Learning To Play the Game of Macro Placement with Deep Reinforcement Learning

Young-Joon Lee, Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang*, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak*, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc Le, James Laudon, Richard Ho, Roger Carpenter, Jeff Dean

Google

*Formerly Google
Outline

• Introduction
• Details of Work
• Comparison of Results with Previously Reported Work
• Conclusion
Motivation

In the past decade, systems and hardware have transformed ML. Now, it’s time for ML to transform systems and hardware.
We need significantly better systems and chips to keep up with the computational demands of AI

Implications of achieving performance on the computation, carbon emissions, and economic costs from deep learning on projections from polynomial models. *The Computational Limits of Deep Learning, Thompson et al., 2020*

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Error rate</th>
<th>Computation Required (Gflops)</th>
<th>Environmental Cost ($CO_2$)</th>
<th>Economic Cost ($)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Today: 11.5%</td>
<td>$10^{14}$</td>
<td>$10^6$</td>
<td>$10^6$</td>
<td></td>
</tr>
<tr>
<td>Target 1: 5%</td>
<td>$10^{10}$</td>
<td>$10^{10}$</td>
<td>$10^{11}$</td>
<td></td>
</tr>
<tr>
<td>Target 2: 1%</td>
<td>$10^{20}$</td>
<td>$10^{20}$</td>
<td>$10^{20}$</td>
<td></td>
</tr>
</tbody>
</table>

Since 2012, the amount of compute used in the largest AI training runs doubled every 3.4 months, *OpenAI, 2019*
Complexity of Chip Placement Problem

Chess

Go

Chip Placement

Number of states \( \sim 10^{123} \)

Number of states \( \sim 10^{360} \)

Number of states \( \sim 10^{9000} \)
Many problems in systems and chips are combinatorial optimization problems on graph data:

- **Compiler optimization:**
  - Input: XLA/HLO graph
  - Objective: Scheduling/fusion of ops
- **Chip placement:**
  - Input: A chip netlist graph
  - Objective: Placement on 2D or ND grids
- **Datacenter resource allocation:**
  - Input: A jobs workload graph
  - Objective: Placement on datacenter cells and racks
- ...
Advantages of Learning Based Approaches

ML models, unlike traditional approaches (such as branch and bound, hill climbing methods, or ILP solvers) can:

- Learn the underlying relationship between the context and target optimization metrics and leverage it to explore various optimization trade-offs
- “Gain experience” as they solve more instances of the problem and become “experts” over time
- Scale on distributed platforms and train billions of parameters
Chip Placement Problem

- A form of graph resource optimization
- Place the chip components to minimize the latency of computation, power consumption, chip area and cost, while adhering to constraints, such as congestion, cell utilization, heat profile, etc.
<table>
<thead>
<tr>
<th>Prior Approaches to Chip Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Partitioning-Based Methods</strong></td>
</tr>
<tr>
<td>(e.g. MinCut)</td>
</tr>
<tr>
<td><strong>Stochastic/Hill-Climbing Methods</strong></td>
</tr>
<tr>
<td>(e.g. Simulated Annealing)</td>
</tr>
<tr>
<td><strong>Analytic Solvers</strong></td>
</tr>
<tr>
<td>(e.g. RePlAce)</td>
</tr>
<tr>
<td>Prior Approaches to Chip Placement</td>
</tr>
<tr>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Partitioning-Based Methods (e.g. MinCut)</td>
</tr>
<tr>
<td>Analytic Solvers (e.g. RePlAce)</td>
</tr>
</tbody>
</table>
Chip Placement with Reinforcement Learning

**State:** Graph embedding of chip netlist, embedding of the current node, and the canvas.

**Action:** Placing the current node onto a grid cell.

**Reward:** A weighted average of total wirelength, density, and congestion
Our Objective Function

\[ J(\theta, G) = \frac{1}{K} \sum_{g \sim G} E_{g,p \sim \pi_\theta}[R_{p,g}] \]

- Reward corresponding to placement \( p \) of netlist (graph) \( g \)
- \( J(\theta, G) \) is the objective function
- \( K \) is size of training set
- RL policy parameterized by \( \theta \)

\[ R_{p,g} = -Wirelength(p, g) \]
\[-\lambda \ Congestion(p, g) - \gamma \ Density(p, g) \]
A Hybrid Approach to Placement Optimization

Chip canvas

RL Agent Places Macros One at a Time

Force-Directed Method Places Standard Cells

$\mathbf{r}_0 = 0$

$\mathbf{a}_0$  $\mathbf{s}_1$  $\mathbf{a}_1$  $\mathbf{s}_2$  $\mathbf{r}_2 = 0$

$\mathbf{a}_{t-1}$  $\mathbf{s}_T$

$\mathbf{r}_T = -\text{HPWL} - c \ast \text{Congestion}$
Results on a TPU-v4 Block

White area are macros and the green area is composed of standard cell clusters. Our method finds smoother, rounder macro placements to reduce the wirelength.

