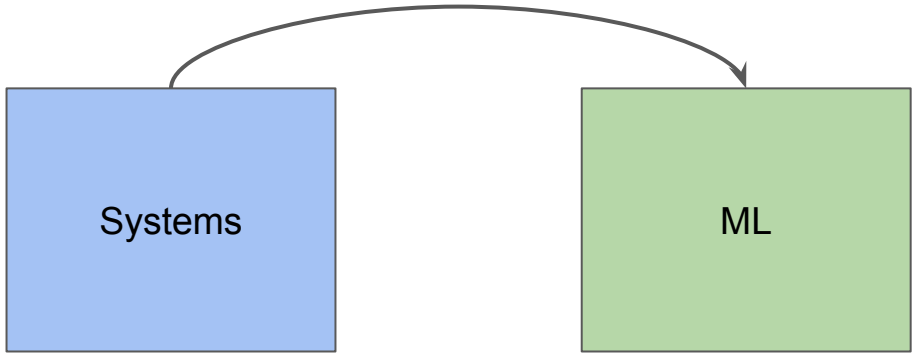
ML for Systems and Chip Design

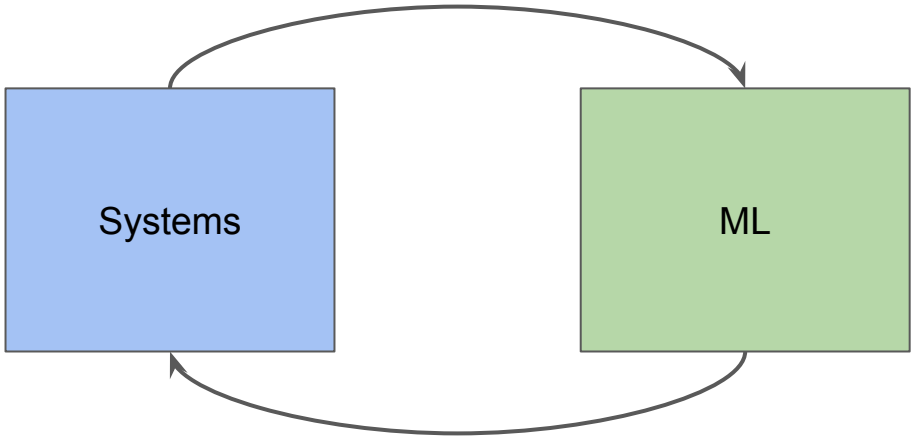
Azalia Mirhoseini, Anna Goldie
May 28th, 2020

In the past decade, systems and hardware have transformed ML.



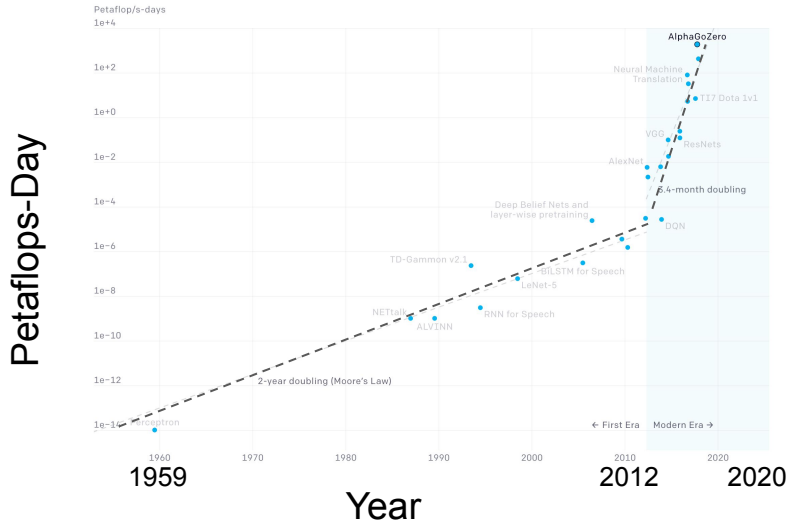
In the past decade, systems and hardware have transformed ML.

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

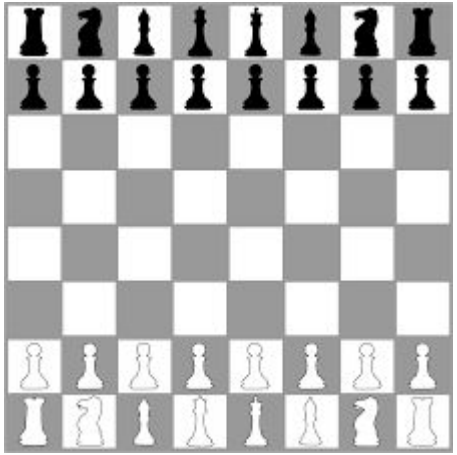
- Between 1959 to 2012, compute usage roughly doubled every two years
- Since 2012, the amount of compute used in the largest AI training runs doubled every 3.4 months[1]
- By comparison, Moore's Law had an 18-month doubling period!



1.OpenAI'19

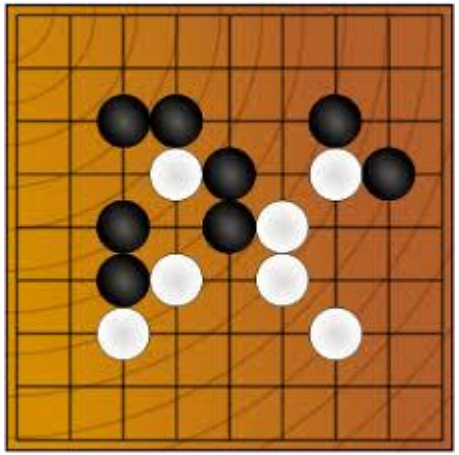
Complexity of Chip Placement Problem

Chess



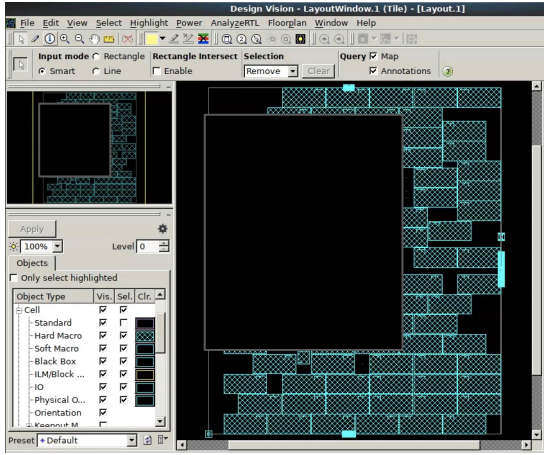
Number of states ~ 10^{123}

Go



Number of states ~ 10^{360}

Chip Placement

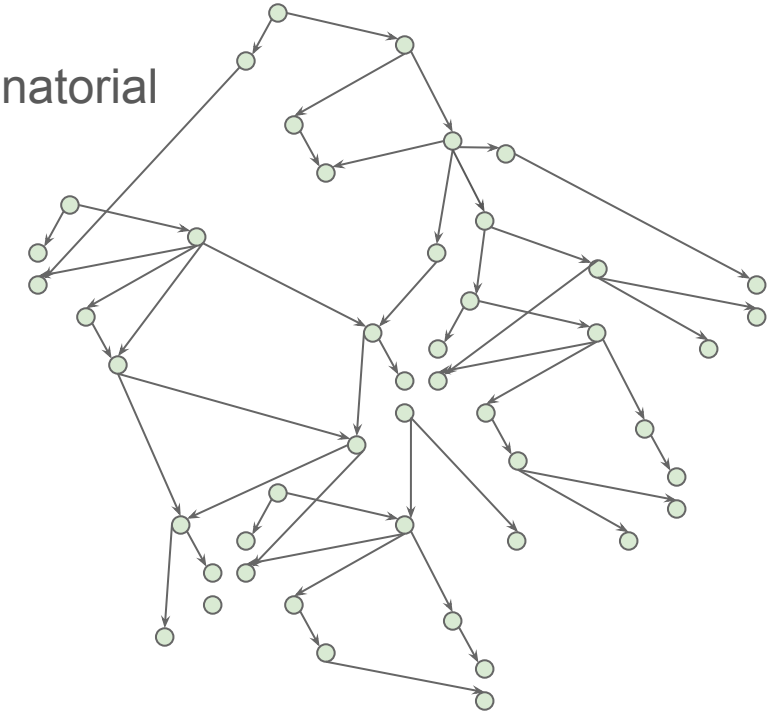


Number of states ~ 10^{9000}

Combinatorial Optimization on Graph Data

Many problems in systems and chips are combinatorial optimization problems on graph data:

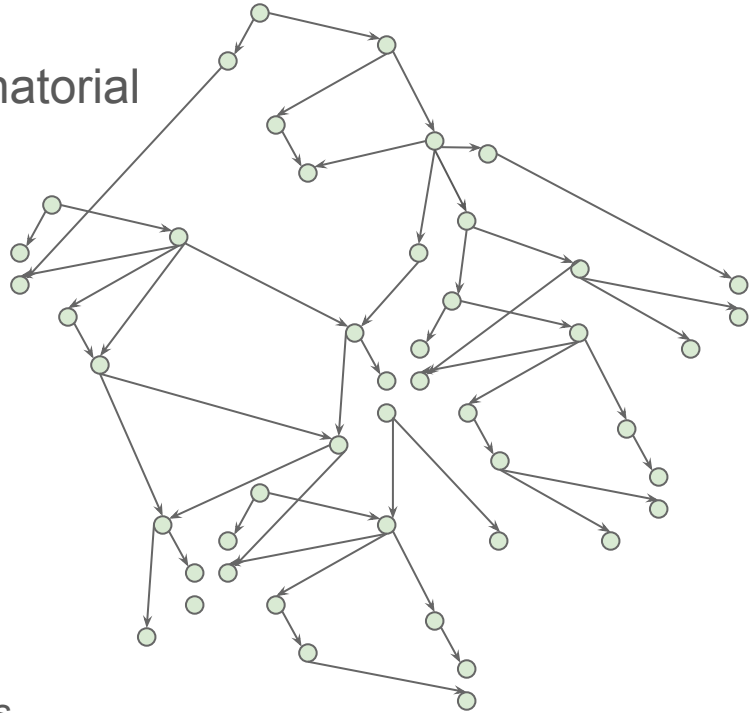
- Compiler optimization
- Chip placement
- Datacenter resource allocation
- ...



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
- ...



