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

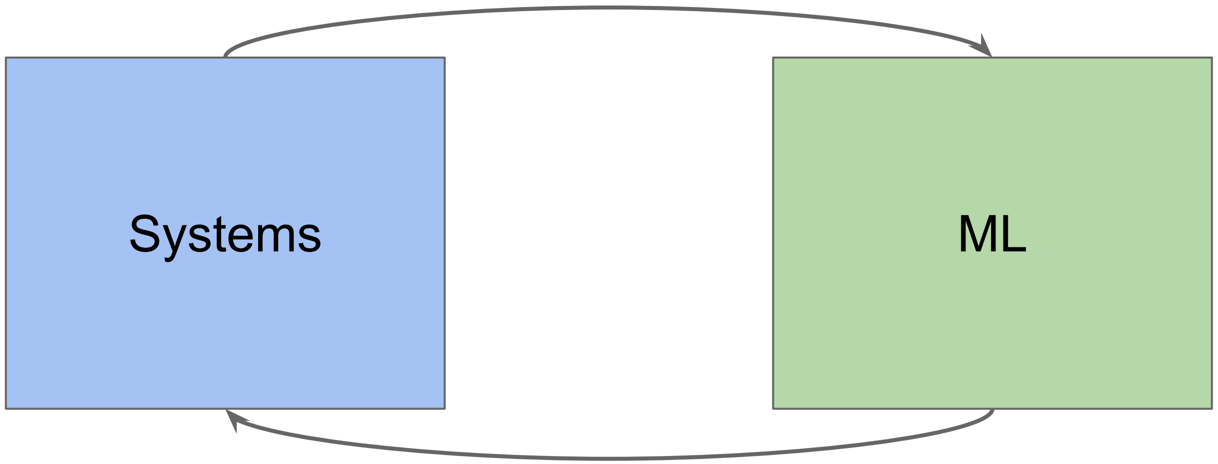
*Formerly Google

Outline

- Introduction
- Details of Work
- Comparison of Results with Previously Reported Work
- Conclusion

Motivation

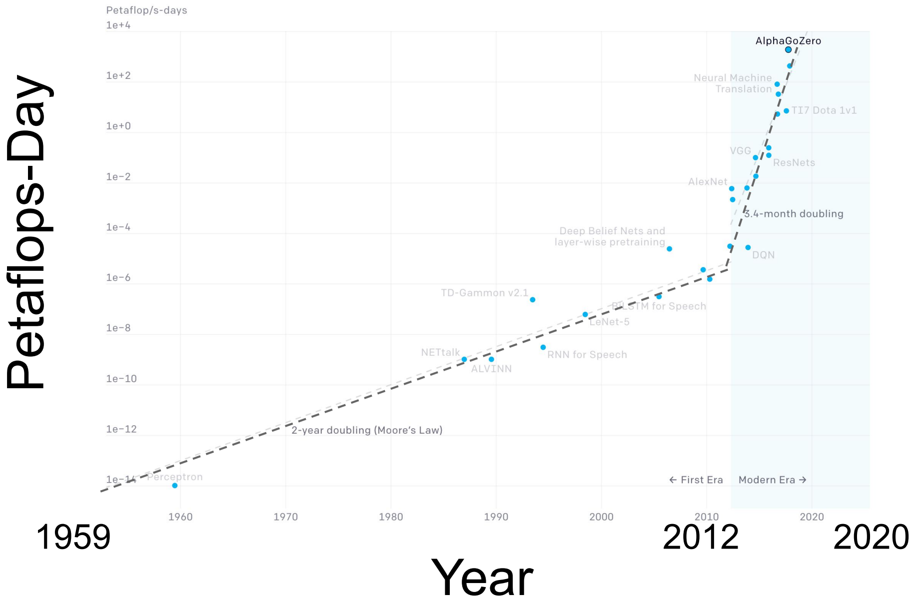
In the past decade, systems and hardware have transformed ML.
Now, it's time for ML to transform systems and hardware.



We need significantly better systems and chips to keep up with the computational demands of AI

Benchmark	Error rate	Polynomial		
		Computation Required (Gflops)	Environmental Cost (CO_2)	Economic Cost ($)
ImageNet	Today: 11.5%	10^{14}	10^{6}	10^{6}
	Target 1: 5%	10^{19}	10^{10}	10^{11}
	Target 2: 1%	10^{28}	10^{20}	10^{20}

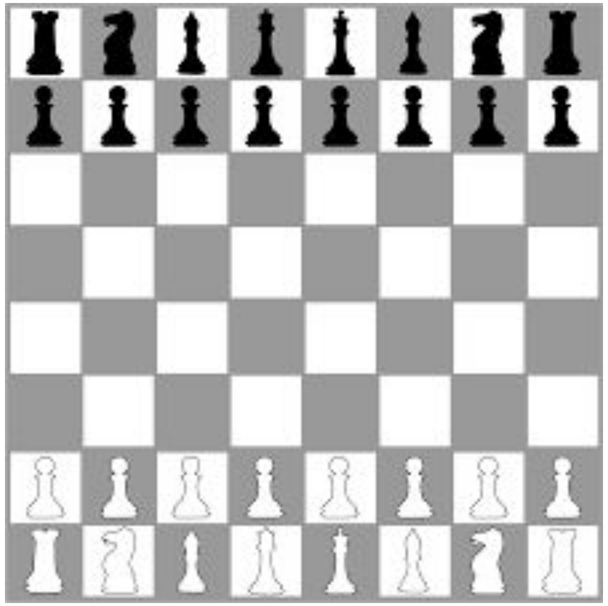
Implications of achieving performance on the computation, carbon emissions, and economic costs from deep learning on projections from polynomial models. *The Computational Limits of Deep Learning, Thompson et al., 2020*



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

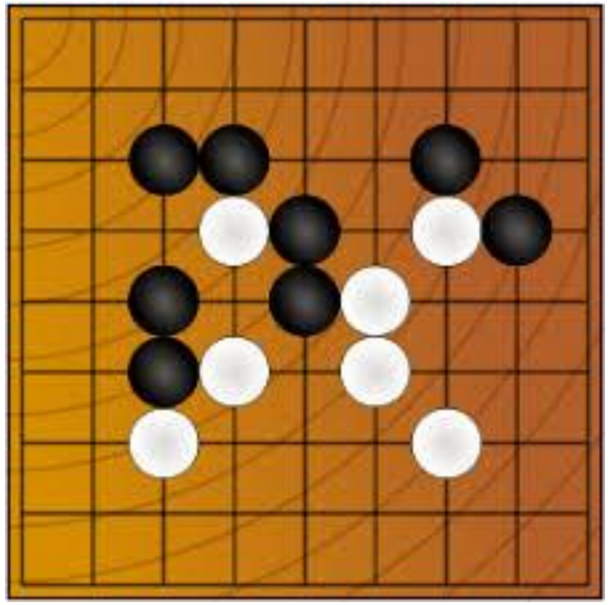
Complexity of Chip Placement Problem

Chess



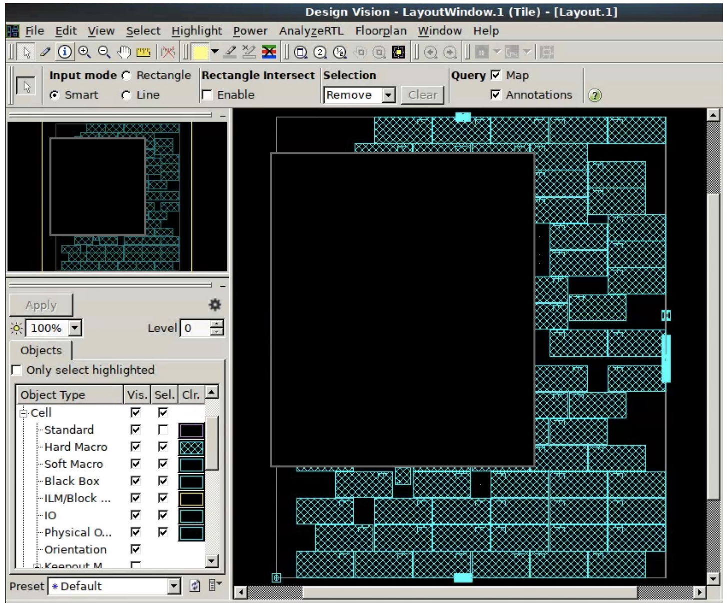
Number of states ~ 10^{123}

Go



Number of states ~ 10^{360}

Chip Placement

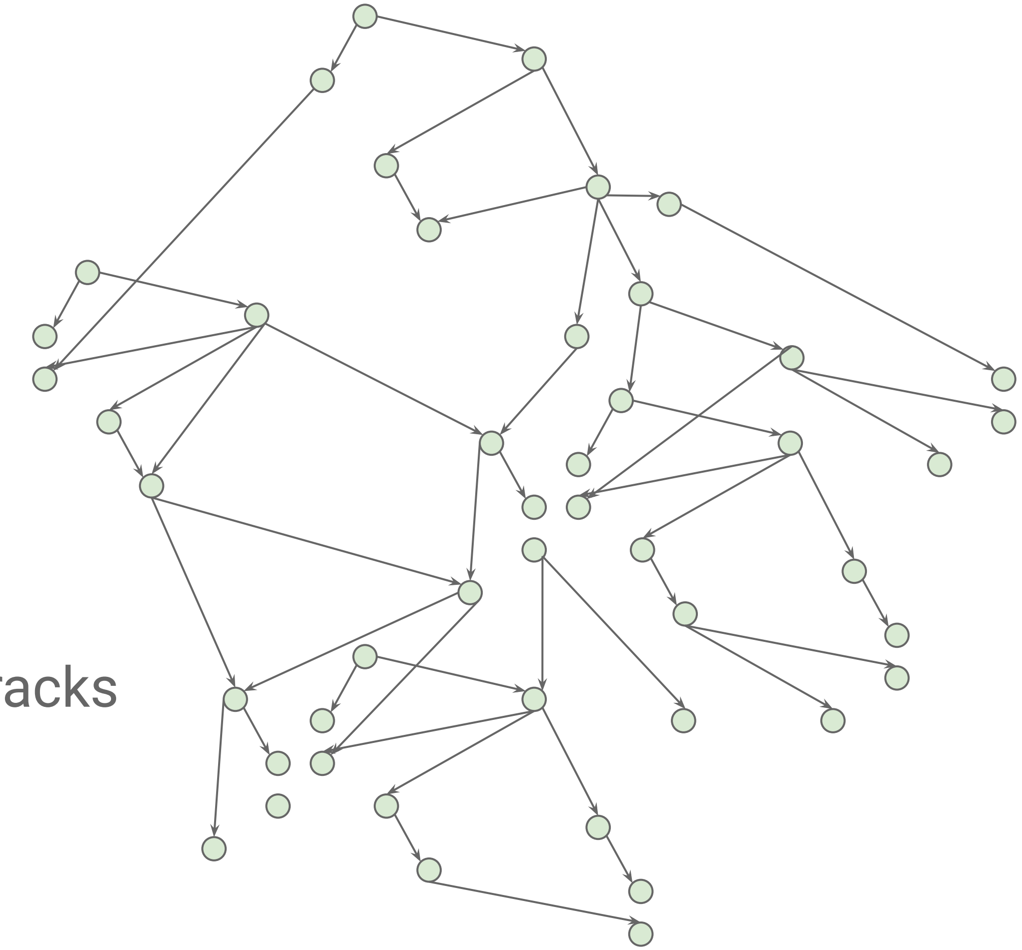


Number of states ~ 10^{9000}

Combinatorial Optimization on Graph Data

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
- ...