**Human Expert**

- Time taken: ~6-8 weeks
- Total wirelength: 57.07m
- Route DRC violations: 1766

**ML Placer**

- Time taken: 24 hours
- Total wirelength: 55.42m (-2.9% shorter)
- Route DRC violations: 1789 (+23 - negligible difference)

DRC: Design Rule Checking
Moving Towards Generalized Placements

**Before:** Training from scratch for each netlist

**Now:** Pre-training the policy and fine-tuning on new netlists

---

**Training**

- New Netlist
- Placer Policy
- Placements
- 10,000s of iterations
- Final result

**Inference**

- New Netlist
- Pre-Trained Policy
- Placements
- 100s of iterations
- Final result

---

**Training**

- Netlist 1
- Netlist 2
- Netlist N
- 10,000s of iterations

**Inference**

- New Netlist
- Pre-Trained Policy
- Placements
- 100s of iterations
- Final result
First Attempts at Generalization

- Using the previous RL policy architecture, we trained it on multiple chips and tested it on new unseen chips. -> Didn’t work!

- Freezing different layers of the RL policy and then testing it on new unseen chips -> Didn’t work either!

- What did work? Leveraging supervised learning to find the right architecture!
Achieving Generalization by Training Accurate Reward Predictors

- We observed that a value network trained only on placements generated by a single policy is unable to accurately predict the quality of placements generated by another policy, limiting the ability of the policy network to generalize.

- To decompose the problem, we trained models capable of accurately predicting reward from off-policy data.
Compiling a Dataset of Chip Placements

- To train a more accurate predictor, we generated a dataset of 10k placements for 5 blocks.
- Each placement was labeled with their wirelength and congestion, which were drawn from vanilla RL policies.
Reward Model Architecture and Features

Input Features

- Node Feature (x, y, w, h, type*)
- Graph (macro, standard cells, clusters)
- Netlist Metadata (Total number of wires and macros, name of netlist)

Graph Conv

Predictions
- fc
- fc

Wirelength
- fc

Congestion

Different colors depict different part of the tensor.

*Node type: One-hot category (Hard macro, soft macro)
Reward Model Architecture and Features

Input Features
- Node Features (x, y, w, h, type)
- Graph (macro, standard cells)
- Netlist Metadata (Total number of wires and macros, name of netlist)

Graph Conv

while Not converged do
    Update edge: $e_{ij} = fc_1(concat(fc_0(v_i)|fc_0(v_j)|w_{ij}^e))$
    Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$
end

Predictions
- Wirelength
- Congestion

*Node type: One-hot category (Hard macro, soft macro)
Edge-based Graph Convolution: Node Embeddings
Edge-based Graph Convolution: Edge Embedding
Edge-based Graph Convolution: Edge Embedding
Edge-based Graph Convolution: Propagate
Edge-based Graph Convolution: Repeat
Final Step: Get Graph Embedding
Reward Model Architecture and Features

Input Features

- Node Features: $(x, y, w, h, \text{type})$
- Graph (macro, standard cell)
- Netlist Metadata: (Total number of wires and macros, name of netlist)

Graph Conv

Predictions

- fc
- fc

Wirelength
Congestion

*Node type: One-hot category (Hard macro, soft macro)*
Label Prediction Results on Test Chips
Policy/Value Model Architecture
Experimental Setup

- For pre-training, we used the same number of workers as blocks in the training dataset
  - For example, for the largest training set with 20 blocks, we pre-trained with 20 GPU workers
- The pre-training runtime was 48 hours
- For fine-tuning results, our method ran on 16 GPU workers for up to 6 hours, but the runtime was often significantly lower due to early stopping
- For both pre-training and fine-tuning, a worker consists of an Nvidia Volta GPU and 10 CPUs each with 2GB of RAM
- For zero-shot mode (applying a pre-trained policy to a new netlist with no fine-tuning), we can generate a placement in less than a second on a single GPU
Ariane (RISC-V) Placement Visualization

Training policy from scratch

Finetuning a pre-trained policy

The animation shows the macro placements as the training progresses. Each square shows the center of a macro.

Ariane is an open-source RISC-V processor. See: https://github.com/pulp-platform/ariane
Convergence Curve: Training from Scratch vs. Finetuning

![Graph showing convergence curve]

- Train Policy From Scratch
- Finetune a Pre-Trained Policy
Generalization Results

The zero shot and fine-tuned policies were pre-trained on other blocks for ~24 hrs.
Effects of Training Set Size on Convergence
Humans Were Inspired by Our ML Placer!