Advantages of Learning Based Approaches

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

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

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

Advantages of Learning Based Approaches

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

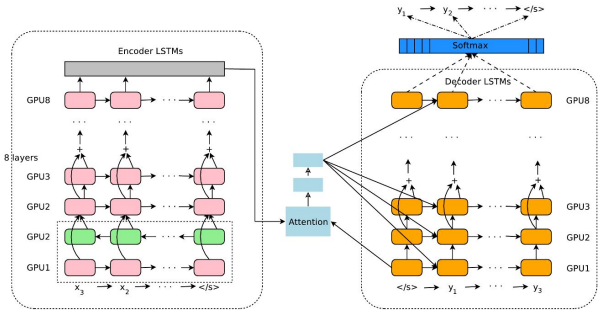
- Learn the underlying relationship between the context and target optimization metrics and leverage it to explore various optimization trade-offs
- “Gain experience” as they solve more instances of the problem and become “experts” over time
- Scale on distributed platforms and train billions of parameters

Outline of This Talk

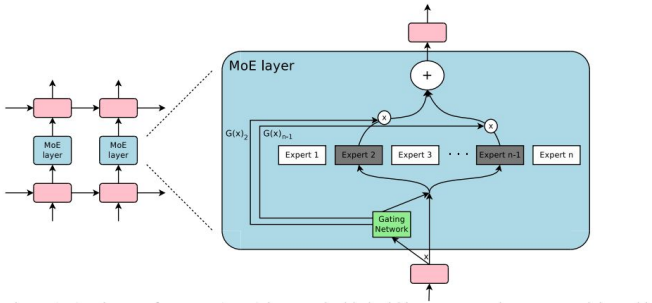
- **Learning to Optimize Device Placement**
- Learning to Partition Graphs
- Learning to Optimize Chip Placement

What Is Device Placement and Why Is It Important?

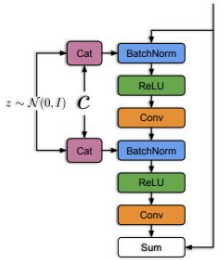
Trend towards many-device training, bigger models, larger batch sizes



Google neural machine translation'16
300 million parameters,
trained on 128 GPUs

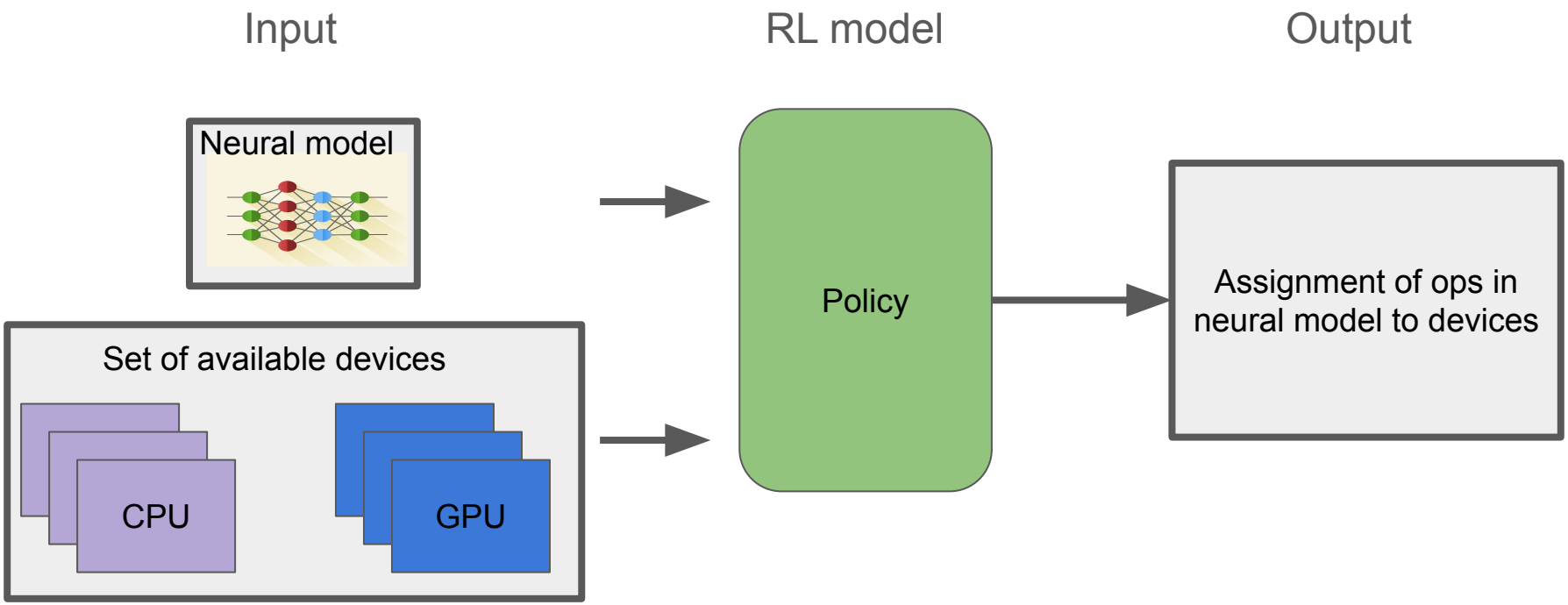


Sparsely gated mixture of experts'17
130 billion parameters,
trained on 128 GPUs

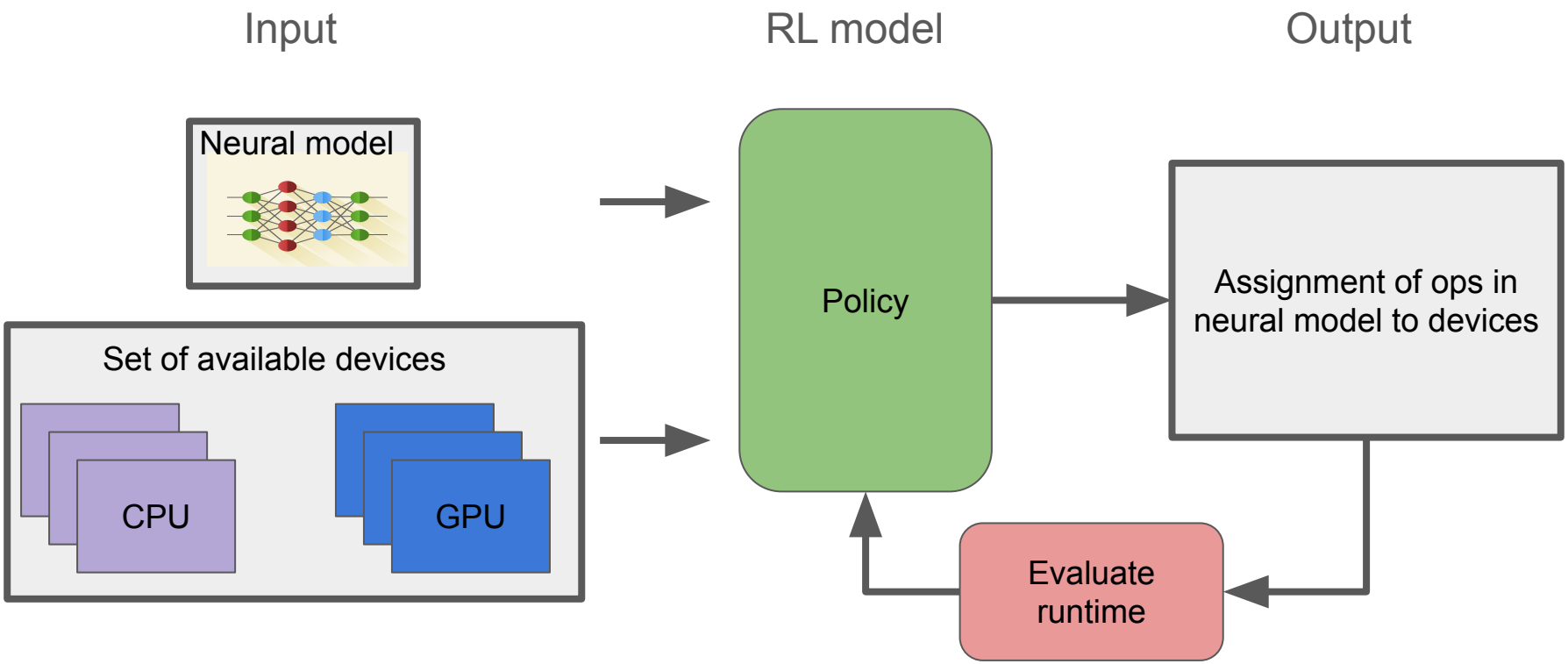


BigGAN'18
355 million parameters,
trained on 512 TPU cores

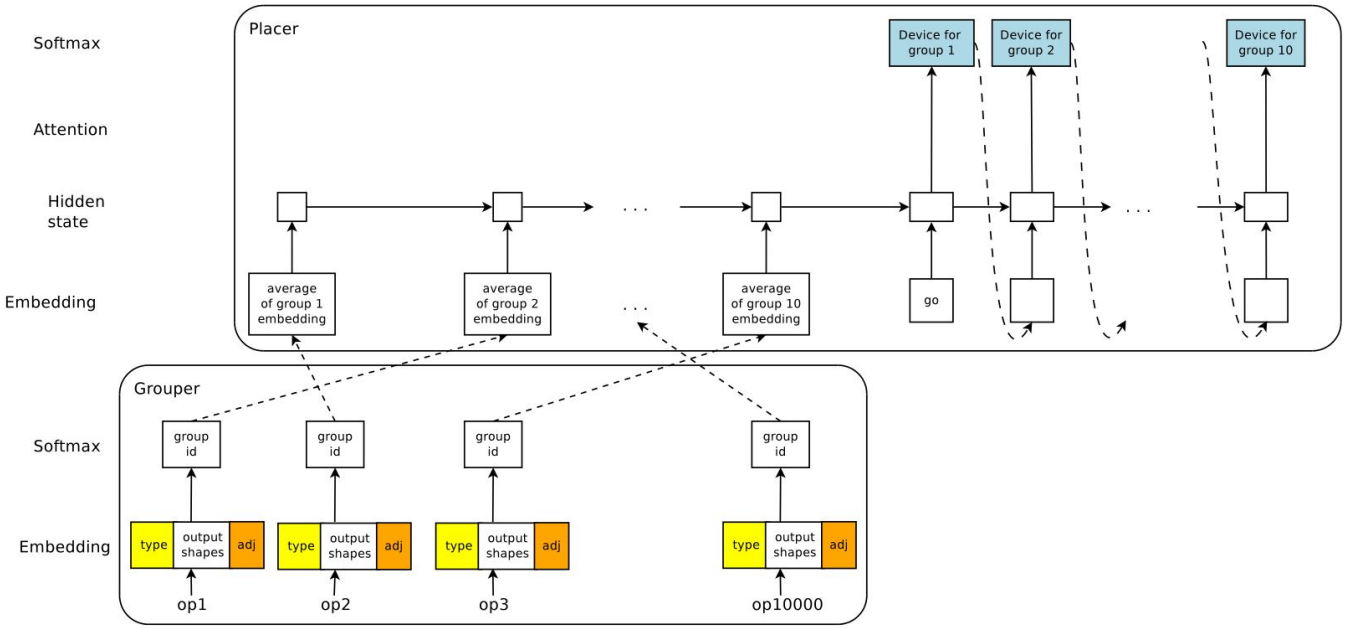
Posing Device Placement as an RL Problem



