A graph placement methodology for fast chip design

Nature, June 2021

Azalia Mirhoseini, Anna Goldie (equal contributions)
…+ 18 more co-authors + Jeff Dean

Presented by Igor L. Markov

Context: why RL for chip design?

Why chip design @ Google?

- Efficient data centers
- Cloud computing services
- Cutting-edge AI accelerators → Google TPU design
- Big bet: use AI to design better TPUs
  - ML learns from data/experience
  - Consider TPU blocks as separate entities
- Solve hard combinatorial optimization problems, claim AI as a general tool

Why Reinforcement Learning?

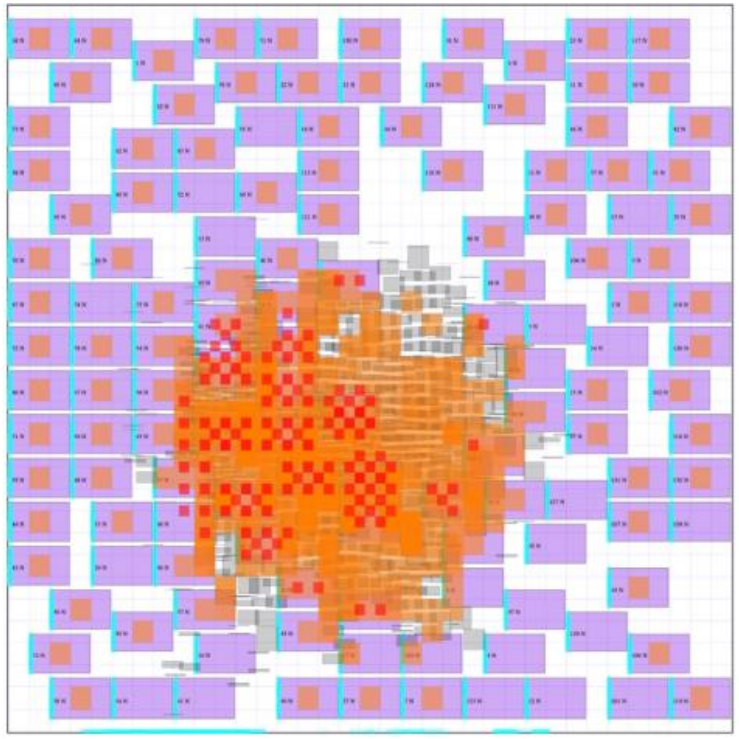
- Great success at games (Go, Chess)
  - Self-play
- "Learns by doing" how to optimize *delayed* objectives
- In general, RL can learn practical reward functions from feedback
- "Gamify" chip design
  - Solve combinatorial optimization problems by optimizing the ordering of smaller optimizations
  - RL handles sequences well

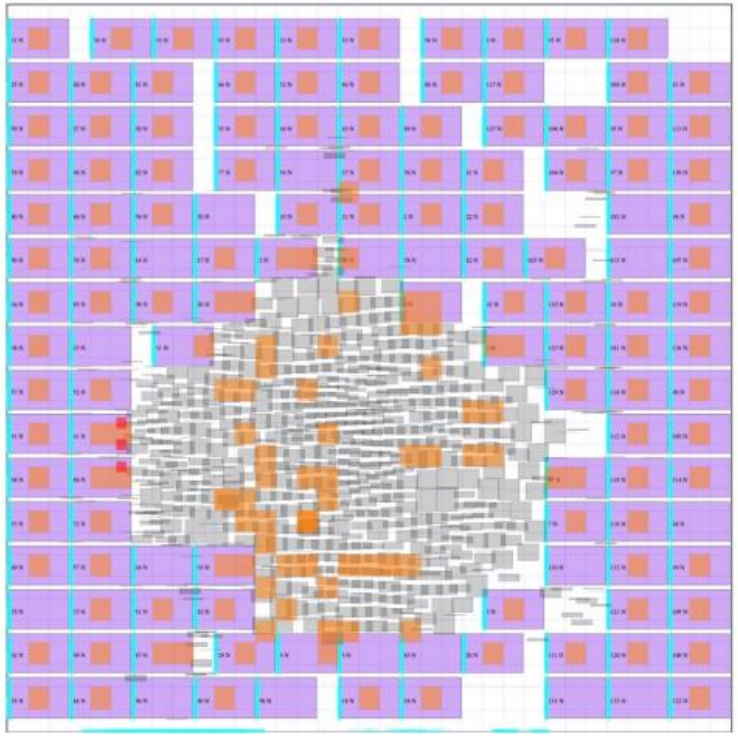
Optimization goals for chip design

- Traditional metrics: PPA (power, performance + chip/block area)
  - TPU blocks do not allow changing their sizes
  - Block chip area is not changed during fixed-canvas macro placement
- Chip metrics in the Nature paper
  - Circuit power (dynamic power dominates)
  - Total cell area: not altered by macro placement or standard-cell placement
  - Circuit delay metrics
    - Total negative slack
    - Worst negative slack