<table>
<thead>
<tr>
<th></th>
<th>Wirelength</th>
<th>Congestion_ Horizontal</th>
<th>Congestion_ Vertical</th>
<th>Worst Negative Slack</th>
<th>Total Negative Slack</th>
<th>Number of Violating Endpoints</th>
<th>Area Buf Inv</th>
<th>Area Total</th>
<th>EDA Tool Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML Placer</td>
<td>2.45E+07</td>
<td>0.00</td>
<td>0.01</td>
<td>-0.143</td>
<td>-11.45</td>
<td>508</td>
<td>5,619</td>
<td>323,964</td>
<td>6.6hr</td>
</tr>
<tr>
<td>Inspired Manual</td>
<td>2.44E+07</td>
<td>0.00</td>
<td>0.01</td>
<td>-0.114</td>
<td>-16.29</td>
<td>901</td>
<td>5,634</td>
<td>324,094</td>
<td>7.1hr</td>
</tr>
</tbody>
</table>
Comparisons with State-of-the-Art Academic Baselines

Using GRL-only, we outperform the prior state-of-the-art RePlAce on 5 TPU-v4 blocks.

<table>
<thead>
<tr>
<th>Name</th>
<th>Method</th>
<th>Timing</th>
<th>Area</th>
<th>Power</th>
<th>Wirelength</th>
<th>Congestion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>WNS (ps)</td>
<td>TNS (ns)</td>
<td>Total (μm²)</td>
<td>Total (W)</td>
<td>(m)</td>
</tr>
<tr>
<td>Block 1</td>
<td>RePlAce</td>
<td>374</td>
<td>233.7</td>
<td>1693139</td>
<td>3.70</td>
<td>52.14</td>
</tr>
<tr>
<td></td>
<td>Manual</td>
<td>136</td>
<td>47.6</td>
<td>1680790</td>
<td>3.74</td>
<td>51.12</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>84</td>
<td>23.3</td>
<td>1681767</td>
<td>3.59</td>
<td>51.29</td>
</tr>
<tr>
<td>Block 2</td>
<td>RePlAce</td>
<td>97</td>
<td>6.6</td>
<td>785655</td>
<td>3.52</td>
<td>61.07</td>
</tr>
<tr>
<td></td>
<td>Manual</td>
<td>75</td>
<td>98.1</td>
<td>830470</td>
<td>3.56</td>
<td>62.92</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>59</td>
<td>170</td>
<td>694757</td>
<td>3.13</td>
<td>59.11</td>
</tr>
<tr>
<td>Block 3</td>
<td>RePlAce</td>
<td>193</td>
<td>3.9</td>
<td>867390</td>
<td>1.36</td>
<td>18.84</td>
</tr>
<tr>
<td></td>
<td>Manual</td>
<td>18</td>
<td>0.2</td>
<td>869779</td>
<td>1.42</td>
<td>20.74</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>11</td>
<td>2.2</td>
<td>868101</td>
<td>1.38</td>
<td>20.80</td>
</tr>
<tr>
<td>Block 4</td>
<td>RePlAce</td>
<td>58</td>
<td>11.2</td>
<td>944211</td>
<td>2.21</td>
<td>27.37</td>
</tr>
<tr>
<td></td>
<td>Manual</td>
<td>58</td>
<td>17.9</td>
<td>947766</td>
<td>2.17</td>
<td>29.16</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>52</td>
<td>0.7</td>
<td>942867</td>
<td>2.21</td>
<td>28.50</td>
</tr>
<tr>
<td>Block 5</td>
<td>RePlAce</td>
<td>156</td>
<td>254.6</td>
<td>1477283</td>
<td>3.24</td>
<td>31.83</td>
</tr>
<tr>
<td></td>
<td>Manual</td>
<td>107</td>
<td>97.2</td>
<td>1480881</td>
<td>3.23</td>
<td>37.99</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>68</td>
<td>141.0</td>
<td>1472302</td>
<td>3.28</td>
<td>36.59</td>
</tr>
</tbody>
</table>

- We freeze the macro placements generated by each method and report the place opt results by a commercial EDA.
Comparisons with Commercial Auto Macro Placers

**Commercial Auto Macro Placers:**

**EDA Vendor A**
- Legacy auto macro placer: WL Driven

**EDA Vendor B**

### TPU-v5 Block Compositions

| Composition of TPU-v5 (Canvas Saturation): | 62% Low Sat; 24% Med Sat; 14% High Sat. | Composition of TPU-v5 (Hard Macro Count): | 33.3% Low Cnt; 33.3% Med Cnt; 33.3% High Cnt. |