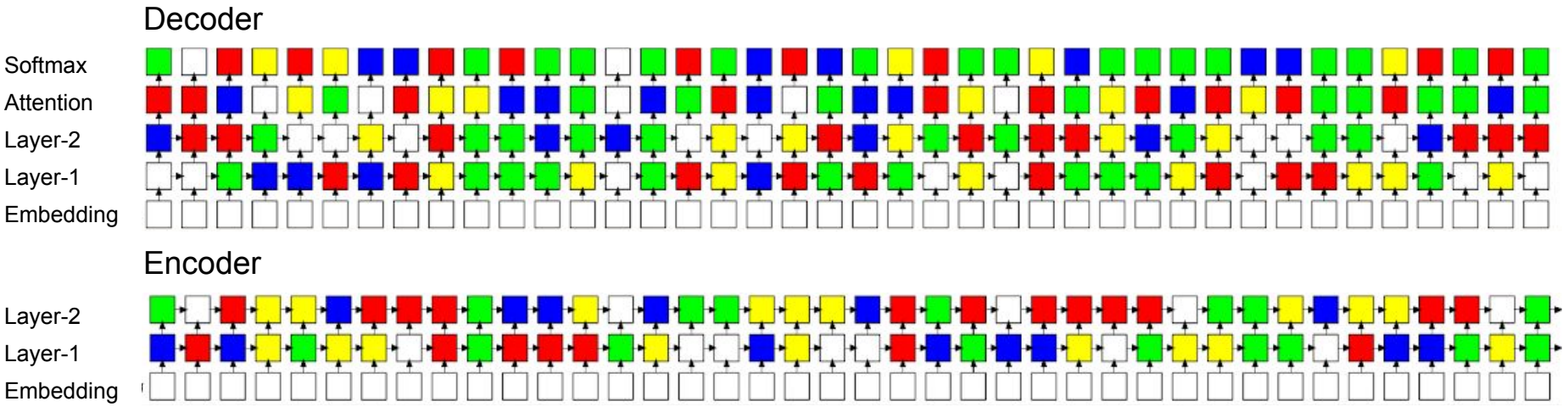
Posing Device Placement as an RL Problem



An End-to-End Hierarchical Placement Model

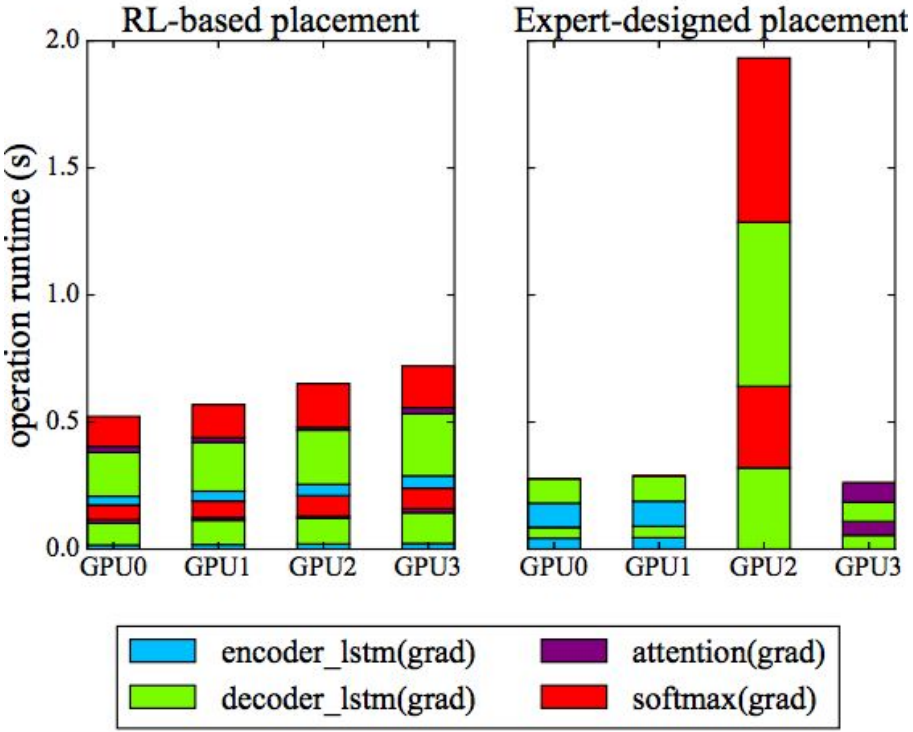


Learned Placement on NMT

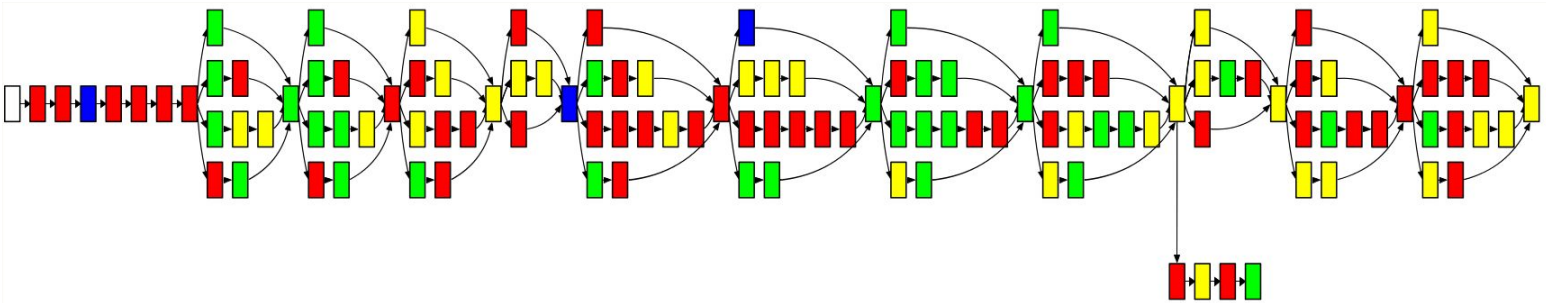


White represents CPU (Ixion Haswell 2300)
Each other color represents a separate GPU (Nvidia Tesla K80)
Searching over a space of 5^280 possible assignments

Profiling Placement on NMT

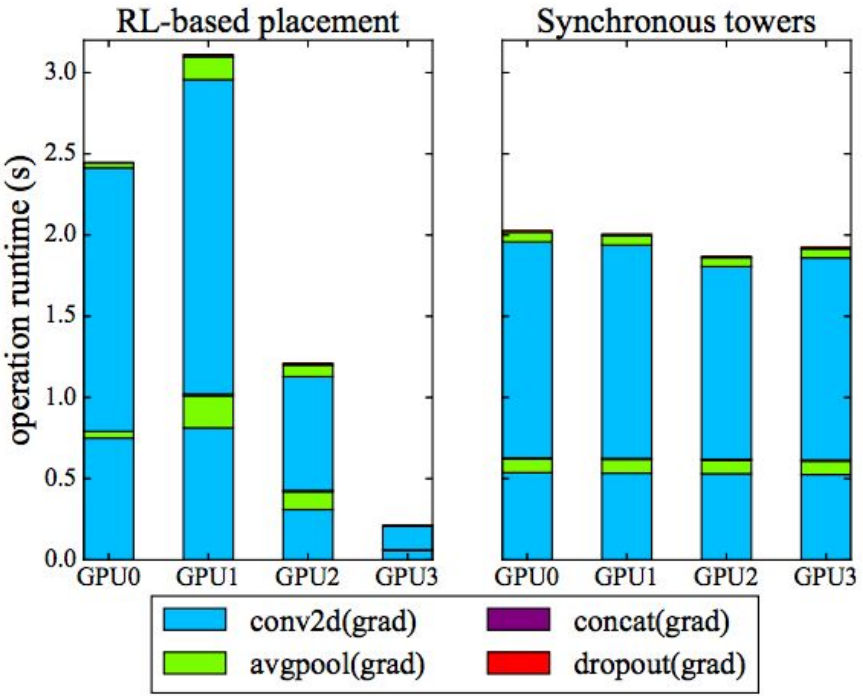


Learned Placement on Inception-V3

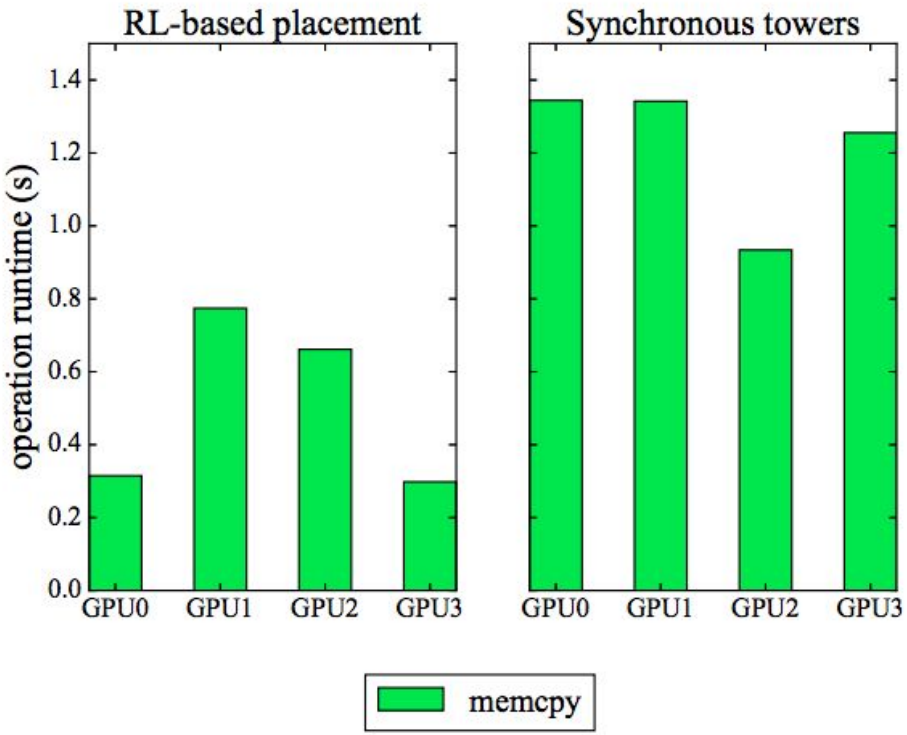


White represents CPU (Ixion Haswell 2300)
Each other color represents a separate GPU (Nvidia Tesla K80)
Searching over a space of 5^83 possible assignments