Google

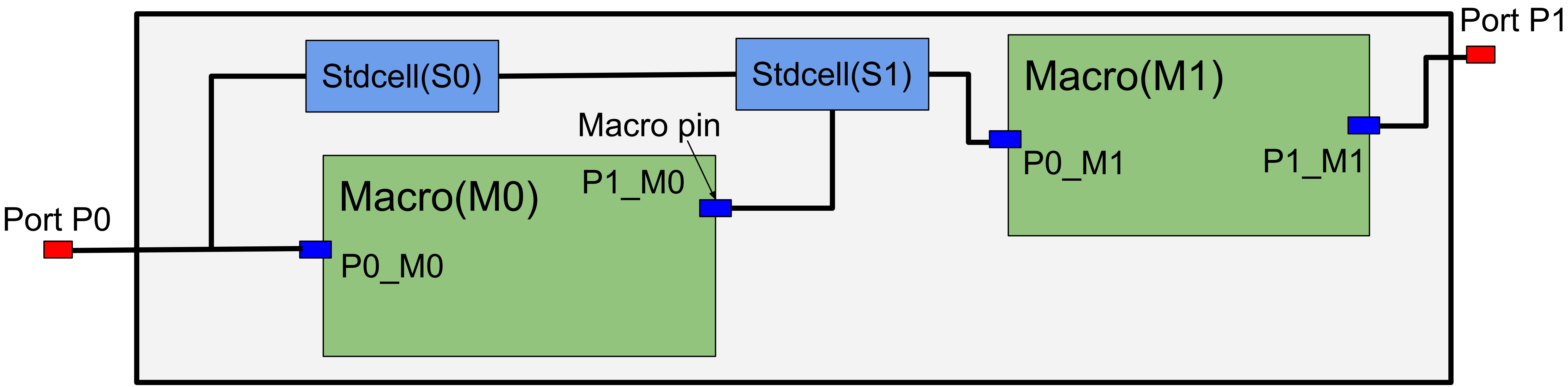
Advantages of Learning Based Approaches

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.



Prior Approaches to Chip Placement

Partitioning-Based Methods
(e.g. MinCut)

Stochastic/Hill-Climbing Methods
(e.g. Simulated Annealing)

Analytic Solvers
(e.g. RePlAce)

Prior Approaches to Chip Placement

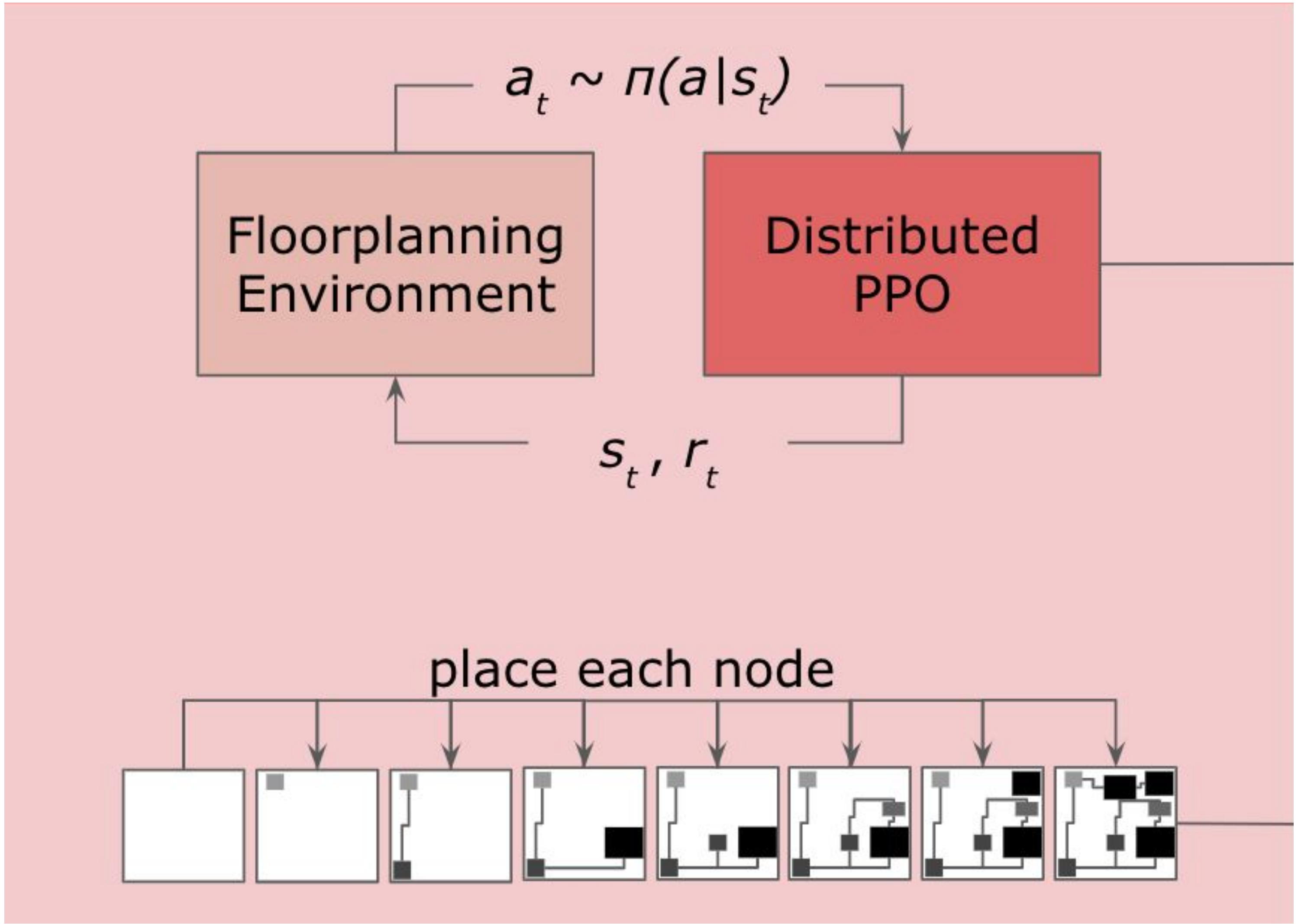
Partitioning-Based Methods (e.g. MinCut)	Stochastic/Hill-Climbing Methods (e.g. Simulated Annealing)
Analytic Solvers (e.g. RePlAce)	Learning-Based Methods

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

Reward corresponding to placement p of netlist (graph) g

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

Set of training graphs G

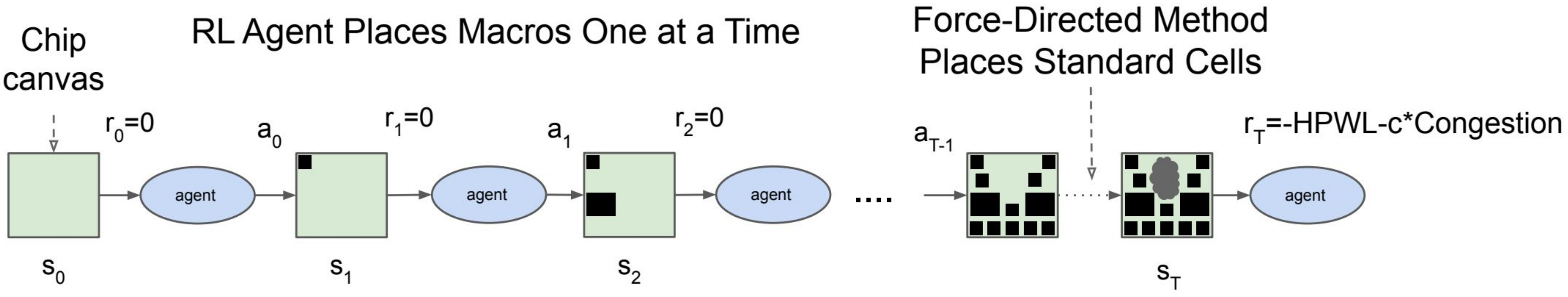
K is size of training set

RL policy parameterized by theta

$$R_{p,g} = \quad -Wirelength(p,g)$$
$$-\lambda\ Congestion(p,g) - \gamma Density(p,g)$$