Challenges of mixed-size placement

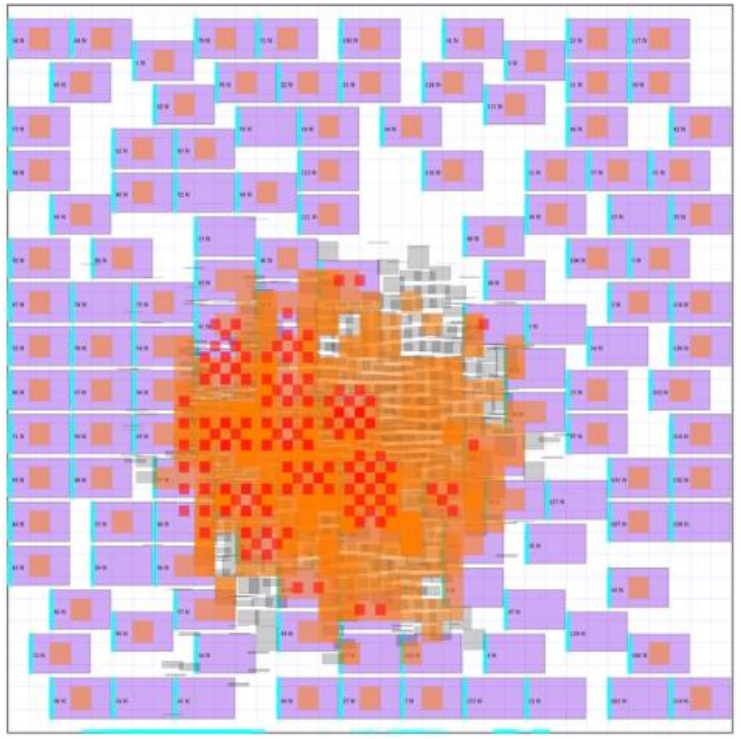
- Macros compete for area but also affect interconnects and congestion
- **Poor macro placement → to routing congestion**
  - **In a fixed canvas, high congestion → routing failures**
  - **Manually redoing macro placement → costly trial and error**
- Small components (standard cells) are too numerous to handle the same way as large components
- If we neglect small components, we neglect a lot of interconnect

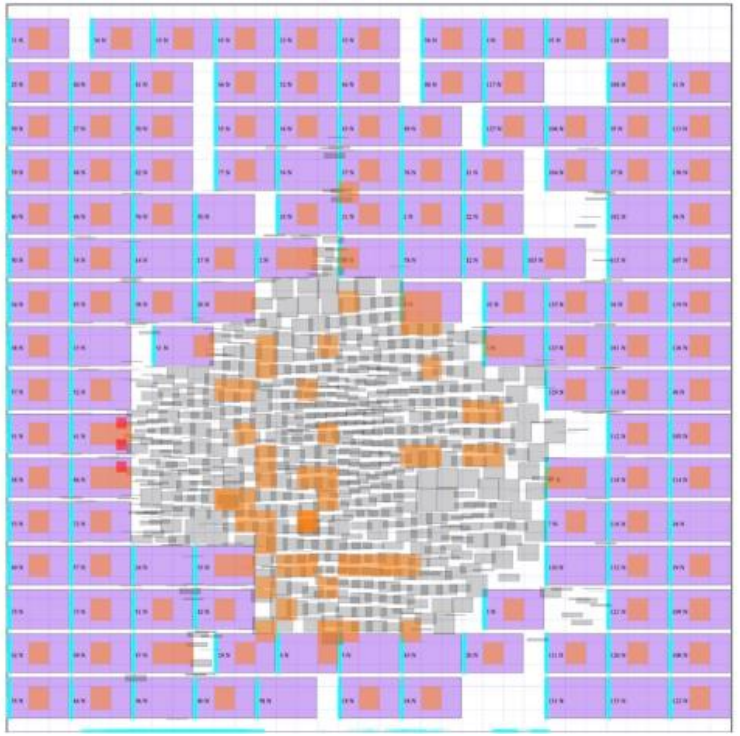




Physical design flow (used for RL and a baseline)