Profiling Placement on Inception-V3



Profiling Placement on Inception-V3



Generalized Policies for Device Placement

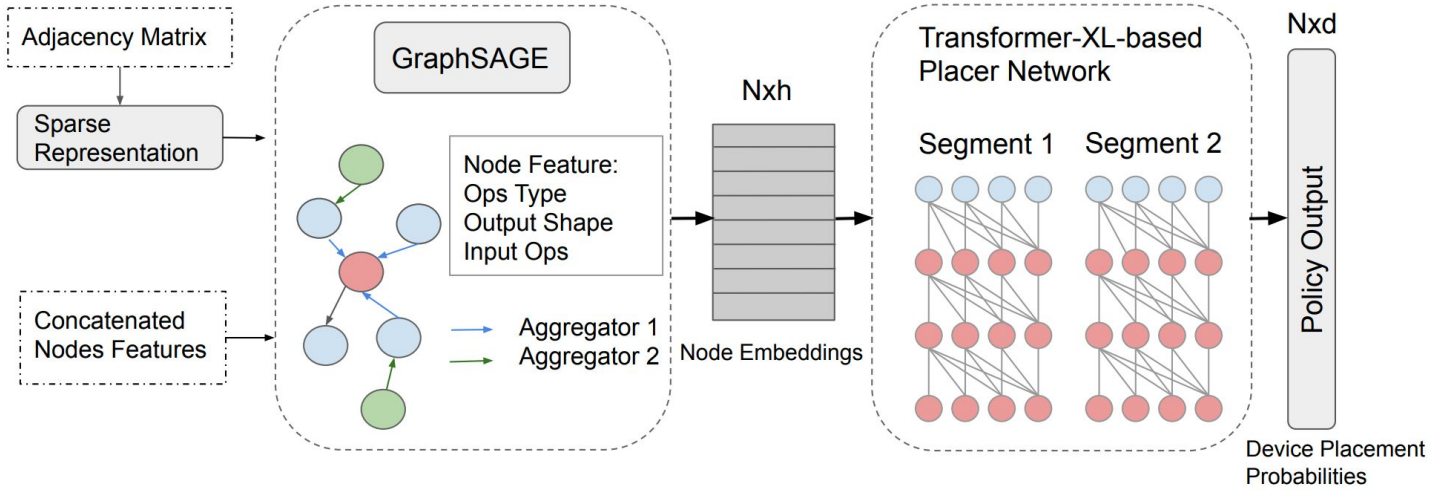
Can we train a policy that can apply device placement to new unseen graphs at inference?

Objective: Minimize expected runtime for predicted placement d across graphs

$$J(\theta) = \mathbb{E}_{G\sim\mathcal{G}, D\sim\pi_\theta(G)}[r_{G,D}] \approx \frac{1}{N}\sum_{G}\mathbb{E}_{D\sim\pi_\theta(G)}[r_{G,D}]$$

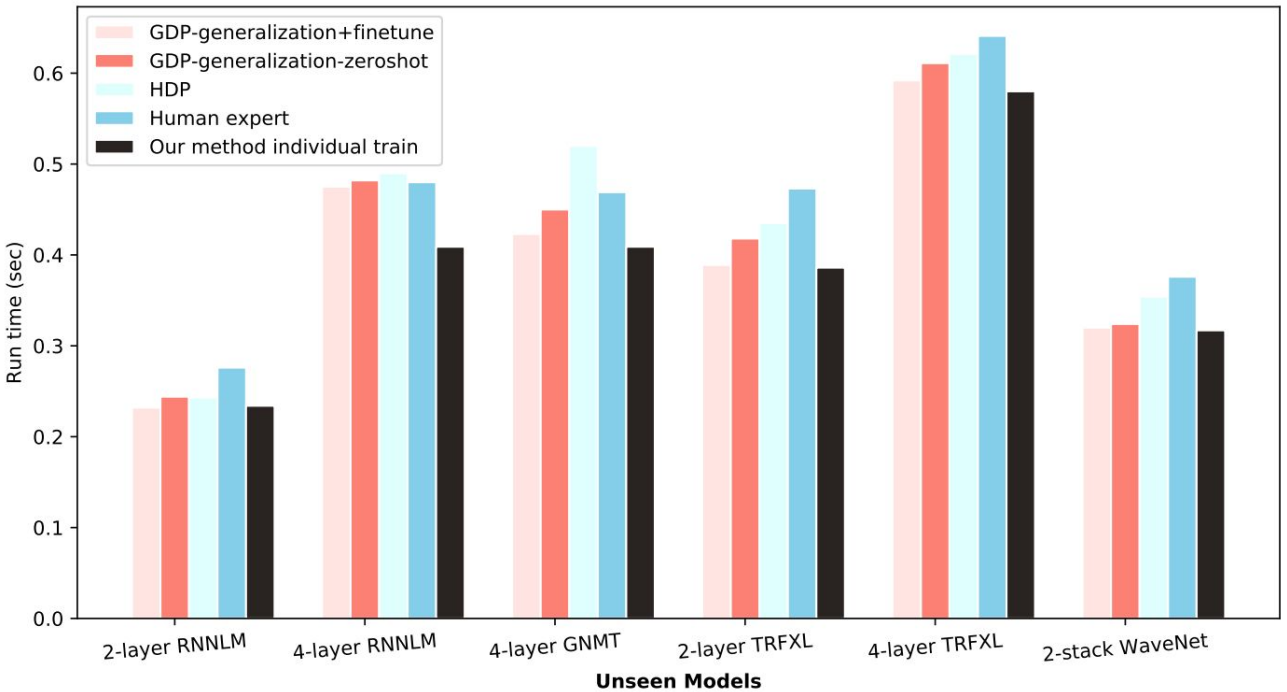
Generalized Device Placement Architecture

Generalization requires new architectures and embeddings that transfer knowledge across graphs with different size and operation types



Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, James Laudon, arxiv 2019 “GDP: generalized device placement for dataflow graphs”

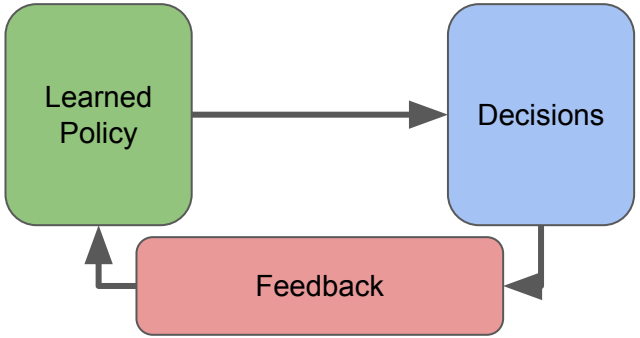
Generalization Results



Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, James Laudon, arxiv 2019 “GDP: generalized device placement for dataflow graphs”

Policy Optimization for Device Placement

Model (#devices)	**GDP-one** (s)	HP (s)	METIS (s)	HDP (s)	Run time speed up over HP / HDP	Search speed up
2-layer RNNLM (2)	0.234	0.257	0.355	0.243	9.8% / 4%	2.95x
4-layer RNNLM (4)	0.409	0.48	OOM	0.490	17.4% / 19.8%	1.76x
2-layer GNMT (2)	0.301	0.384	OOM	0.376	27.6% / 24.9%	30x
4-layer GNMT (4)	0.409	0.469	OOM	0.520	14.7% / 27.1%	58.8x
8-layer GNMT (8)	0.649	0.610	OOM	0.693	-6% / 6.8%	7.35x
2-layer Transformer-XL (2)	0.386	0.473	OOM	0.435	22.5% / 12.7%	40x
4-layer Transformer-XL (4)	0.580	0.641	OOM	0.621	11.4% / 7.1%	26.7x
8-layer Transformer-XL (8)	0.748	0.813	OOM	0.789	8.9% / 5.5%	16.7x
Inception (2)	0.405	0.418	0.423	0.417	3.2% / 3%	13.5x
AmoebaNet (4)	0.394	0.44	0.426	0.418	26.1% / 6.1%	58.8x
2-stack 18-layer WaveNet (2)	0.317	0.376	OOM	0.354	18.6% / 11.7%	6.67x
4-stack 36-layer WaveNet (4)	0.659	0.988	OOM	0.721	50% / 9.4%	20x
GEOMEAN	-	-	-	-	**16% / 9.2%**	**15x**



1- Azalia Mirhoseini*, Hieu Pham*, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean, ICML'17: Device Placement Optimization with Reinforcement Learning

2- Azalia Mirhoseini*, Anna Goldie*, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean, ICLR'18: A Hierarchical Model for Device Placement

3- Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, James Laudon, arxiv 2019 "GDP: generalized device placement for dataflow graphs"

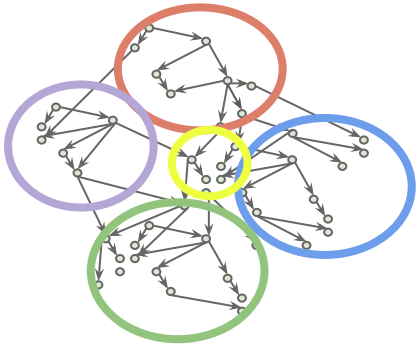
Outline of This Talk

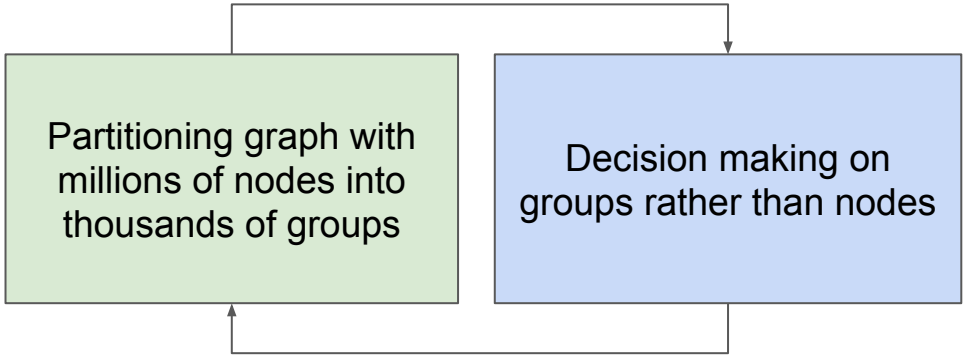
- Learning to Optimize Device Placement
- **Learning to Partition Graphs**
- Learning to Optimize Chip Placement

Why Graph Partitioning?

- Reduce complexity by breaking down problems into smaller subproblems

- Applications in many fields:
  - VLSI Design
  - Device placement
  - Distributed social network
  - Clustering
  - ...