Google

A Hybrid Approach to Placement Optimization



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

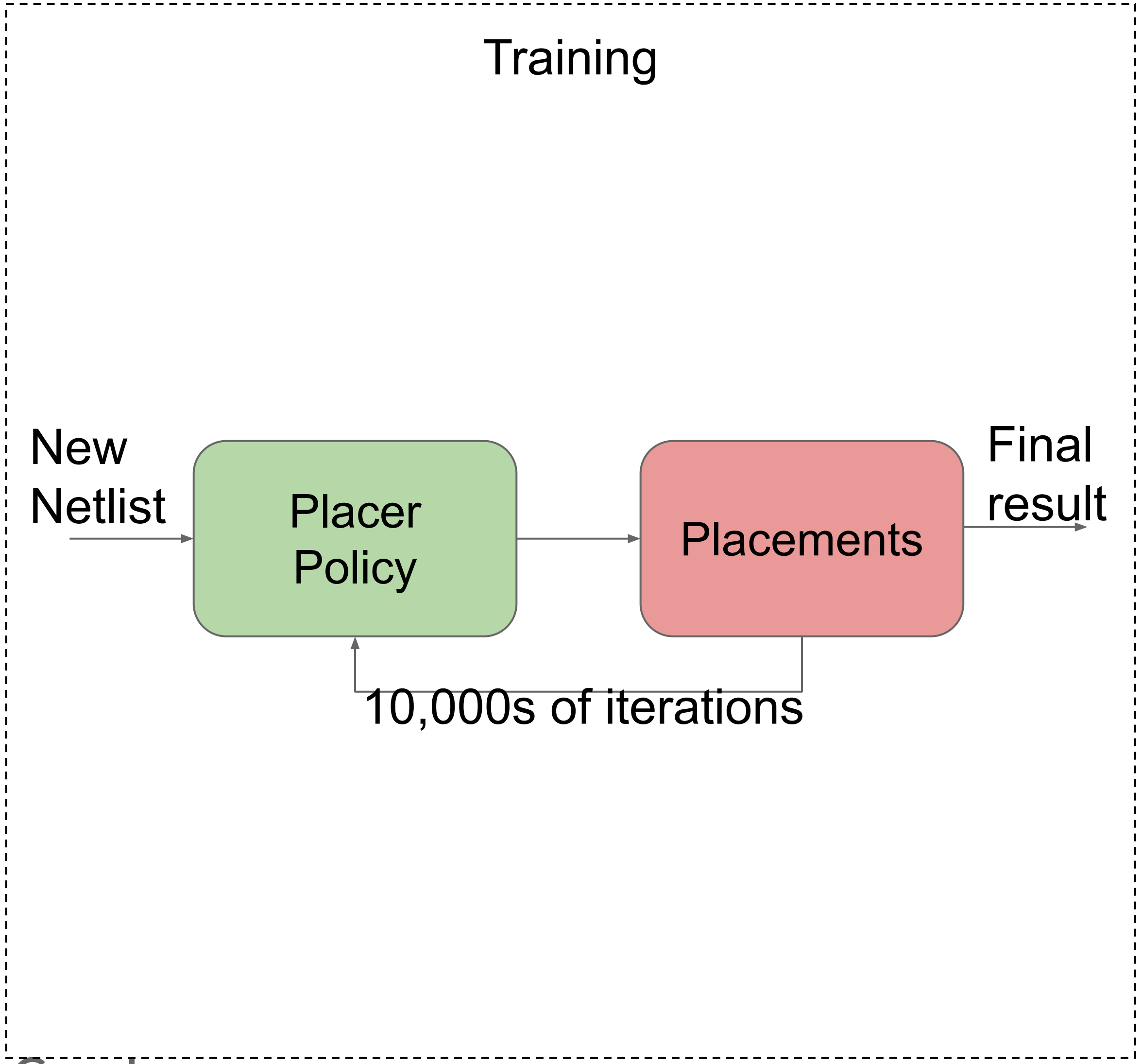
DRC: Design Rule Checking

ML Placer

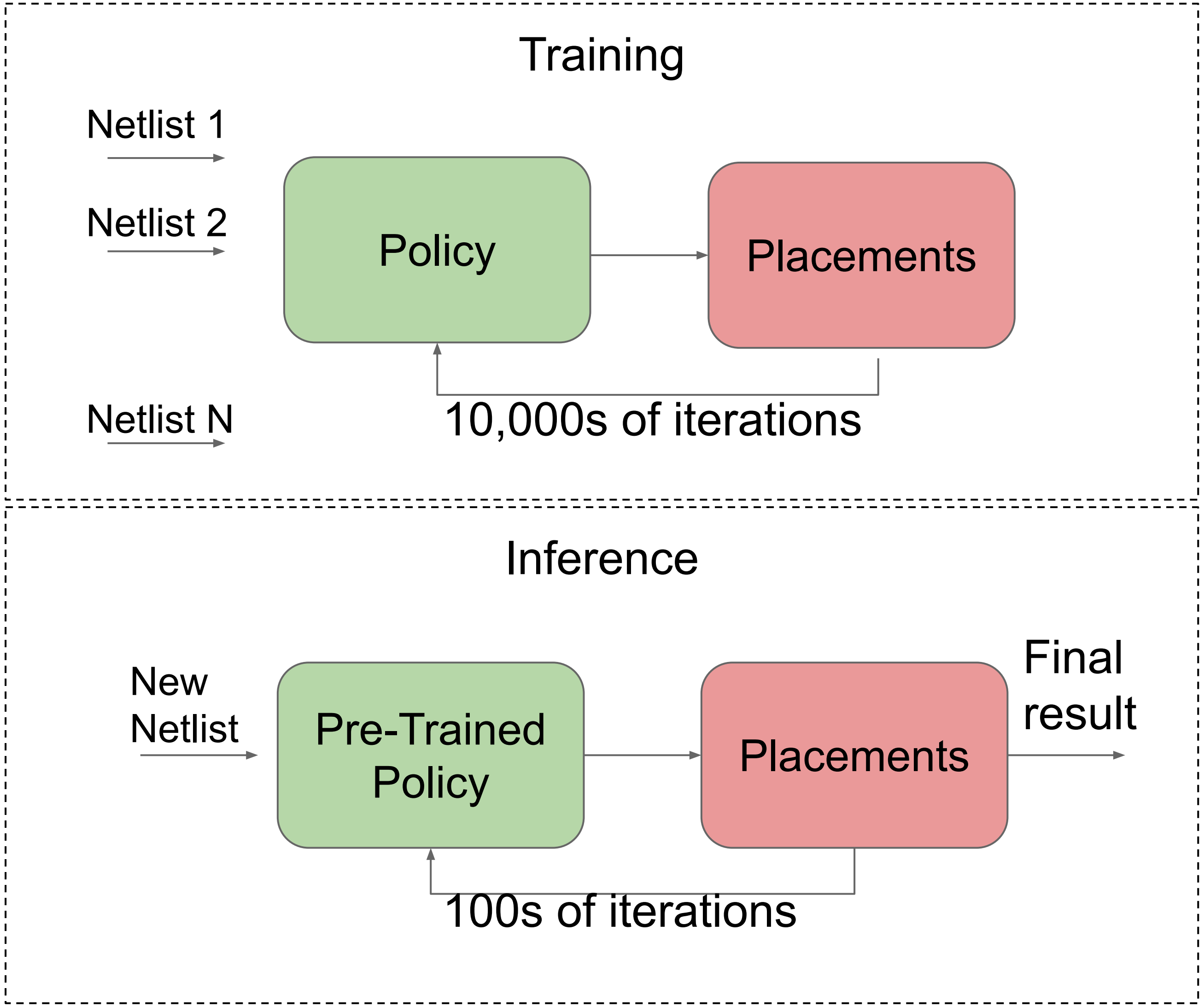
Time taken: **24 hours**
Total wirelength: 55.42m (-2.9% shorter)
Route DRC violations: 1789 (+23 - negligible difference)

Moving Towards Generalized Placements

Before: Training from scratch for each netlist



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



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

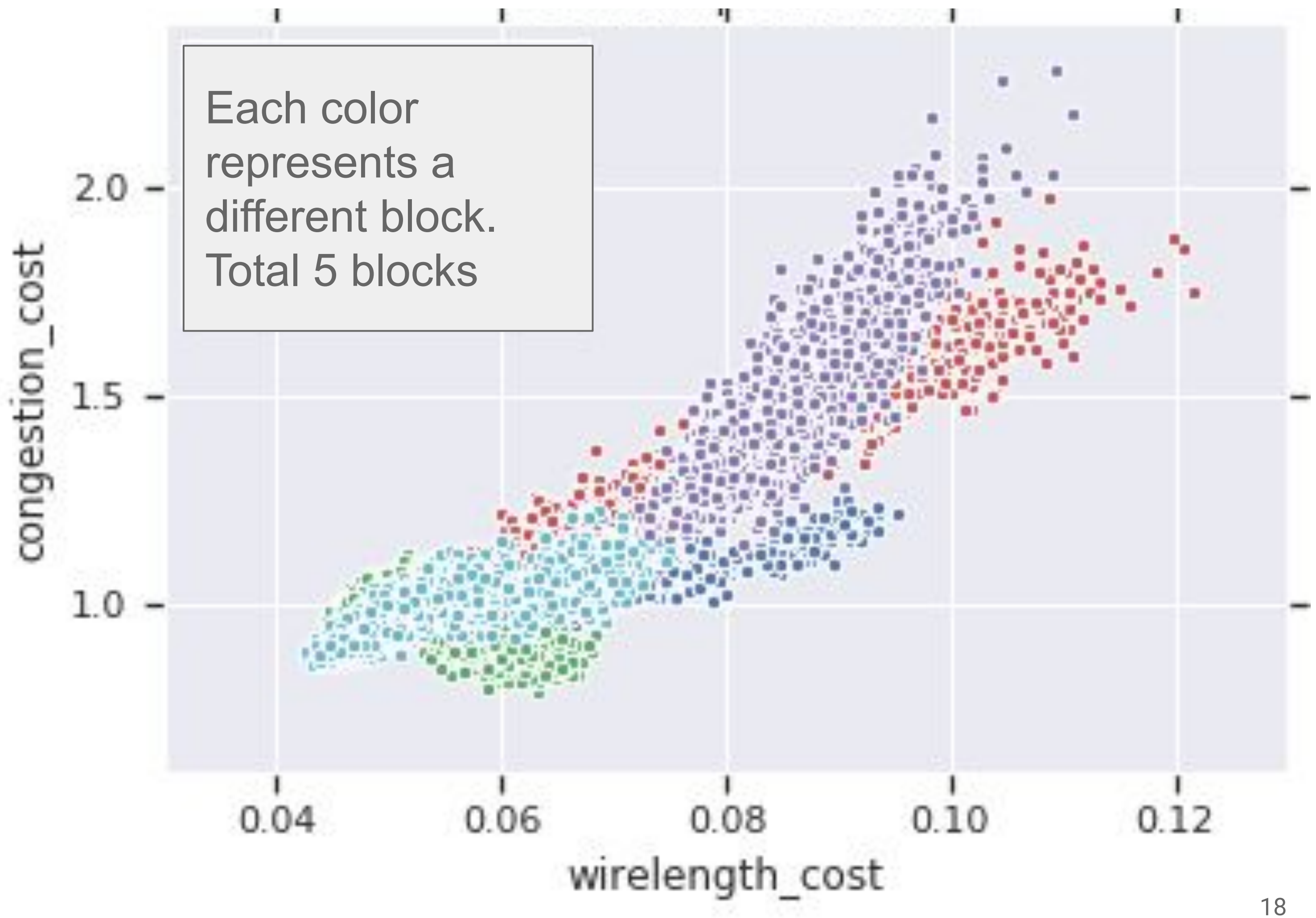
- We observed that a value network trained only on placements generated by a single policy is unable to accurately predict the quality of placements generated by another policy, limiting the ability of the policy network to generalize.

- To decompose the problem, we trained models capable of accurately predicting reward from off-policy data.

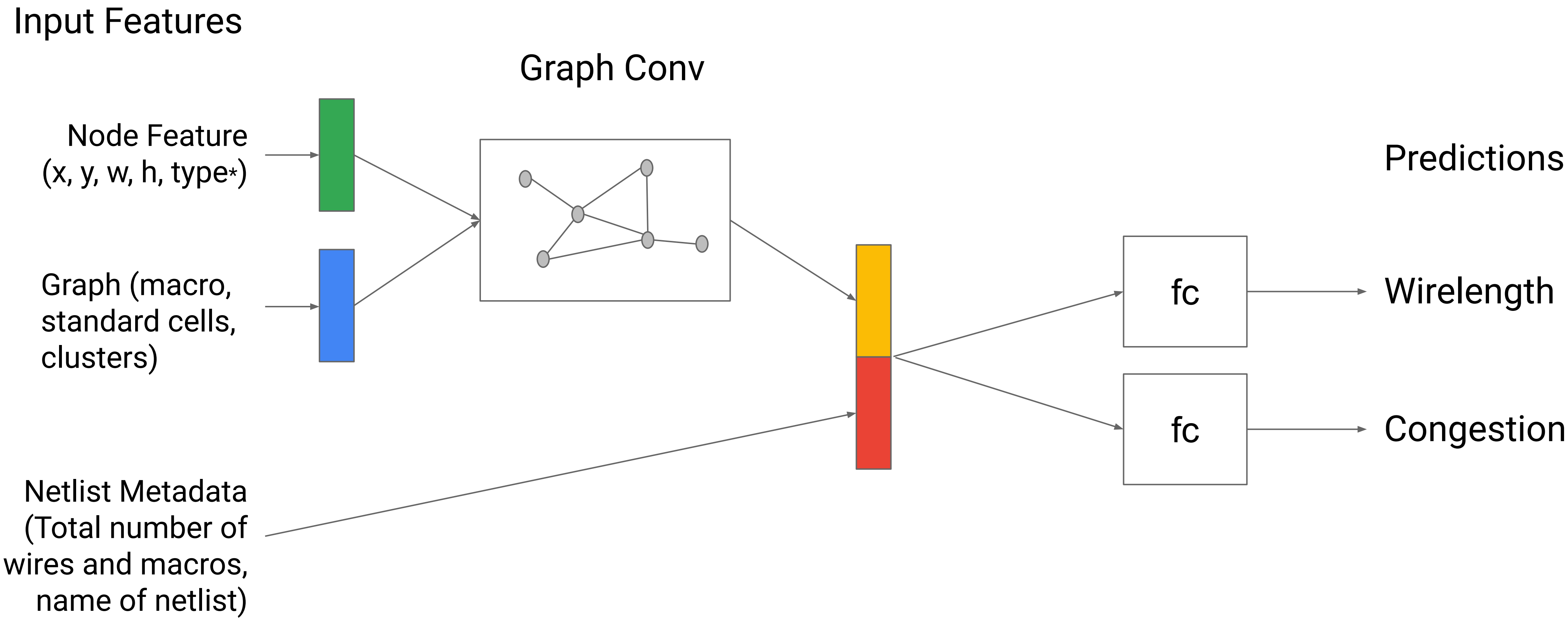
Compiling a Dataset of Chip Placements

- 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



*Node type: One-hot category {Hard macro, soft macro}

Different colors depict different part of the tensor.