- Estimate routing congestion during placement
  - Perform fast global routing on a grid
  - Compare routing supply to routing demand
- Proposed design flow:
  - netlist (physical) synthesis using Synopsys
  - **place macros**
  - place standard cells w Cadence
  - route wires w Cadence
  - optimize circuit timing etc
- Advanced technology nodes





Proposed approach to macro placement

- Layout gridding
  - Divide chip canvas into a rectangular grid (say, 40x40).
  - This helps evaluate peak cell density and routing congestion.
- Netlist clustering with the hMetis package
  - Cluster the netlist (soft and hard clusters) and place entire clusters *on the grid*
- Proxy objective function combines: **HPWL**, component density and routing congestion
  - routing congestion adds up 10% worst grid cells
  - no circuit delay evaluation is used

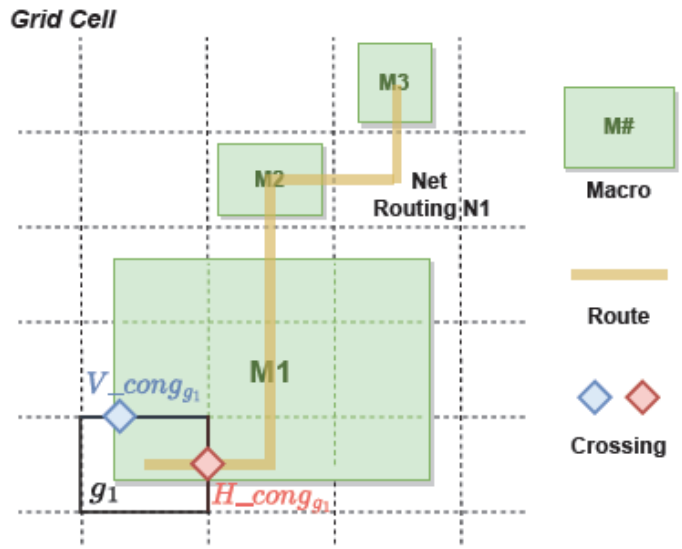
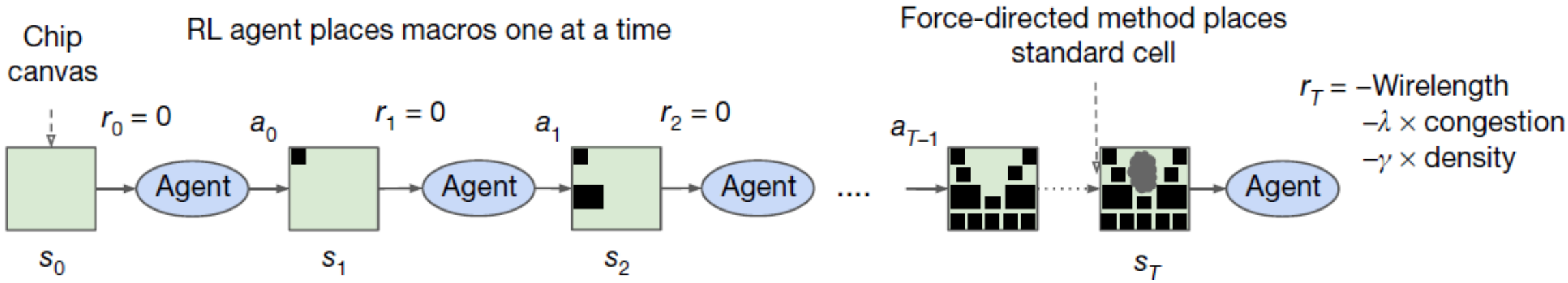