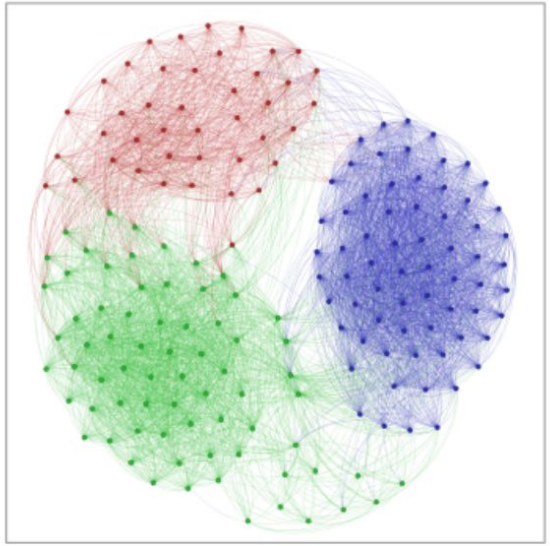


Graph Partitioning: the Normalized Cuts Objective

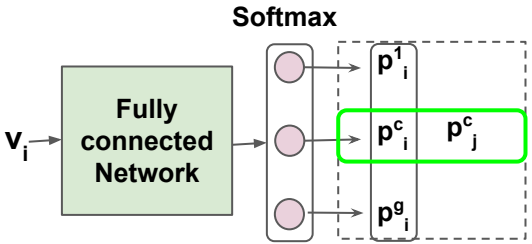
$$Ncuts(S_1, S_2, \dots S_g) = \sum_{k=1}^{g} \frac{cut(S_k, \bar{S}_k)}{vol(S_k, V)}$$

$$cut(S_k, \bar{S}_k) = \sum_{v_i \in S_k, v_j \in \bar{S}_k} w_{ij}$$

$$vol(S_k, V) = \sum_{v_i \in S_k} \sum_{v_j \in V} w_{ij}$$



Can We Learn to Partition Graphs?



P^c_i : Probability of v_i belonging to cluster c

Node	Probability of belonging to cluster c	Probability of Not belonging to cluster c
v_i	p^c_i	$1- p^c_i$
v_j	p^c_j	$1- p^c_j$

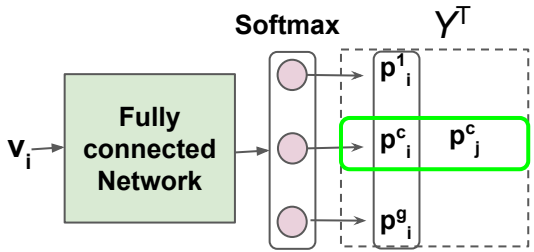
Expected cuts of cluster c with respect to v_i and v_j	Expected Volume of cluster c with respect to v_i and v_j
$p^c_i (1- p^c_j) + p^c_j (1- p^c_i)$	$p^c_i + p^c_j$

Continuous relaxation of Normalized cuts

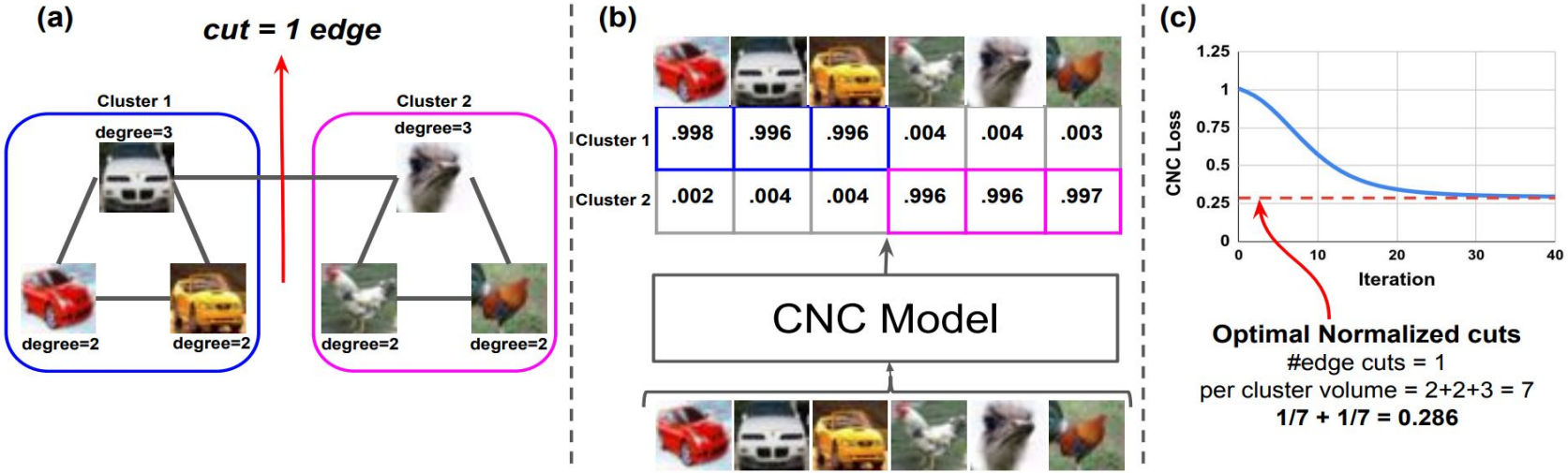
We Define an Unsupervised Loss to Optimize Expected Normalized Cuts

$$\mathbb{E}[Ncuts(S_1, S_2, \ldots S_g)] = \sum_{\text{reduce-sum}} (Y \oslash \Gamma)(1 - Y)^\intercal \odot W$$

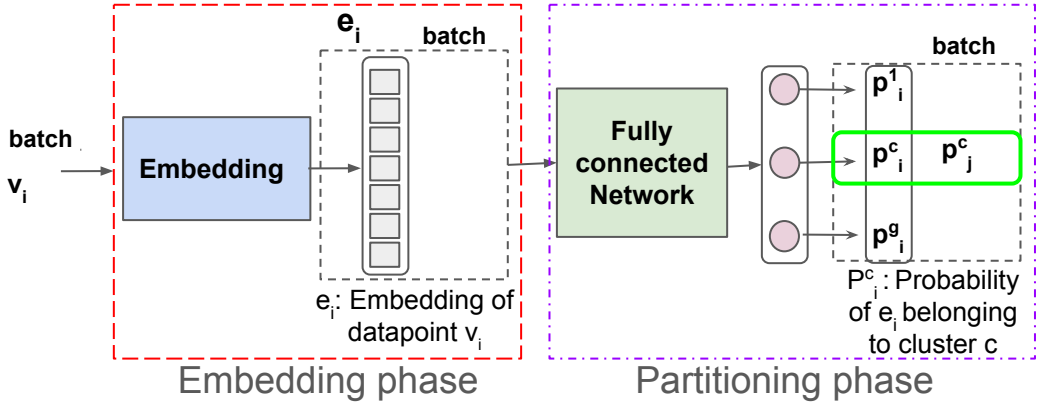
$$\Gamma = Y^\intercal D$$



Motivation Example:
Optimize Expected Normalized Cuts



Generalized Graph Partitioning Framework



Generalization Result for Graph Partitioning

Clustering Application

Method	MNIST		Reuters		CIFAR-10		CIFAR-100	
	ACC	NMI	ACC	NMI	ACC	NMI	ACC	NMI
DEC	0.843*	0.800*	0.756*	-	0.469	-	-	-
DCN	0.830**	0.810**	-	-	-	-	-	-
VaDE	0.945†	-	0.794†	-	-	-	-	-
DEPICT	0.965††	0.917††	-	-	-	-	-	-
IMSAT	0.984‡‡	-	0.719‡‡	-	0.456‡‡	-	0.275‡‡	-
IIC	**0.993**†††	-	-	-	0.617†††	-	-	-
SpectralNet	0.971‡	**0.924**‡	0.803‡	0.532‡	0.501	0.463	0.236	0.231
CNC	0.972	**0.924**	**0.824**	**0.583**	**0.702**	**0.586**	**0.345**	**0.502**

Device placement Application

Embedding	AlexNet		Inception-v3		ResNet	
	Cut	Bal	Cut	Bal	Cut	Bal
None	0.166	0.717	0.242	0.740	0.450	0.908
GCN	0.078	0.996	0.123	0.984	0.110	0.941
GraphSAGE-off	0.070	0.998	0.088	0.992	0.093	0.959
GraphSAGE-on	0.069	0.998	0.064	0.987	0.084	0.983
Graph partitioning tools	0.046	0.99	0.037	0.99	0.033	0.99

- A generalized framework for graph partitioning:
  - No need to redo the entire optimization for each new graph
  - In device placement, model is trained on VGG graph (1,326 nodes) and tested on Resnet (20,586)
- An end to end unsupervised learning approach
  - We define a differentiable loss function with continuous relaxation of the normalized cut
- Scales to graphs with millions of nodes

1) Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, Azalia Mirhoseini, "Generalized Clustering by Learning to Optimize Expected normalized cuts" (To be open-sourced on github.com/google-research by 11/2019, Under review in ICLR'20)
2) Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, Azalia Mirhoseini, "A Deep Learning Framework for Graph Partitioning", ICLR'19 workshop on graph representations

Outline of This Talk

- Learning to Optimize Device Placement
- Learning to Partition Graphs
- **Learning to Optimize Chip Placement**

ML for Chip Design Goals

- Reducing the design cycle from 1.5-2 years to weeks
  - Today, we design chips for the NN architectures of 2-5 years from now
  - Shortening the chip design cycle would enable us to be far more adaptive to the rapidly advancing field of machine learning

- New possibilities emerge if we evolve NN architectures and chips together
  - Discovering the next generation of NN architectures (which would not be computationally feasible with today's chips)

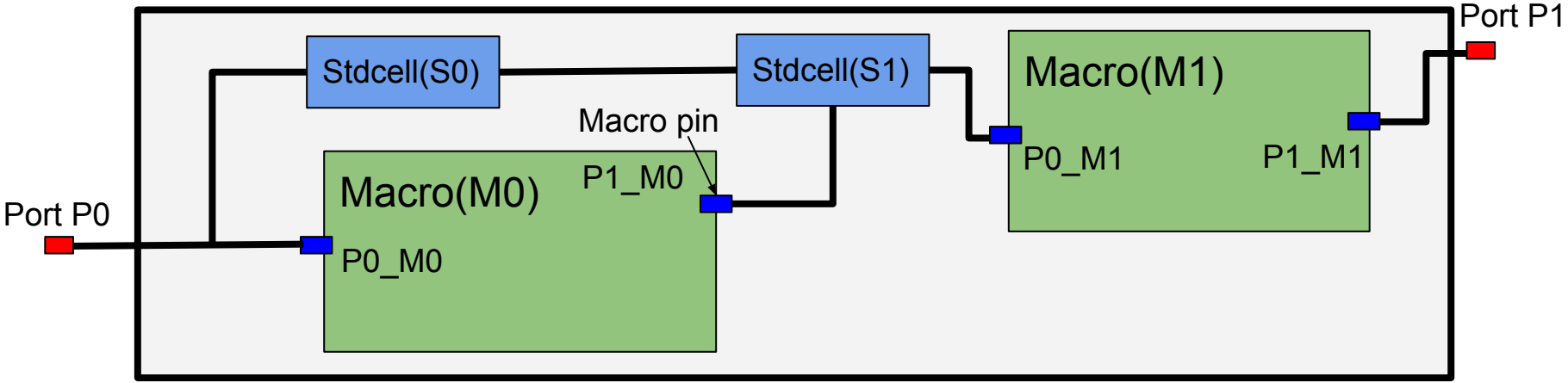
- Enabling cheaper, faster, and more environmentally friendly chips

Learning to Optimize Chip Placement

- **Chip Placement Problem**
- Our Approach
- Knowledge Transfer Across Chips

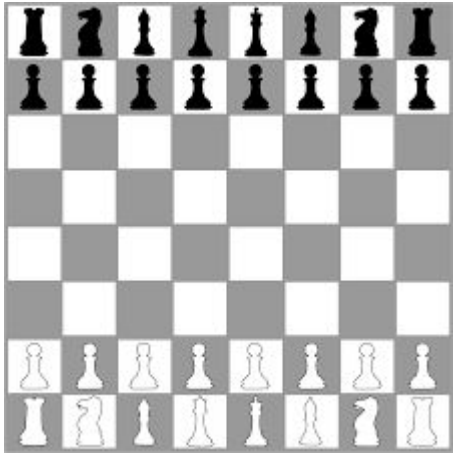
Chip Placement Problem

- 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.