Reward Model Architecture and Features

Input Features

Graph Conv

while *Not converged* **do**

  Update edge: $e_{ij} = fc_1(concat[fc_0(v_i)|fc_0(vj)|w^e_{ij}])$

  Update node: $v_i = mean_{j \in N(v_i)}(e_{ij})$

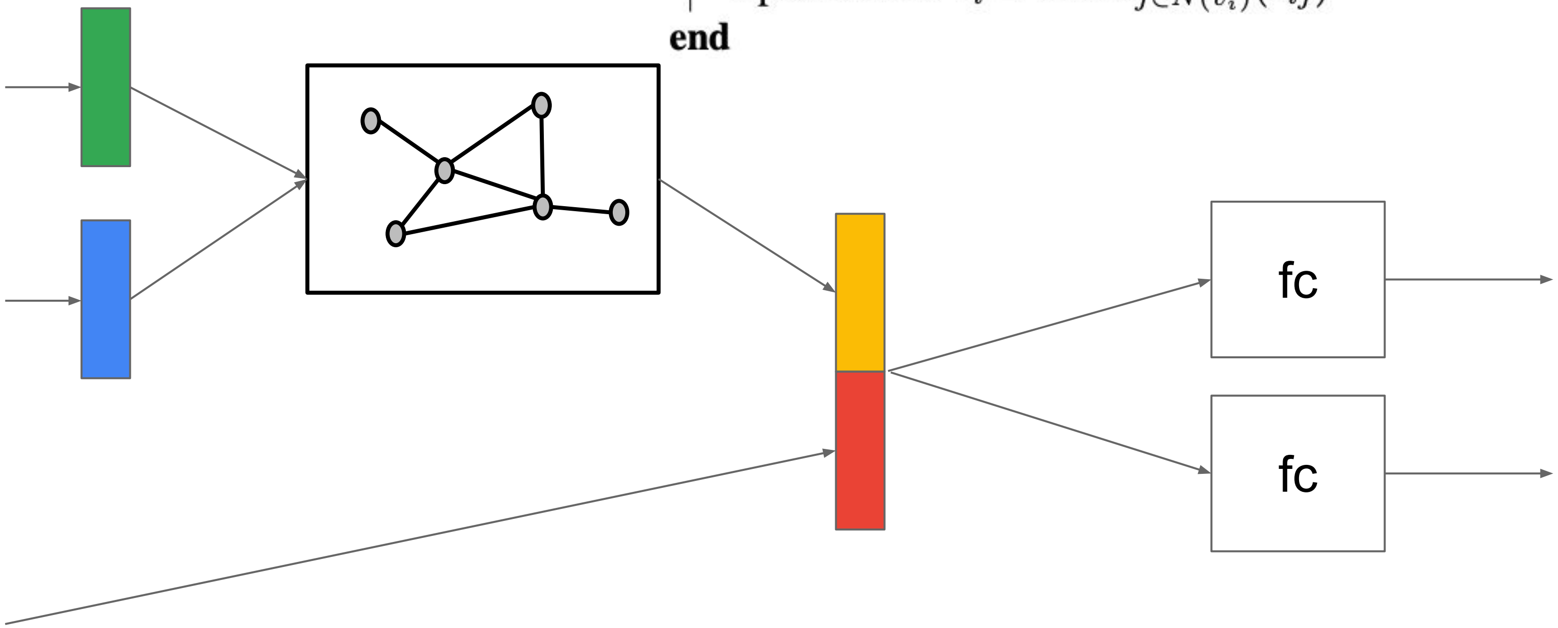
end

Predictions

Node Features
(x, y, w, h, type*)

Graph (macro, standard cells)

Netlist Metadata (Total number of wires and macros, name of netlist)