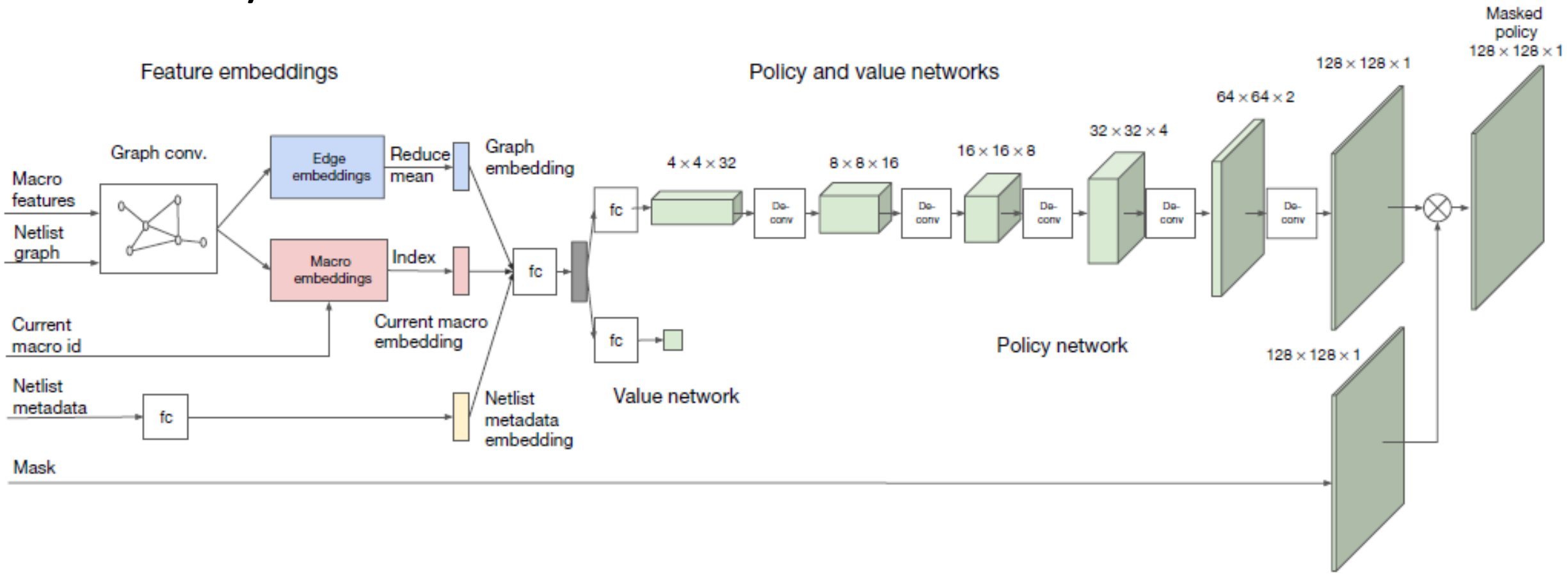
Figure 1: Illustration for congestion cost computation.

Placing macros with RL

- **Iterate the placement of large and small components, to keep optimization consistent**
  - Place one macro at a time to a grid location that minimizes interconnect
  - Update a tentative placement of standard cells using fast quadratic placement
- Sequential process
  - place macros one at a time
  - assign a reward at the end in the amount of negative proxy function value
- **Train a DNN action policy and value network to optimize the proxy as a delayed objective**



Policy and value network



Training the RL policy

- The RL policy **trains** on each new example through many rounds of sequential placement of macros
- Also pre-train the RL policy on multiple designs
  - RL policy was pre-trained on 20 TPU blocks, then used and evaluated on 5 TPU blocks
  - Pre-training runtime was not given
  - Claim 1: pre-training improves runtimes
  - Claim 2: pre-training improves quality (but no chip metric improvement shown)