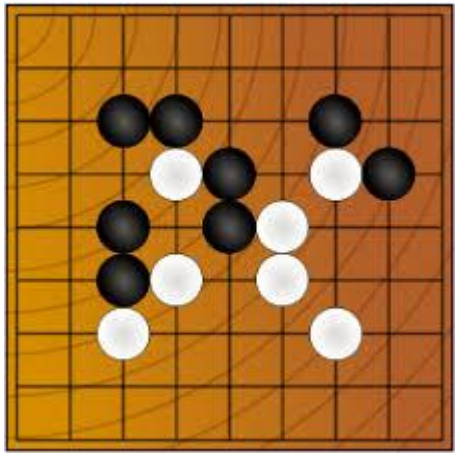
Complexity of Chip Placement Problem

Chess



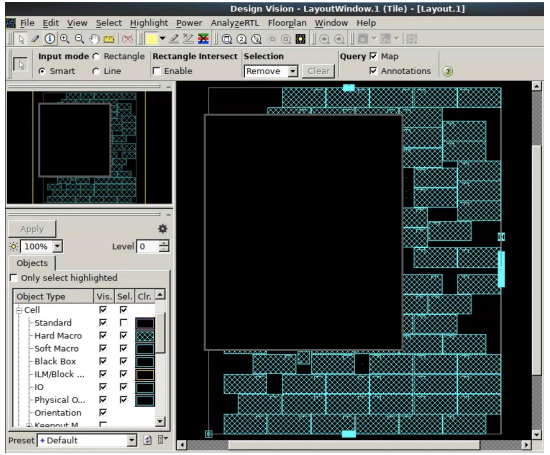
Number of states ~ 10^{123}

Go



Number of states ~ 10^{360}

Chip Placement

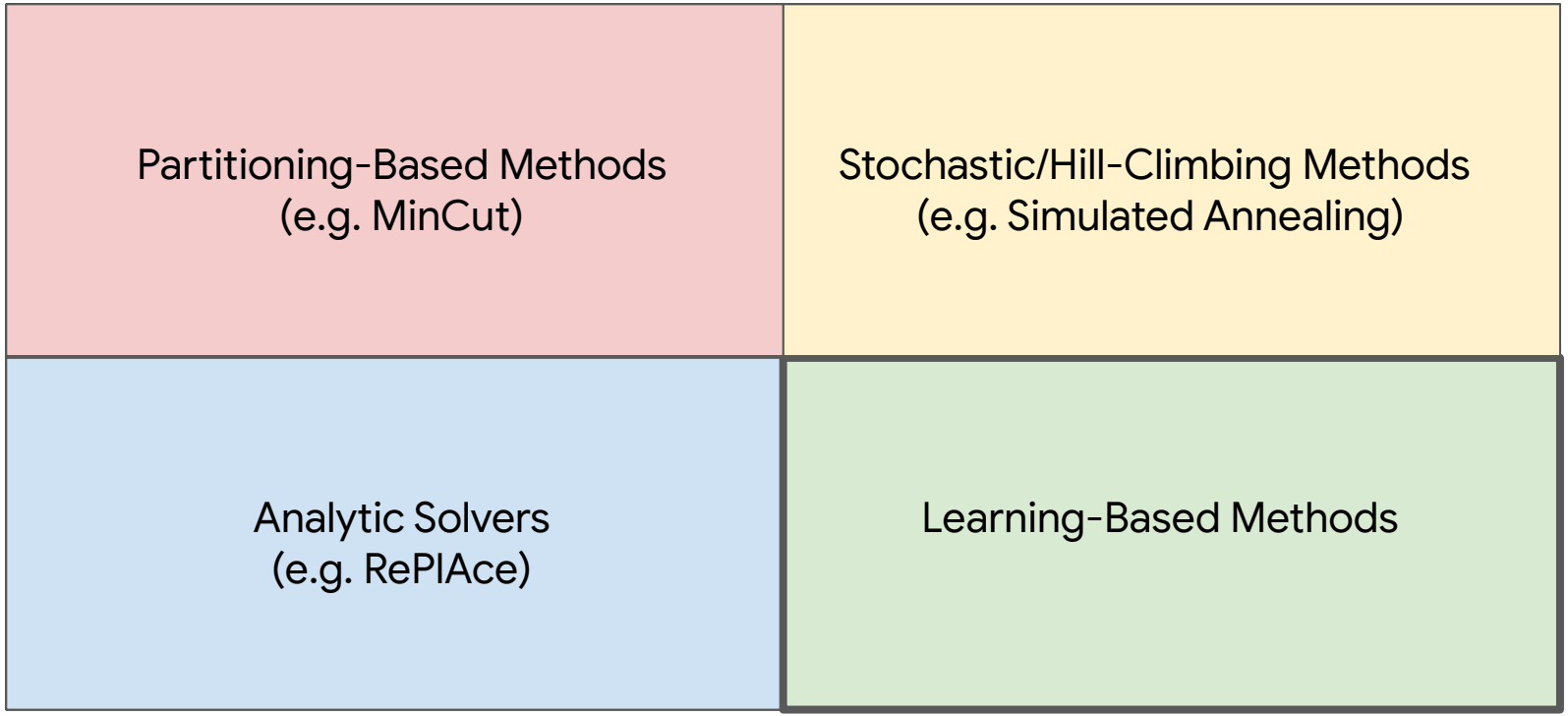


Number of states ~ 10^{9000}

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



Learning to Optimize Chip Placement

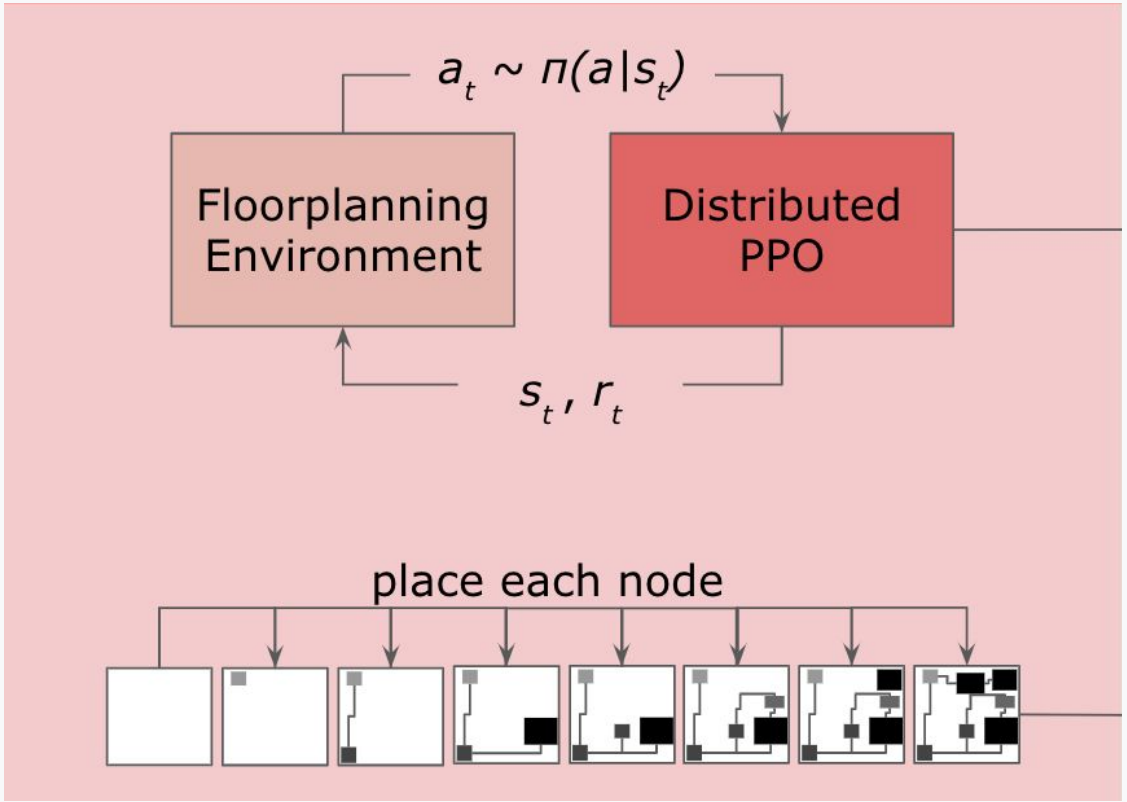
- Chip Placement Problem
- **Our Approach**
- Knowledge Transfer Across Chips

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



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)