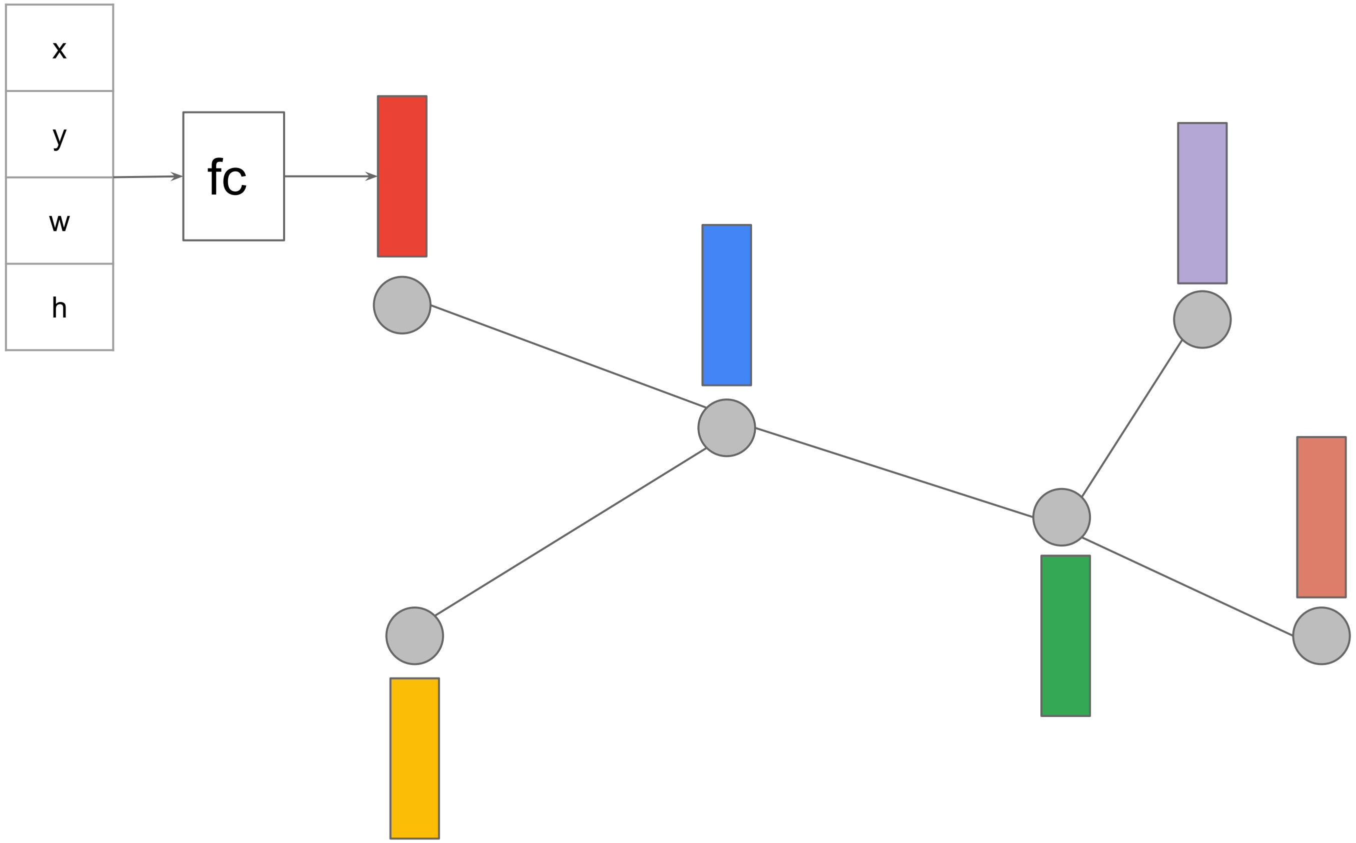


Wirelength

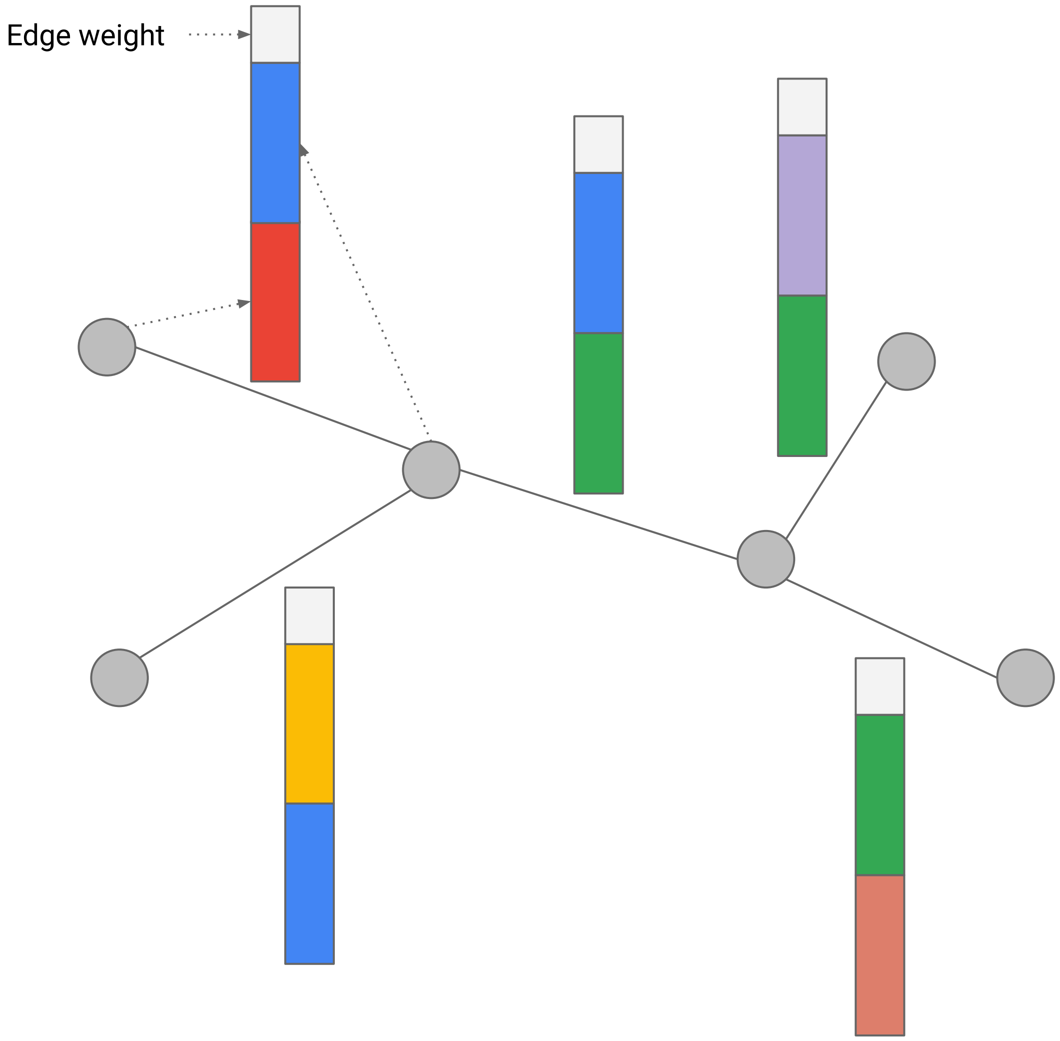
Congestion

*Node type: One-hot category {Hard macro, soft macro}

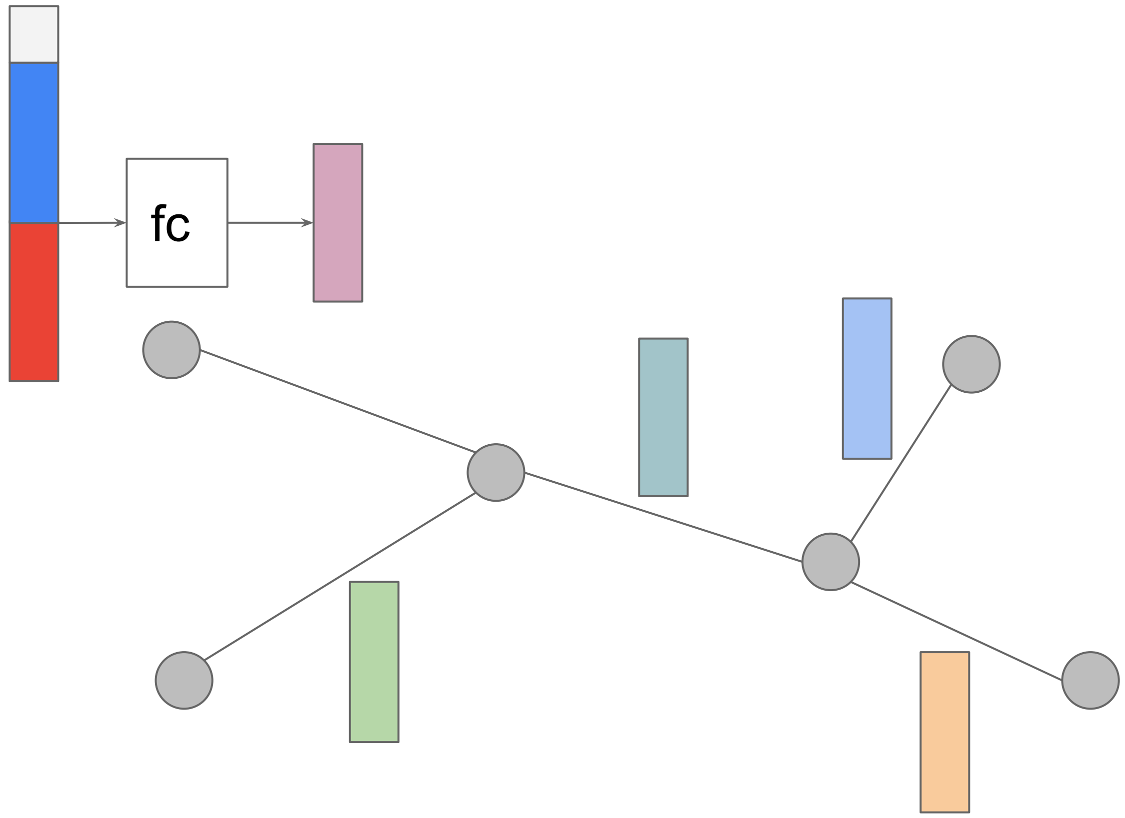
Edge-based Graph Convolution: Node Embeddings



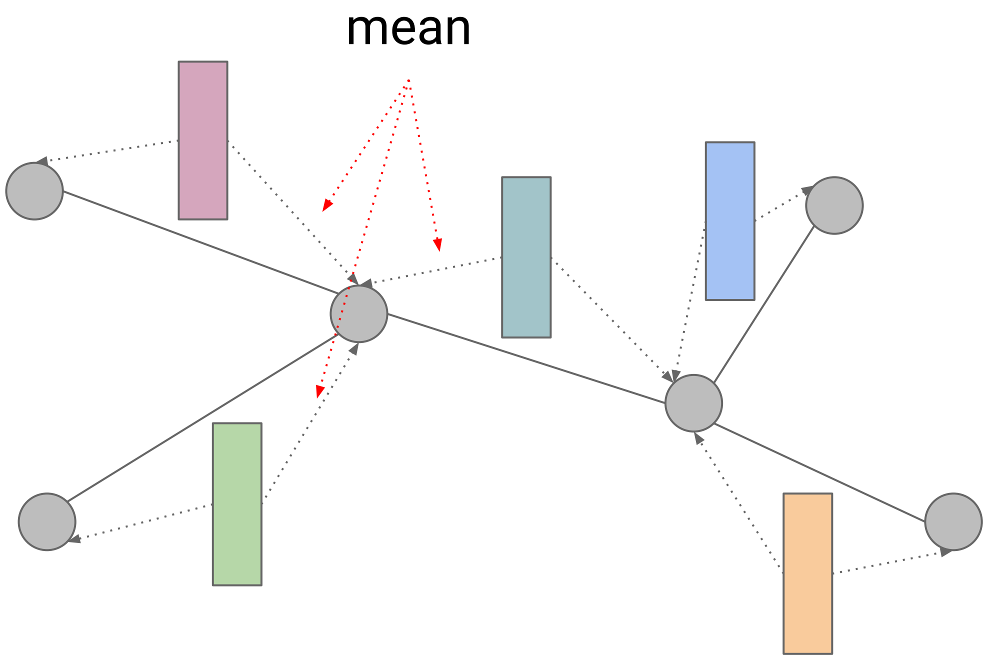
Edge-based Graph Convolution: Edge Embedding



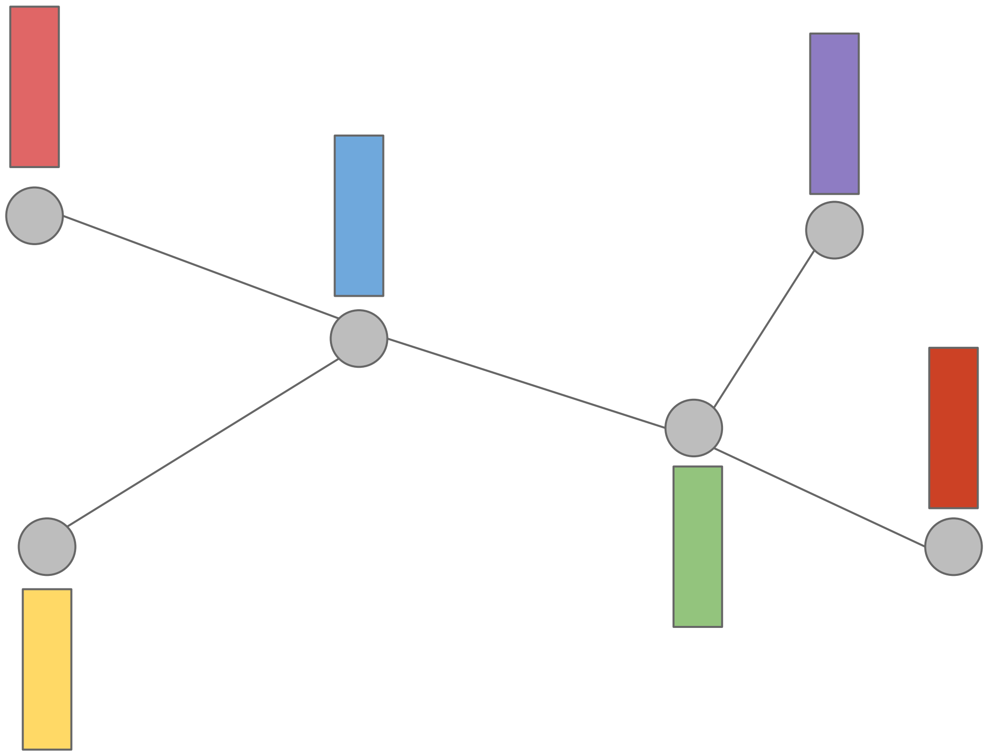
Edge-based Graph Convolution: Edge Embedding