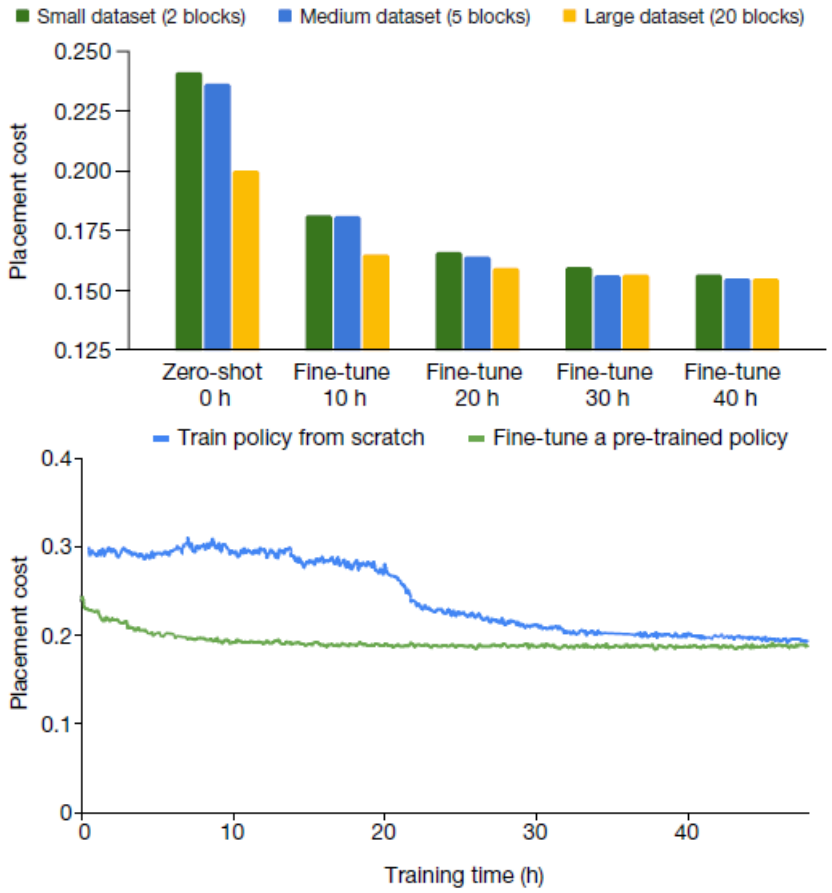


Fig. 4 | Convergence plots on Ariane RISC-V CPU. Placement cost of training a policy network from scratch versus fine-tuning a pre-trained policy network for a block of Ariane RISC-V CPU.

Baselines for comparisons

Simulated annealing

- Applied to the same type of input as RL
- Can use the same proxy
- Macro placement is initialized at random
- Action types for macros: “swap,” “shift,” and “mirror”
- Runs in parallel for different parameter configurations

RePlAce from UCSD

- Applied to the same type of input as RL
- Optimizes a different objective function
- Analytical placer with strong results on common benchmarks
- Does not use RL
- Available in open source

Commercial EDA tools

- Applied to entire netlist
- Cannot be compared to in a published paper(*)
- However, claims of using the proposed method on production TPUs imply that prior EDA tools were no better

Empirical validation

- “For confidentiality reasons, we cannot disclose the details of these blocks, but each contains up to a few hundred macros and millions of standard cells.”
- Manual baseline was generated by the TPU design team
- “We ran all of the evaluations of RePlAce and our method ourselves, but we relied on the TPU physical design team to share metrics for their best-performing manual placements, and they may have evaluated with a slightly different EDA version”.
- RL uses massive parallel resources, up to 6 hours of runtime
- RePlAce runs in under an hour on a single CPU, but some placements are unusable
- **“Our method outperforms or matches manual placements in area, power and wirelength. … our end-to-end learning-based approach takes far less time to meet design criteria.”**
- **“Our method has been used in production to design the next generation of Google TPU”**

Table 1 | Comparisons against baselines

Name	Method	Timing		Total area (μm^2)	Total power (W)	Wirelength (m)	Congestion	
		WNS (ps)	TNS (ns)				H (%)	V (%)
Block 1	RePlAce	374	233.7	1,693,139	3.70	52.14	1.82	0.06
	Manual	136	47.6	1,680,790	3.74	51.12	0.13	0.03
	Our method	84	23.3	1,681,767	3.59	51.29	0.34	0.03
Block 2	RePlAce	97	6.6	785,655	3.52	61.07	1.58	0.06
	Manual	75	98.1	830,470	3.56	62.92	0.23	0.04
	Our method	59	170	694,757	3.13	59.11	0.45	0.03
Block 3	RePlAce	193	3.9	867,390	1.36	18.84	0.19	0.05
	Manual	18	0.2	869,779	1.42	20.74	0.22	0.07
	Our method	11	2.2	868,101	1.38	20.80	0.04	0.04
Block 4	RePlAce	58	11.2	944,211	2.21	27.37	0.03	0.03
	Manual	58	17.9	947,766	2.17	29.16	0.00	0.01
	Our method	52	0.7	942,867	2.21	28.50	0.03	0.02
Block 5	RePlAce	156	254.6	1,477,283	3.24	31.83	0.04	0.03
	Manual	107	97.2	1,480,881	3.23	37.99	0.00	0.01
	Ours	68	141.0	1,472,302	3.28	36.59	0.01	0.03

Here, we compare our method with the state-of-the-art method (RePlAce[14]) and with manual placements using an industry-standard EDA tool. For all metrics in this table, lower is better.
H, horizontal; V, vertical.