Our Objective Function

Reward corresponding to placement p of netlist (graph) g

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

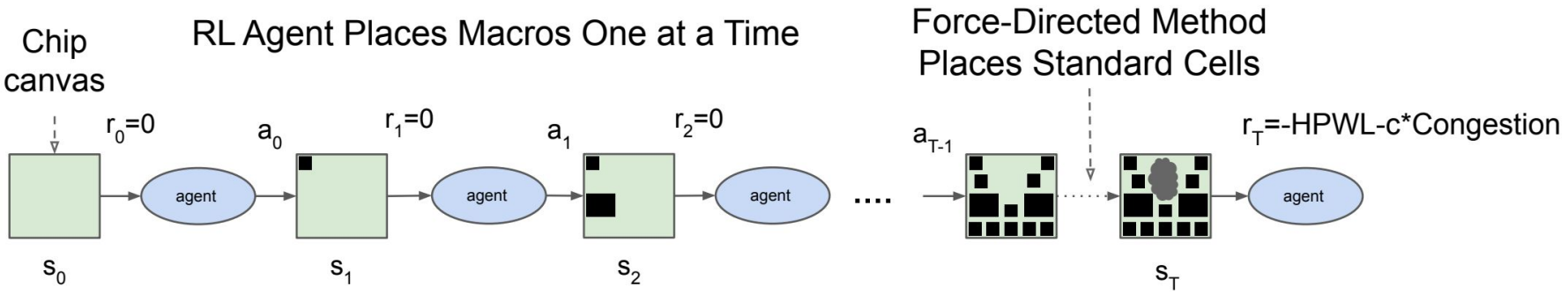
Set of training graphs G

K is size of training set

RL policy parameterized by theta

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

We Take a Hybrid Approach to Placement Optimization

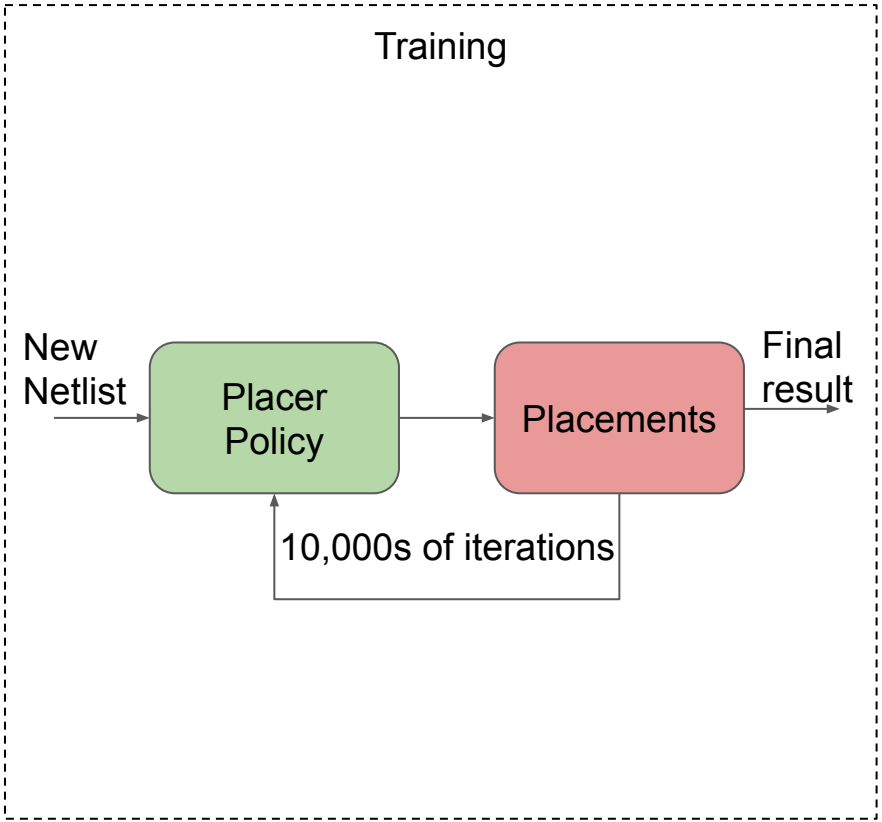


Learning to Optimize Chip Placement

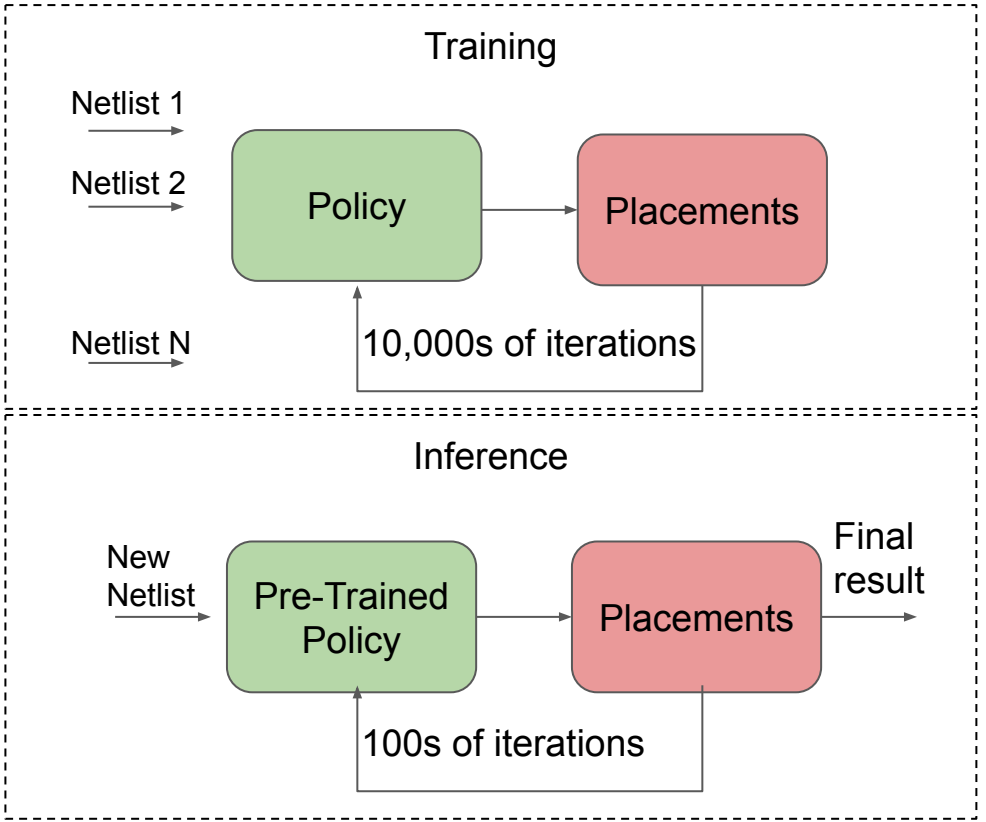
- Chip Placement Problem
- Our Approach
- **Knowledge Transfer Across Chips**

Moving Towards Generalized Placements

Before: Training from scratch for each chip 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!

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.

Achieving Generalization by Training Accurate Reward Predictors

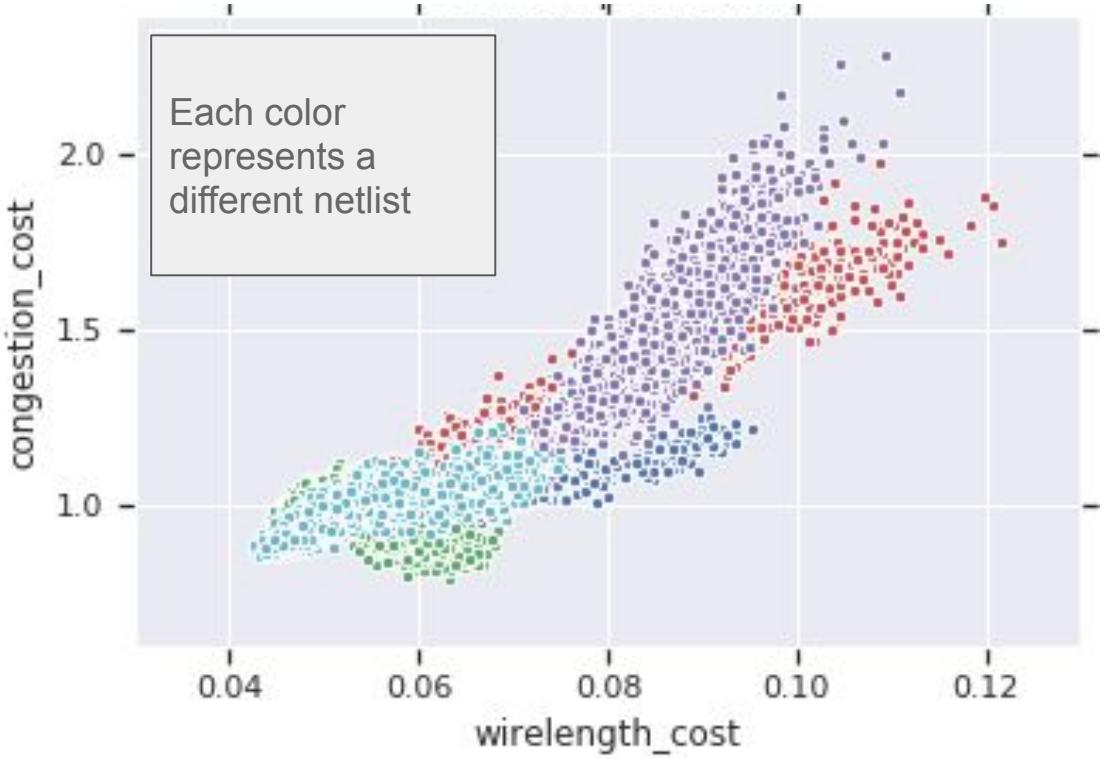
We observed that a value network trained only on placements generated by a single policy is unable to accurately predict the quality of placements generated by another policy, limiting the ability of the policy network to generalize.

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

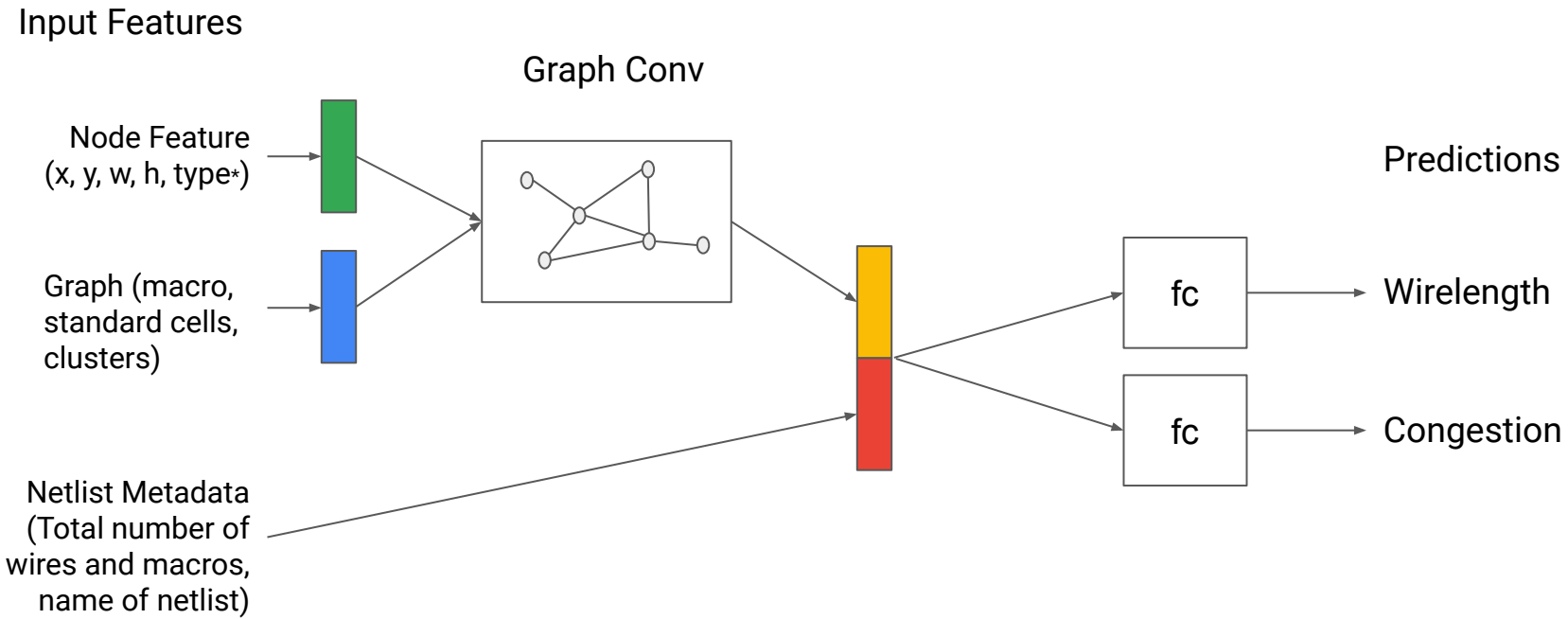
Compiling a Dataset of Chip Placements

To train a more accurate predictor, we generated a dataset of 10k placements

Each placement was labeled with their wirelength and congestion, which were drawn from vanilla RL policies.