<table>
<thead>
<tr>
<th>Blocks</th>
<th>Low Macro Count</th>
<th>Medium Macro Count</th>
<th>High Macro Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Sat.</td>
<td>(7) 33.3% Low Sat. - Low Count</td>
<td>(4) 19.0% Low Sat. - Med Count</td>
<td>(2) 9.5% Low Sat. - High Count</td>
</tr>
<tr>
<td>Med Sat.</td>
<td>(0) 0.0% Med Sat. - Low Count</td>
<td>(2) 9.5% Med Sat. - Med Count</td>
<td>(3) 14.3% Med Sat. - High Count</td>
</tr>
<tr>
<td>High Sat.</td>
<td>(0) 0.0% High Sat. - Low Count</td>
<td>(1) 4.8% High Sat. - Med Count</td>
<td>(2) 9.5% High Sat. - High Count</td>
</tr>
<tr>
<td>Blocks</td>
<td>7 (33%)</td>
<td>7 (33%)</td>
<td>7 (33%)</td>
</tr>
</tbody>
</table>
Auto Macro Placer Comparison Summary

<table>
<thead>
<tr>
<th></th>
<th>ML Placer is superior</th>
<th>ML Placer is equal</th>
<th>ML Placer is inferior</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDA-A</td>
<td>15</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>EDA-B</td>
<td>13</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Manual</td>
<td>11</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>ML Placer</th>
<th>Manual</th>
<th>EDA-A</th>
<th>EDA-B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best</td>
<td>13</td>
<td>9</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>Not best</td>
<td>7</td>
<td>11</td>
<td>15</td>
<td>14</td>
</tr>
</tbody>
</table>
# QoR Comparison for A TPU-v5 Block

<table>
<thead>
<tr>
<th></th>
<th>Wirelength</th>
<th>Cong_H</th>
<th>Cong_V</th>
<th>WNS</th>
<th>WNS Int</th>
<th>WNS Ext</th>
<th>TNS</th>
<th>TNS Int</th>
<th>TNS Ext</th>
<th>NVE</th>
<th>NVE Int</th>
<th>NVE Ext</th>
<th>Area Bufinv</th>
<th>Area Total</th>
<th>EDA Tool Run Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ML Placer</strong></td>
<td>7.06E+06</td>
<td>0.01</td>
<td>0.02</td>
<td>-0.094</td>
<td>-0.094</td>
<td>-0.065</td>
<td>-58.5</td>
<td>-55.3</td>
<td>-3.3</td>
<td>2188</td>
<td>1970</td>
<td>218</td>
<td>3.86E+03</td>
<td>2.52E+05</td>
<td>5.2hr</td>
</tr>
<tr>
<td><strong>Manual</strong></td>
<td>7.56E+06</td>
<td>0.01</td>
<td>0.02</td>
<td>-0.101</td>
<td>-0.101</td>
<td>-0.045</td>
<td>-57.8</td>
<td>-56.3</td>
<td>-1.5</td>
<td>2019</td>
<td>1872</td>
<td>147</td>
<td>3.75E+03</td>
<td>2.52E+05</td>
<td>5.7hr</td>
</tr>
<tr>
<td><strong>EDA-A Mixed Size Placer</strong></td>
<td>6.52E+06</td>
<td>0.00</td>
<td>0.01</td>
<td>-0.135</td>
<td>-0.135</td>
<td>-0.021</td>
<td>-93.7</td>
<td>-93.2</td>
<td>-0.5</td>
<td>2135</td>
<td>2041</td>
<td>94</td>
<td>4.12E+03</td>
<td>2.52E+05</td>
<td>6.6hr</td>
</tr>
<tr>
<td><strong>EDA-B Advanced Macro Placer</strong></td>
<td>7.28E+06</td>
<td>0.01</td>
<td>0.02</td>
<td>-0.099</td>
<td>-0.099</td>
<td>-0.021</td>
<td>-79.7</td>
<td>-77.7</td>
<td>-1.9</td>
<td>2530</td>
<td>2275</td>
<td>255</td>
<td>3.82E+03</td>
<td>2.61E+05</td>
<td>6.0hr</td>
</tr>
</tbody>
</table>
Key Takeaways

• Novel deep reinforcement learning approach that generates superhuman macro placements in several hours (decreasing)
• Outperforms academic state-of-the-art and strongest commercial auto macro placers
• Used for multiple blocks on next generation TPU project!
• User feedback was positive, considered useful indeed
• Unlocks potential for faster chip design process
Thank You

Questions?