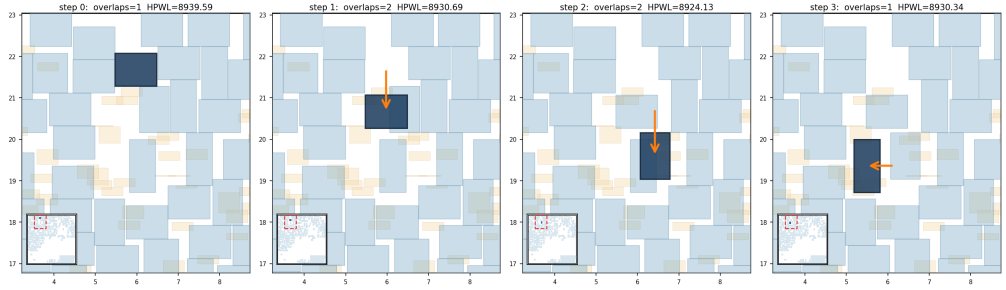

Figure 4: One chain on ibm01. The moved macro (with arrow) displaces a neighbor and the cascade continues. The overlap count rises by one mid-chain and settles back as wirelength drops.

carry through a full place-and-route flow on commercial designs, the challenge's Tier 2 metric, is not something we measured.

Three directions follow from these limits. Larger components, a higher-capacity cost approximator and a wider policy, give the model room to fit the proxy more closely and to rank candidate moves more reliably. Because the cost approximator is differentiable, its gradient gives a way to refine macro positions continuously, using the learned model as a local optimizer rather than only a scorer of discrete moves. On the search side, deeper chains and a larger candidate set per step let a single cascade rework a larger neighborhood and reach layouts the current short chains cannot.

7 Conclusion

We asked whether a learned placer could improve on what an analytical method already produces, and within this setup the answer is yes. Seeded from RePlAce and refined by a learned Lin-Kernighan chain search, the system stays legal on all 17 ICCAD'04 IBM benchmarks within budget, beats a strong simulated annealing baseline on every design, and beats RePlAce on 16 of 17, carrying the average proxy cost below the strongest classical method in the comparison. To our knowledge, no published learning-based macro placer had previously surpassed RePlAce on the aggregate proxy cost for this suite. The remaining headroom, larger learned components, gradient-based local refinement through the differentiable cost model, and deeper chains, points to where the margin can grow.

8 Team Contributions

The division of labor was largely unchanged:

- **Daniel Hagenlocker:** Wrote the learned components, including the cost approximator, the PPO chain policy, and the graph encoder, and integrated them into the inference loop.
- **Jack Herrmann:** Designed the Lin-Kernighan chain cascade and built the placer's inference path, including candidate generation and the commit-or-revert logic.
- **Moritz Schreyögg:** Built the training infrastructure, including the parallel cloud data collection, the iterative training loop, and the per-benchmark evaluation harness.

Changes from Proposal The proposal described a three-stage pipeline: a RePlAce analytical warm start, reinforcement learning refinement with a graph-plus-image encoder and a continuous action over $(\Delta x, \Delta y, \theta)$ edits, and cross-design pre-training. Two changes came out of implementation. The continuous-action edits mishandled the way a move creates overlap, since a free real-valued displacement gives no natural account of which neighbor must move next, so we moved the learning inside a Lin-Kernighan chain cascade with a discrete action over a candidate set plus a stop action. The exact proxy turned out to be too slow to query inside the chain, so we introduced the learned cost approximator to stand in for it. The encoder was simplified to the message-passing graph network used here. We also moved the seed from the challenge's supplied initial placement to a state-of-the-art analytical placement, since refining a layout already near the RePlAce baseline is the problem the challenge actually rewards.

References

Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang, and Lutong Wang. RePlAce: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730, 2019. doi: 10.1109/TCAD.2018.2859220.

Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z. Pan. DREAMPlace: Deep learning toolkit-enabled GPU acceleration for modern VLSI placement. In *Proceedings of the 56th Annual Design Automation Conference (DAC)*, pages 1–6, 2019. doi: 10.1145/3316781.3317803.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021. doi: 10.1038/s41586-021-03544-w.

Chung-Kuan Cheng, Andrew B. Kahng, Sayak Kundu, Yucheng Wang, and Zhiang Wang. An updated assessment of reinforcement learning for macro placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024. Early access; preprint arXiv:2302.11014.

Shen Lin and Brian W. Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, 1973. doi: 10.1287/opre.21.2.498.

Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. doi: 10.1016/S0377-2217(99)00284-2.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Partcl and Hudson River Trading. Partcl/HRT macro placement challenge 2026. `https://github.com/partcleda/macro-place-challenge-2026`, 2026. Accessed June 2026.

TILOS AI Institute. MacroPlacement: Benchmarks, evaluators, and reproducible results. `https://github.com/TILOS-AI-Institute/MacroPlacement`, 2023.

Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. ePlace: Electrostatics-based placement using fast Fourier transform and Nesterov's method. *ACM Transactions on Design Automation of Electronic Systems*, 20(2):1–34, 2015. doi: 10.1145/2699873.

Yao Lai, Yao Mu, and Ping Luo. MaskPlace: Fast chip placement via reinforced visual representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

Saurabh N. Adya, Shubhyant Chaturvedi, Jarrod A. Roy, David A. Papa, and Igor L. Markov. Unification of partitioning, placement and floorplanning. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 550–557, 2004.

A Additional Experiments

During training we tracked how well the cost approximator matched the exact evaluator. An early full run of four rounds across all benchmarks finished overlap-free with an average proxy cost between the simulated annealing and RePlAce baselines, with only small changes between rounds. At that stage the approximator reached a Pearson correlation of about 0.85 with the true proxy change but a Spearman rank correlation of only about 0.37. Since the chain selects moves by argmax over predicted cost change, the rank correlation is the quantity that governs move selection, and the gap between the two correlations identified the approximator's ranking quality as a bottleneck. This observation motivated reading the moved macro's learned embedding into the approximator rather than relying on hand-crafted features alone, and it remains the most direct target for the larger-capacity approximator proposed in Section 6.

B Implementation Details

The encoder is a three-layer message-passing network over the clique-decomposed netlist, with per-macro node features for position, size, movability, and mask signals. The cost approximator is a two-layer MLP over the moved macro's embedding concatenated with the proposed displacement. The policy and value networks read the same embeddings and share the encoder. Candidate generation produces about 16 positions per macro from connected-neighbor centroids, local grid moves, and random canvas jumps. Chains run to a maximum depth of 8. The policy is trained with PPO using a clipped surrogate objective and generalized advantage estimation, warm-started each round from the previous round's weights. The encoder and cost approximator are pretrained by regression on labeled moves before the first policy-optimization round. Data collection runs in parallel across the 17 benchmarks on cloud CPUs; final evaluation runs within a one-hour-per-design budget and uses the TILOS MacroPlacement evaluator to measure the post-legalization proxy cost.