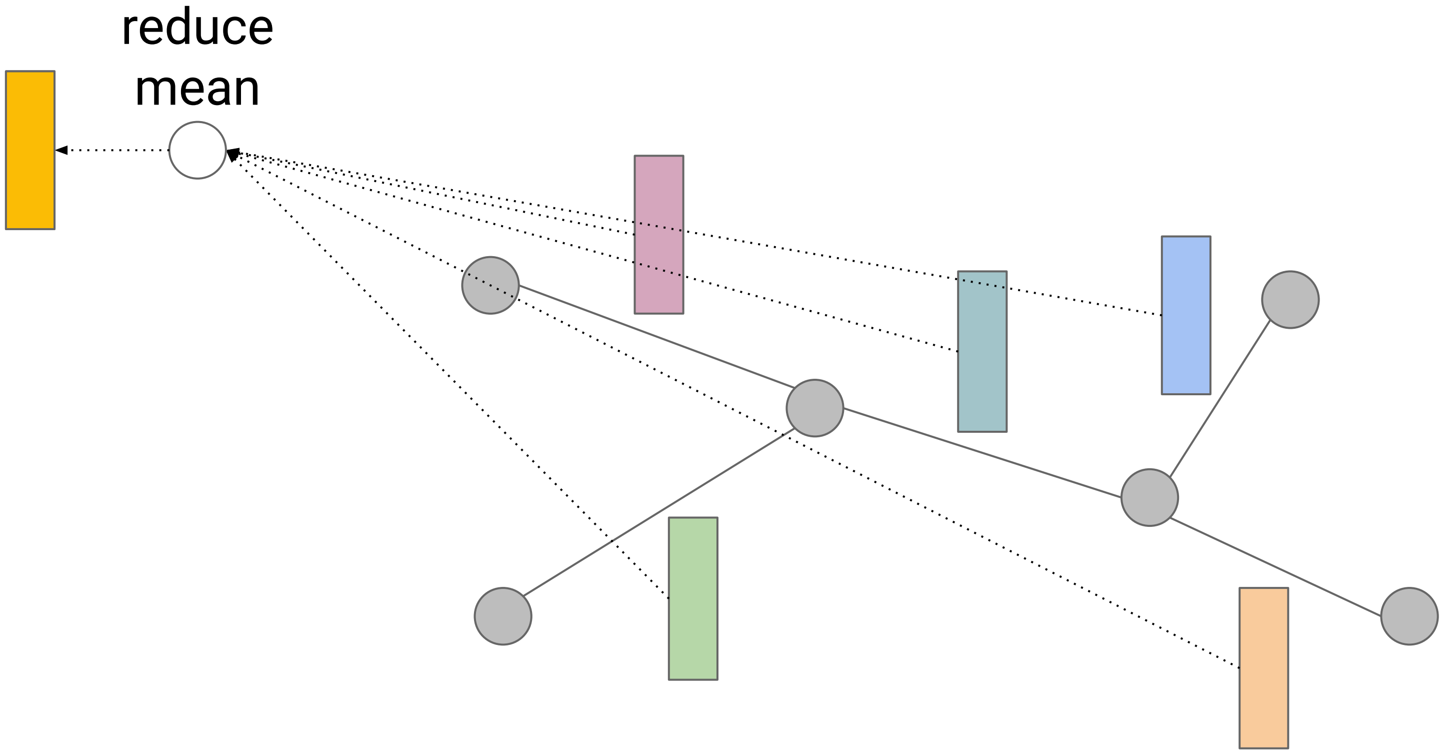
Edge-based Graph Convolution: Propagate



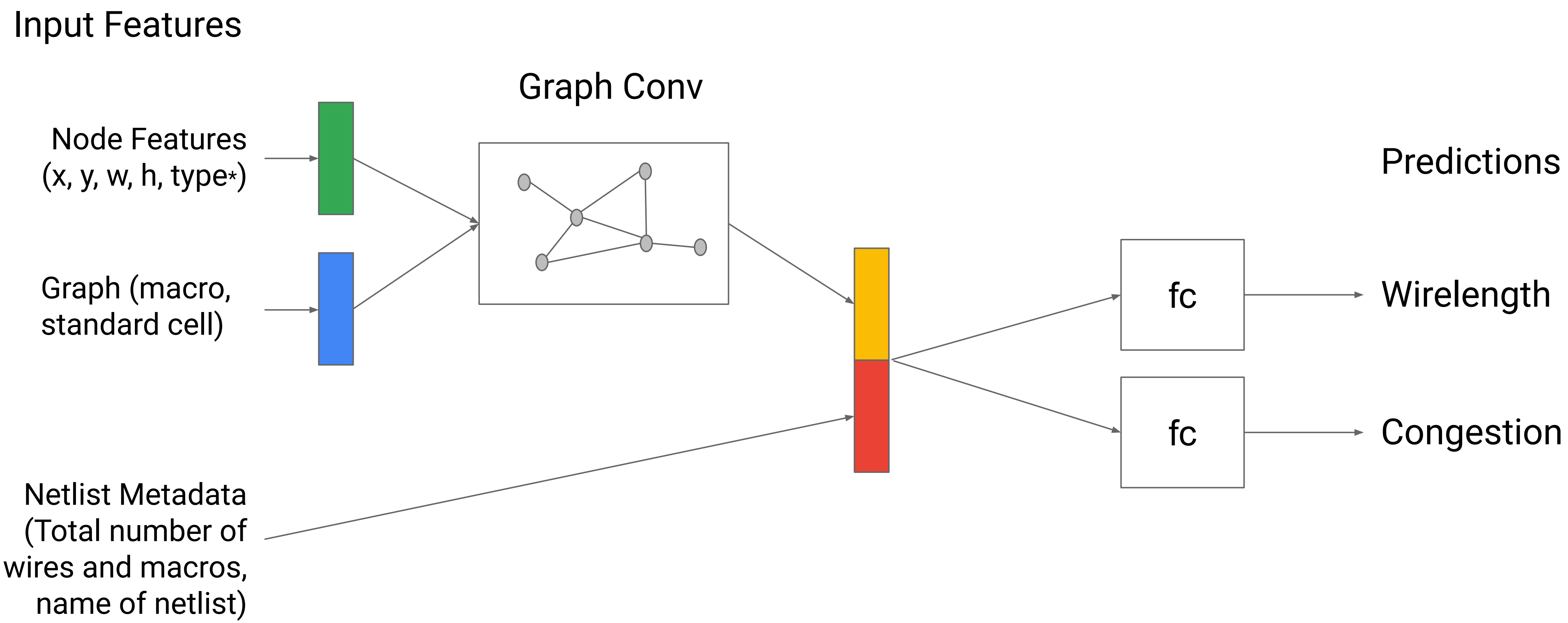
Edge-based Graph Convolution: Repeat



Final Step: Get Graph Embedding

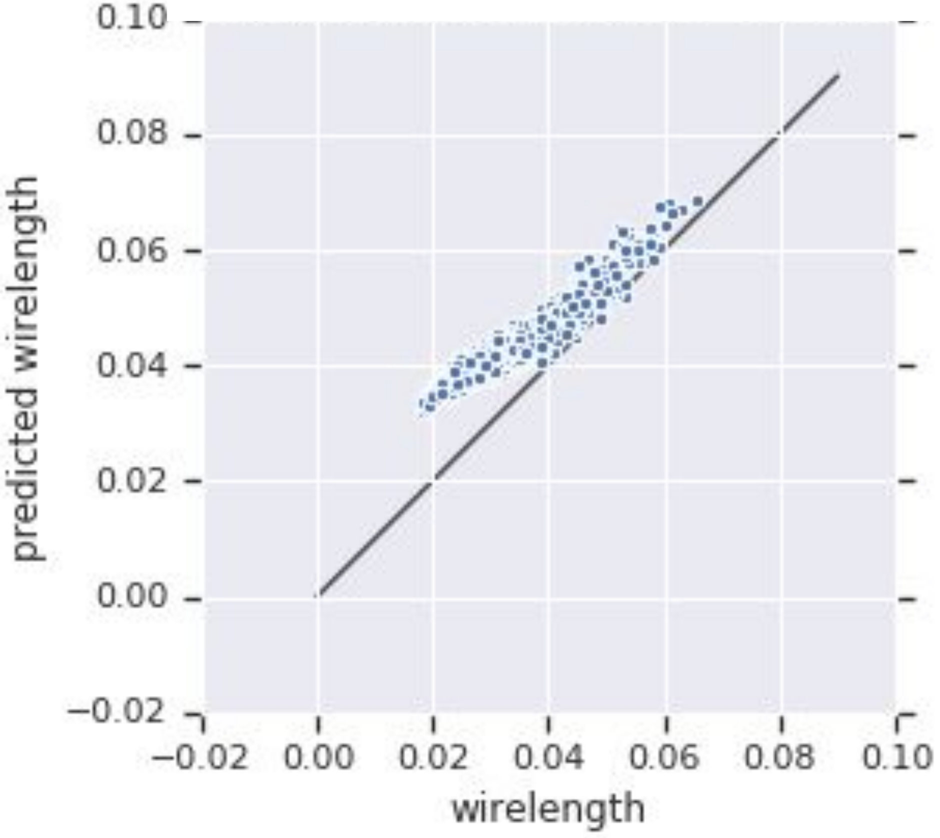


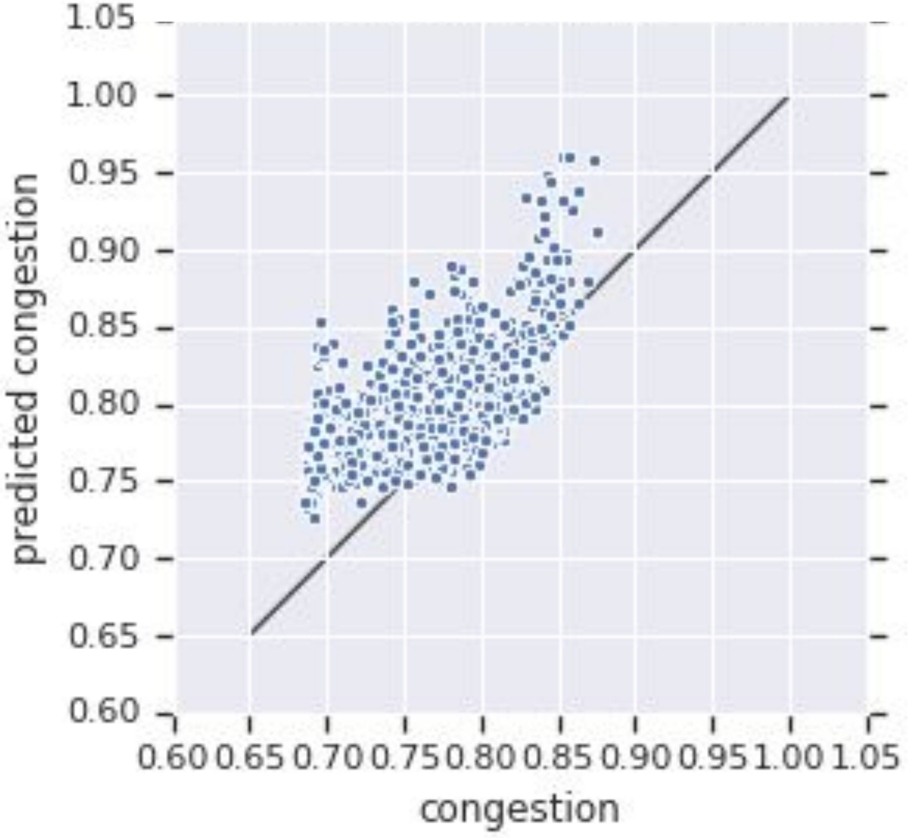
Reward Model Architecture and Features



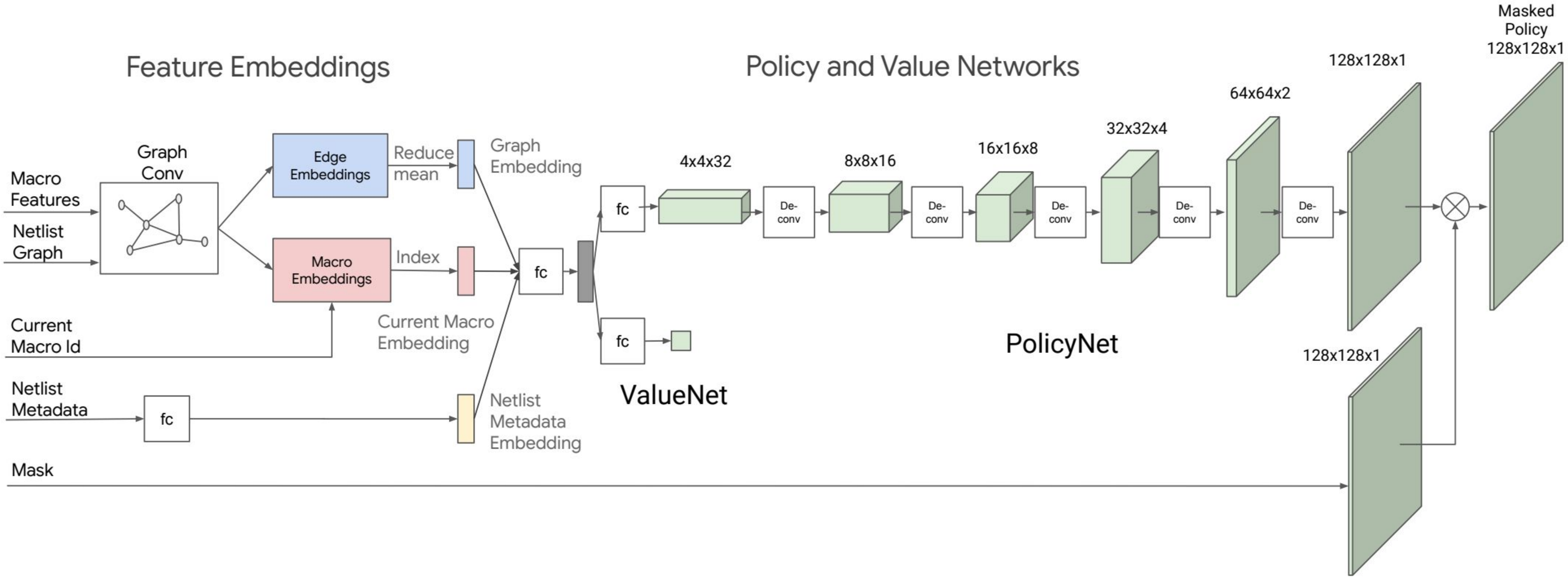
*Node type: One-hot category {Hard macro, soft macro}

Label Prediction Results on Test Chips





Policy/Value Model Architecture

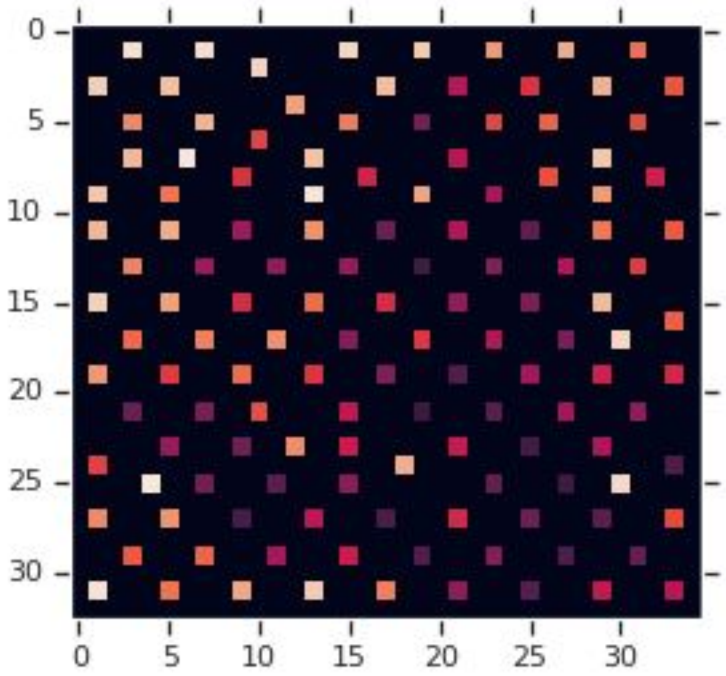


Experimental Setup

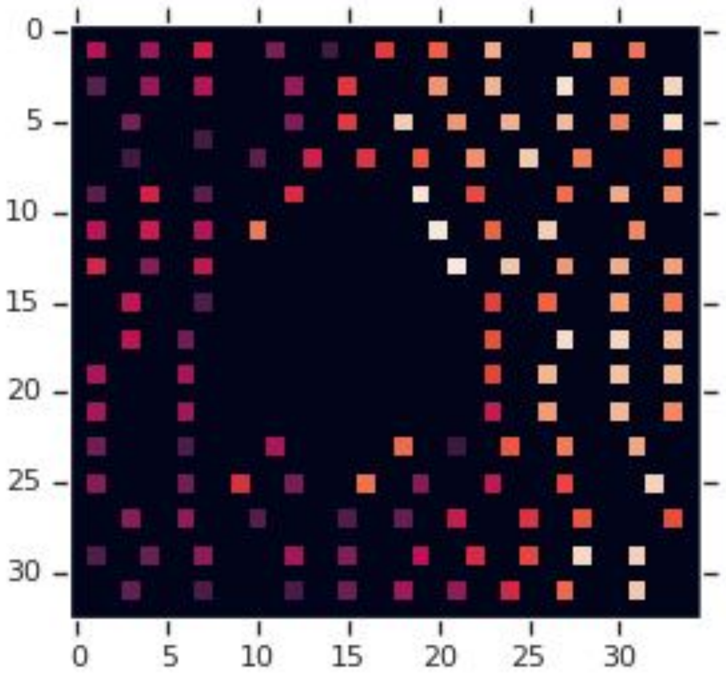
- For pre-training, we used the same number of workers as blocks in the training dataset
  - For example, for the largest training set with 20 blocks, we pre-trained with 20 GPU workers
- The pre-training runtime was 48 hours
- For fine-tuning results, our method ran on 16 GPU workers for up to 6 hours, but the runtime was often significantly lower due to early stopping
- For both pre-training and finetuning, 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



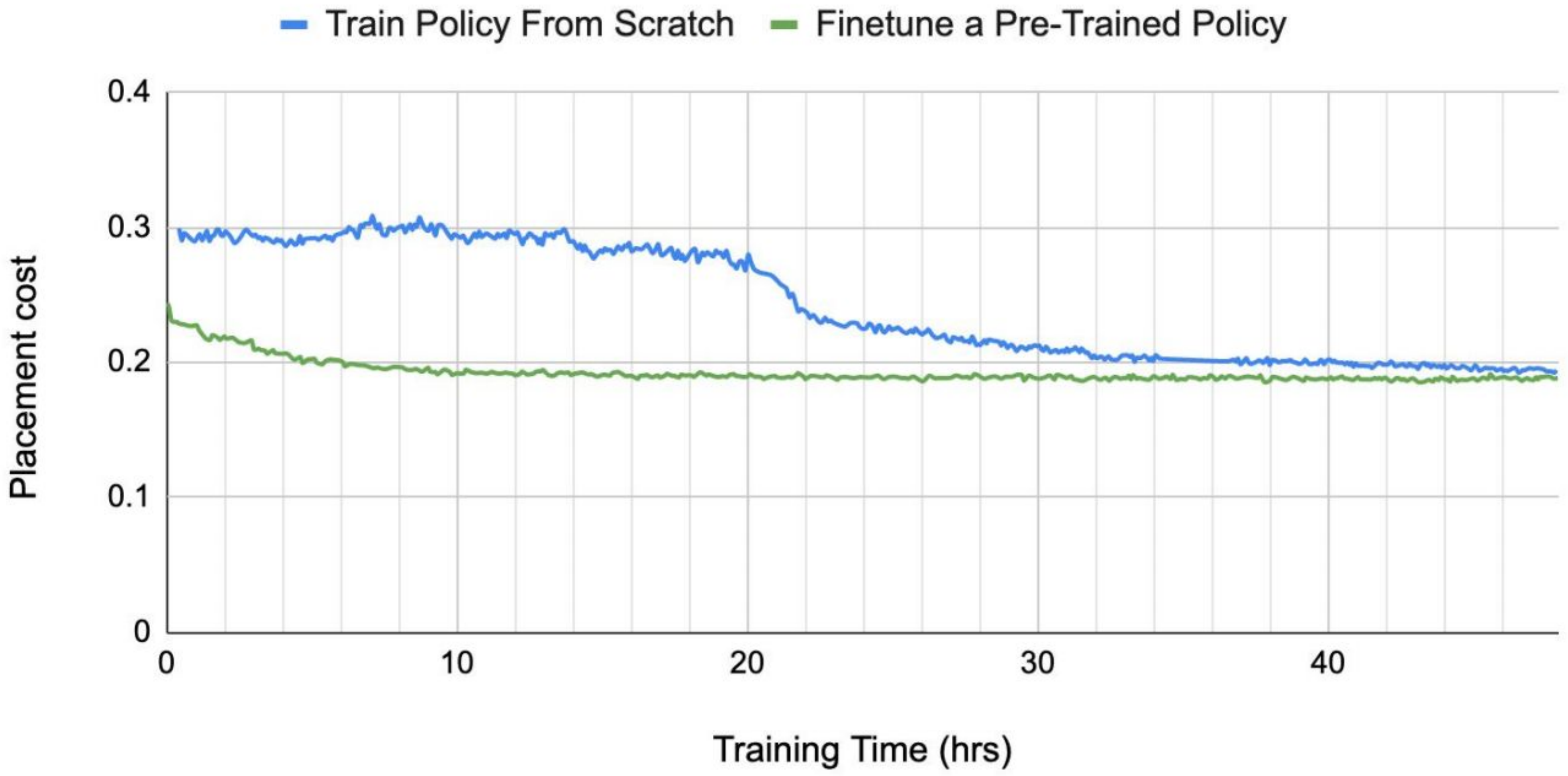
Finetuning a pre-trained policy



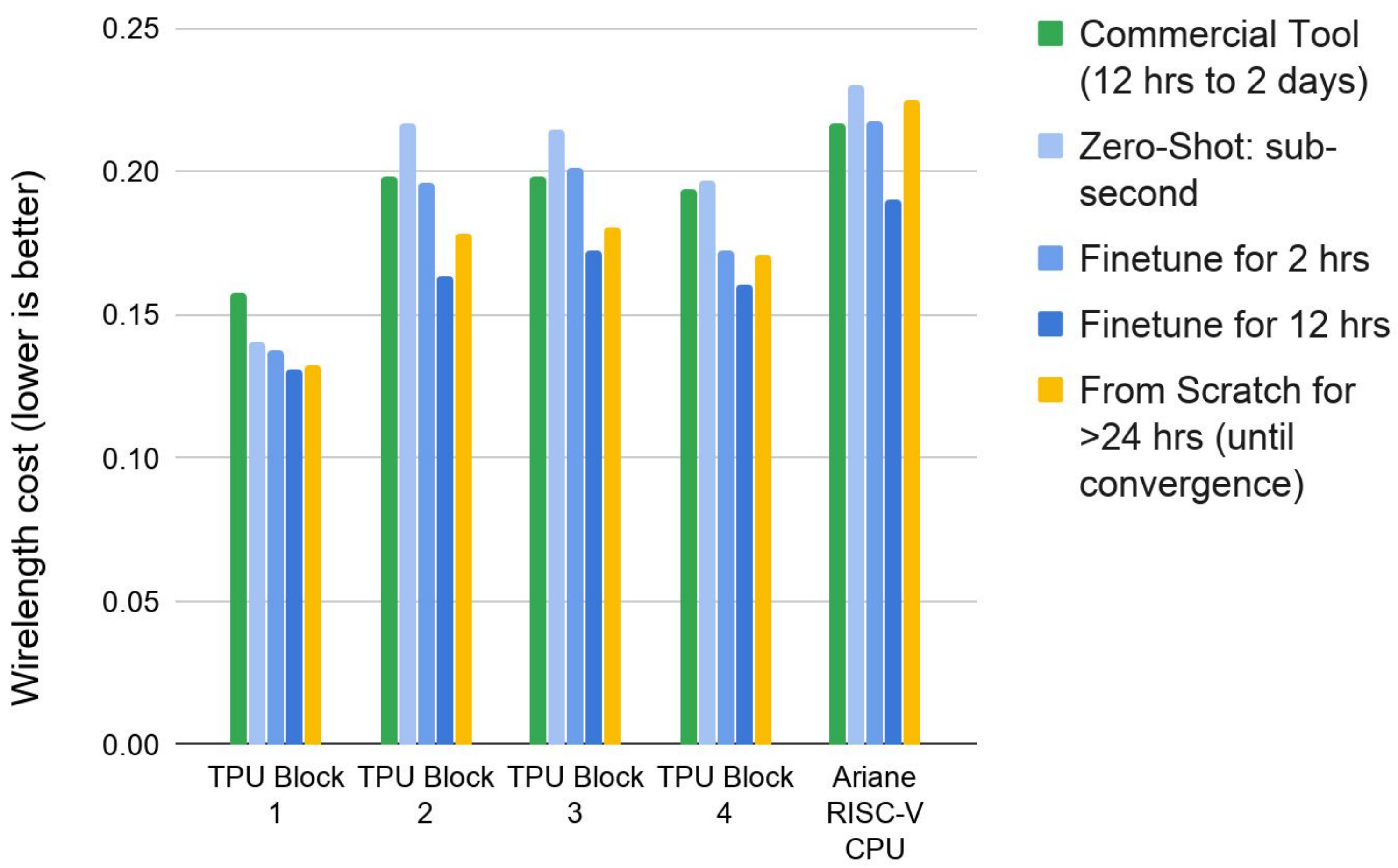
The animation shows the macro placements as the training progresses.
Each square shows the center of a macro.

Ariane is an open-source RISC-V processor. See: https://github.com/pulp-platform/ariane

Convergence Curve: Training from Scratch vs. Finetuning

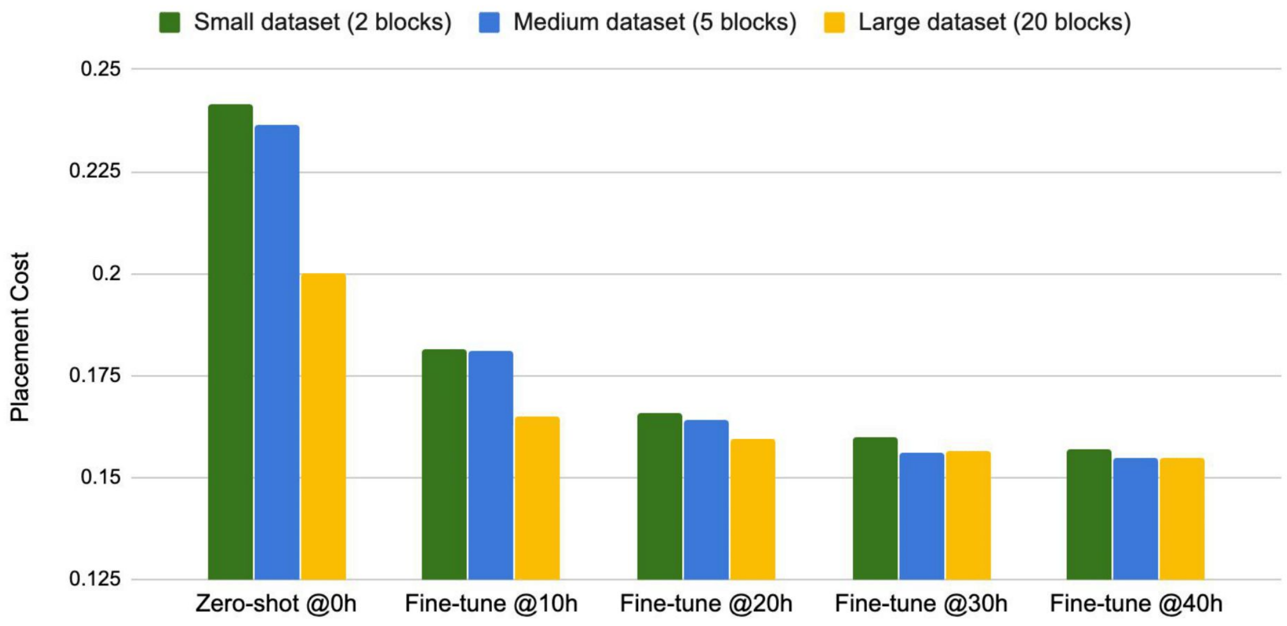


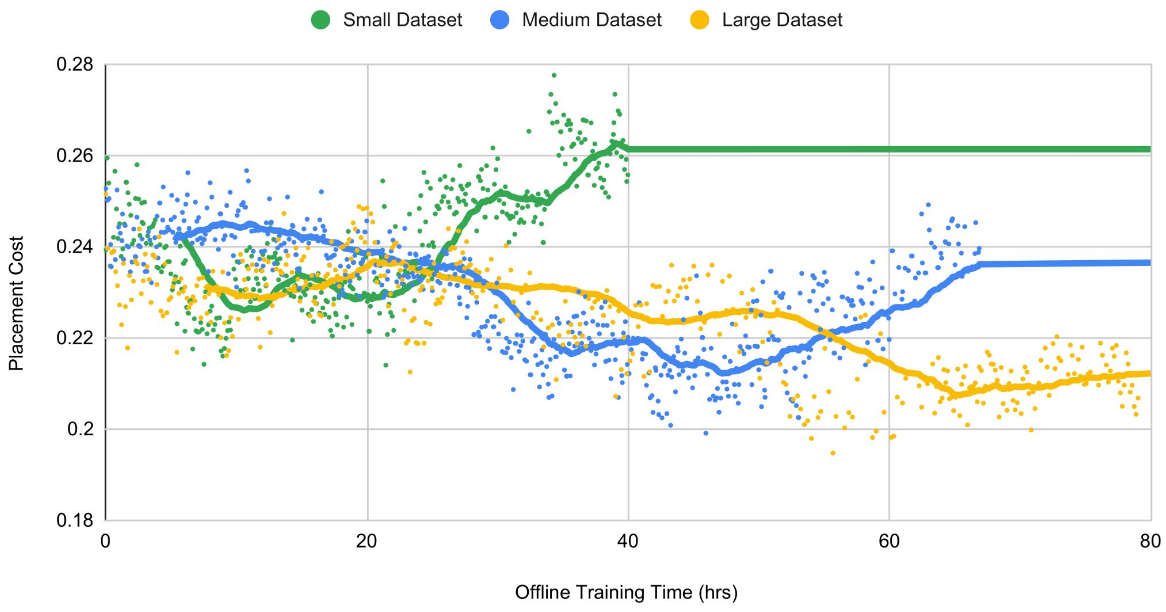
Generalization Results



The zero shot and fine-tuned policies were pre-trained on other blocks for ~24 hrs.

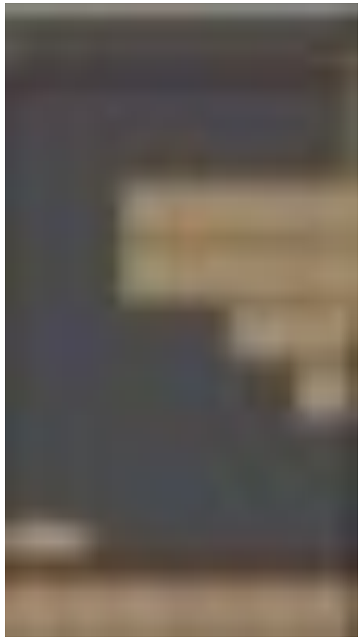
Effects of Training Set Size on Convergence



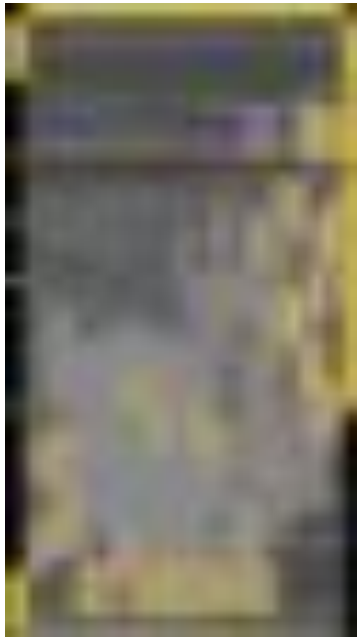


Google

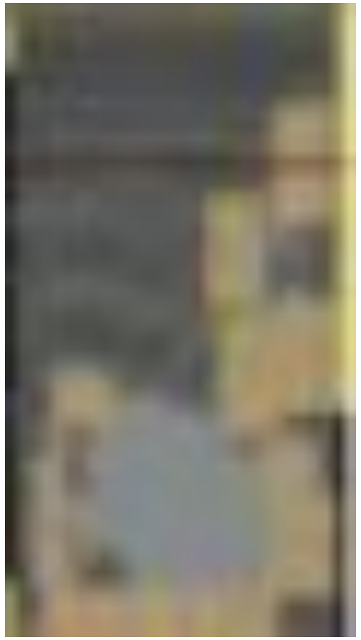
Humans Were Inspired by Our ML Placer!



Manual



ML Placer



Inspired Manual

	Wirelength	Congestion_ Horizontal	Congestion_ Vertical	Worst Negative Slack	Total Negative Slack	Number of Violating Endpoints	Area Buf Inv	Area Total	EDA Tool Run Time
ML Placer	2.45E+07	0.00	0.01	-0.143	-11.45	508	5,619	323,964	6.6hr
Inspired Manual	2.44E+07	0.00	0.01	-0.114	-16.29	901	5,634	324,094	7.1hr

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.

Name	Method	Timing		Area	Power	Wirelength	Congestion	
		WNS (ps)	TNS (ns)	Total (μm^2)	Total (W)	(m)	H (%)	V (%)
Block 1	RePlAce	374	233.7	1693139	3.70	52.14	1.82	0.06
	Manual	136	47.6	1680790	3.74	51.12	0.13	0.03
	Ours	84	23.3	1681767	3.59	51.29	0.34	0.03
Block 2	RePlAce	97	6.6	785655	3.52	61.07	1.58	0.06
	Manual	75	98.1	830470	3.56	62.92	0.23	0.04
	Ours	59	170	694757	3.13	59.11	0.45	0.03
Block 3	RePlAce	193	3.9	867390	1.36	18.84	0.19	0.05
	Manual	18	0.2	869779	1.42	20.74	0.22	0.07
	Ours	11	2.2	868101	1.38	20.80	0.04	0.04
Block 4	RePlAce	58	11.2	944211	2.21	27.37	0.03	0.03
	Manual	58	17.9	947766	2.17	29.16	0.00	0.01
	Ours	52	0.7	942867	2.21	28.50	0.03	0.02
Block 5	RePlAce	156	254.6	1477283	3.24	31.83	0.04	0.03
	Manual	107	97.2	1480881	3.23	37.99	0.00	0.01
	Ours	68	141.0	1472302	3.28	36.59	0.01	0.03

- We freeze the macro placements generated by each method and report the place opt results by a commercial EDA.
- RePlAce: C. Cheng, A. B. Kahng, I. Kang and L. Wang, "RePlAce: Advancing Solution Quality and Routability Validation in Global Placement," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2018

Comparisons with Commercial Auto Macro Placers

Commercial Auto Macro Placers:

EDA Vendor A

- Legacy auto macro placer: WL Driven
- Mixed size placer: WL Driven, Congestion Opt, Timing Opt, WL+Cong+Timing

EDA Vendor B

- Legacy auto macro placer: WL Driven, Congestion Opt, Timing Opt, WL+Cong+Timing
- ‘Advanced’ auto macro placer: WL Driven, Congestion Opt, Timing Opt, WL+Cong+Timing

TPU-v5 Block Compositions

	Low Macro Count	**Medium Macro Count**	**High Macro Count**	Blocks
Low Sat.	(7) 33.3% Low Sat. - Low Count	(4) 19.0% Low Sat. - Med Count	(2) 9.5% Low Sat. - High Count	**13 (62%)**
Med Sat.	(0) 0.0% Med Sat. - Low Count	(2) 9.5% Med Sat. - Med Count	(3) 14.3% Med Sat. - High Count	**5 (24%)**
High Sat.	(0) 0.0% High Sat. - Low Count	(1) 4.8% High Sat. - Med Count	(2) 9.5% High Sat. - High Count	**3 (14%)**
Blocks	**7 (33%)**	**7 (33%)**	**7 (33%)**	**21 blocks**

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.

Auto Macro Placer Comparison Summary

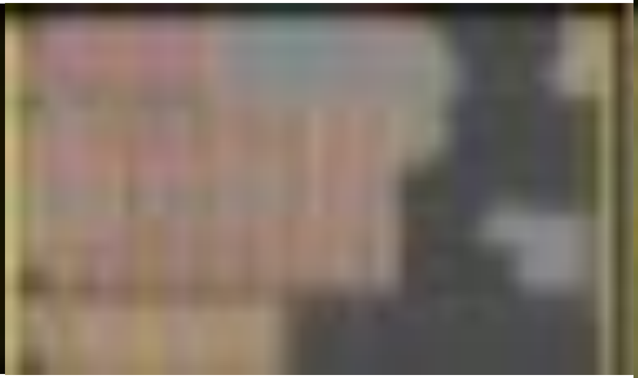
	ML Placer is superior	ML Placer is equal	ML Placer is inferior
EDA-A	15	4	1
EDA-B	13	3	4
Manual	11	5	4

	ML Placer	Manual	EDA-A	EDA-B
Best	13	9	5	6
Not best	7	11	15	14

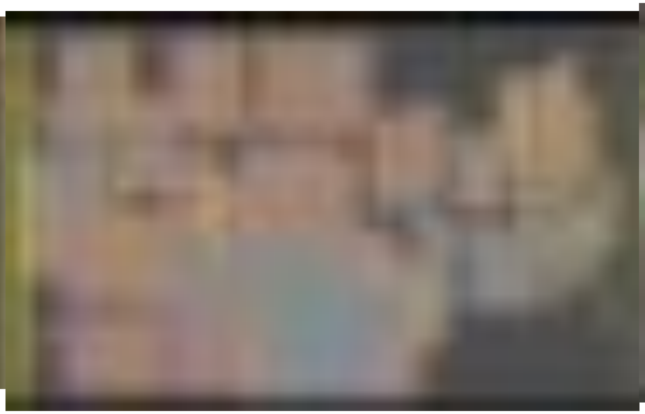
QoR Comparison for A TPU-v5 Block

ML Placer

Manual



EDA-A
Mixed Size Placer



EDA-B
Advanced

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

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?