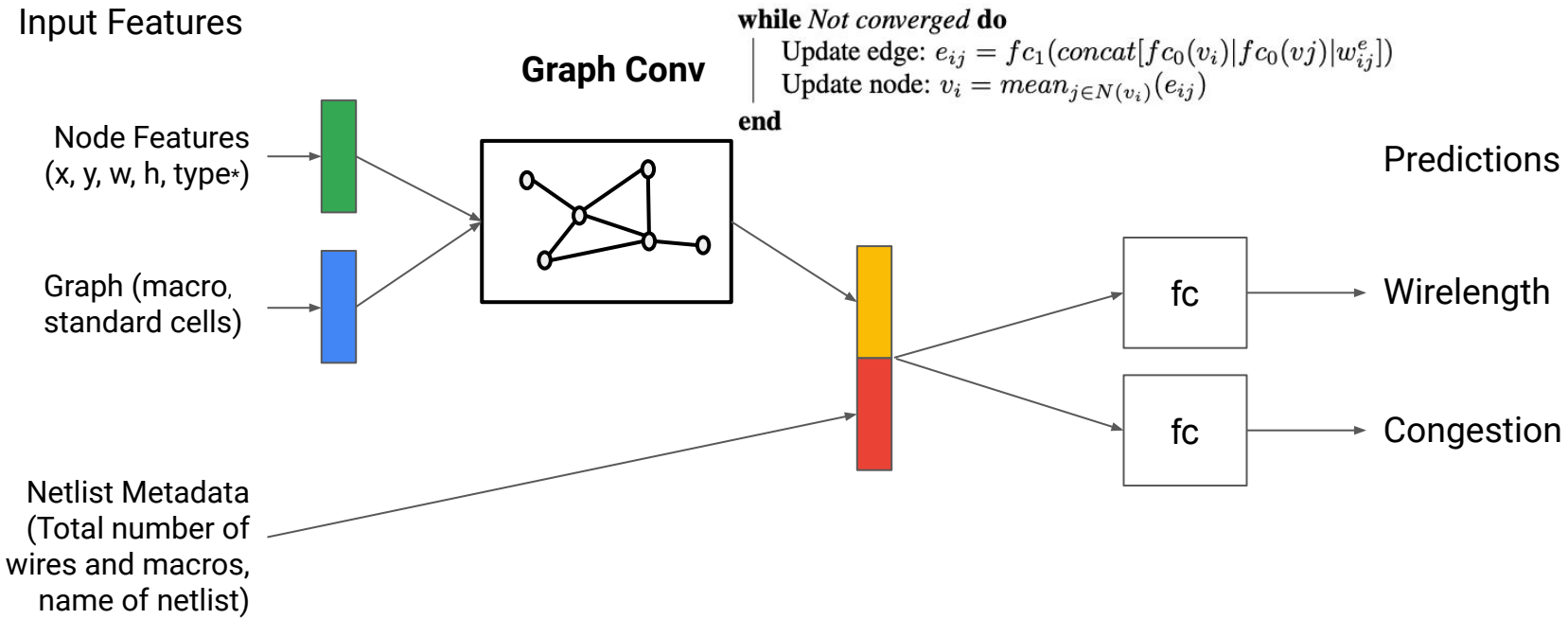


Reward Model Architecture and Features



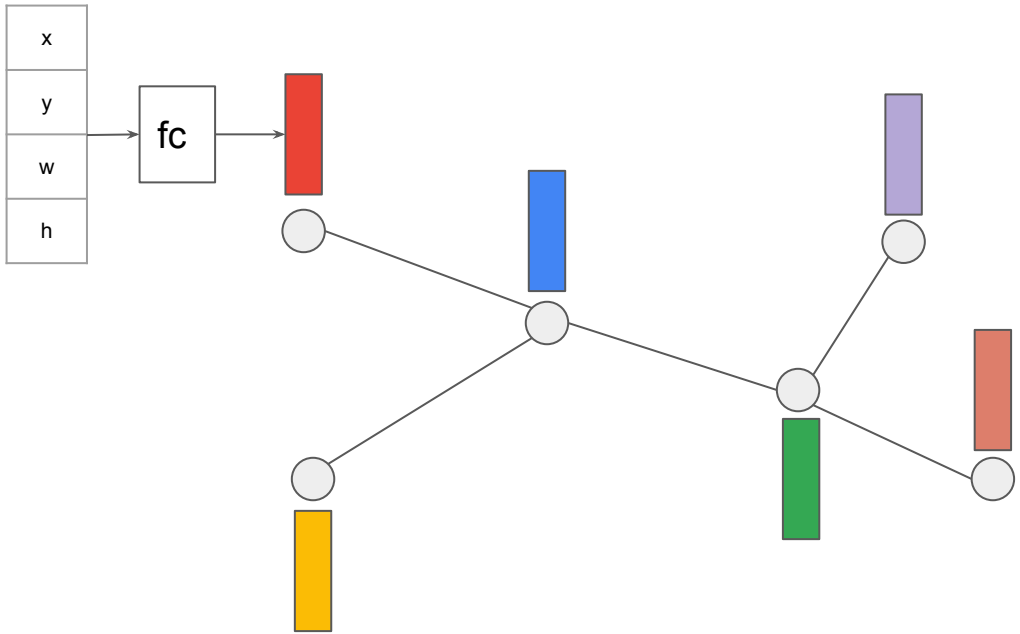
*Node type: One-hot category {Hard macro, soft macro}

Reward Model Architecture and Features

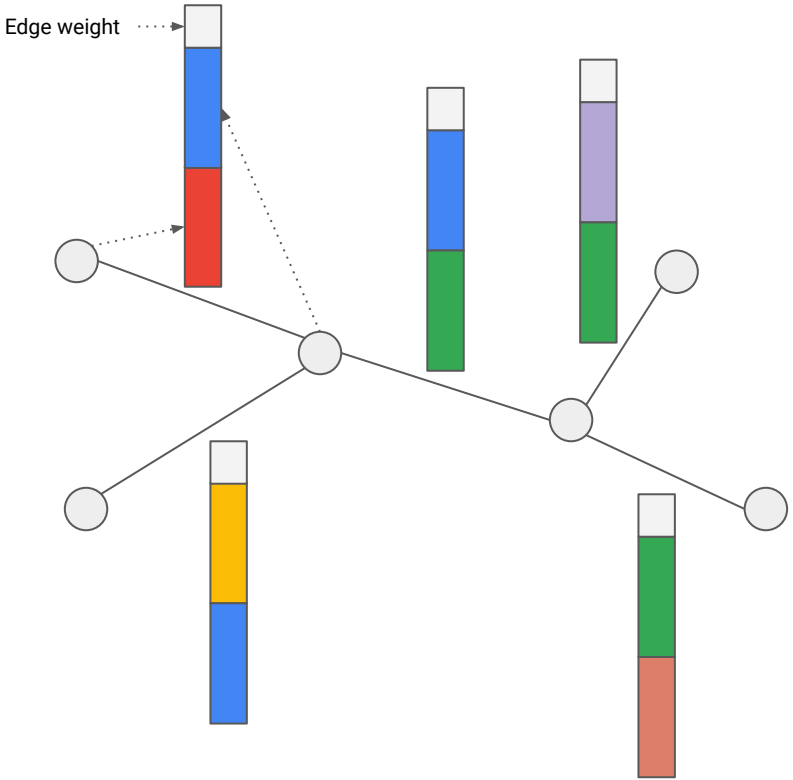


*Node type: One-hot category {Hard macro, soft macro}

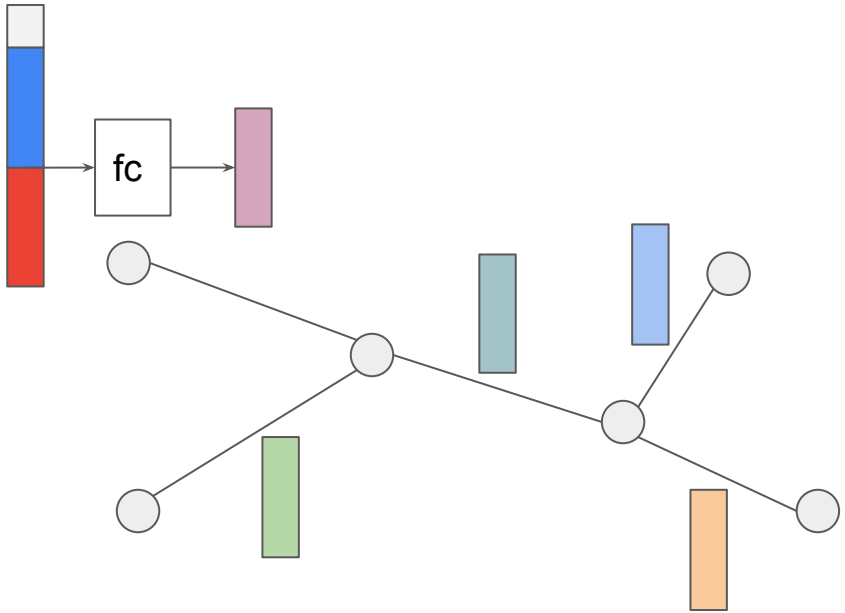
Edge-based Graph Convolution: Node Embeddings



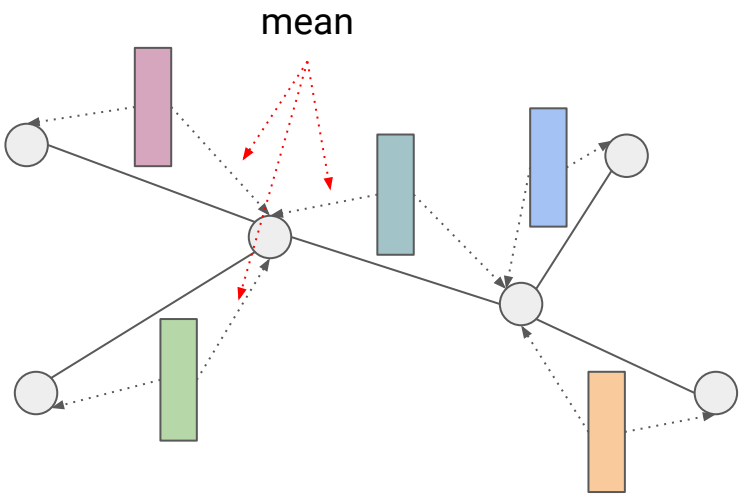
Edge-based Graph Convolution: Edge Embedding



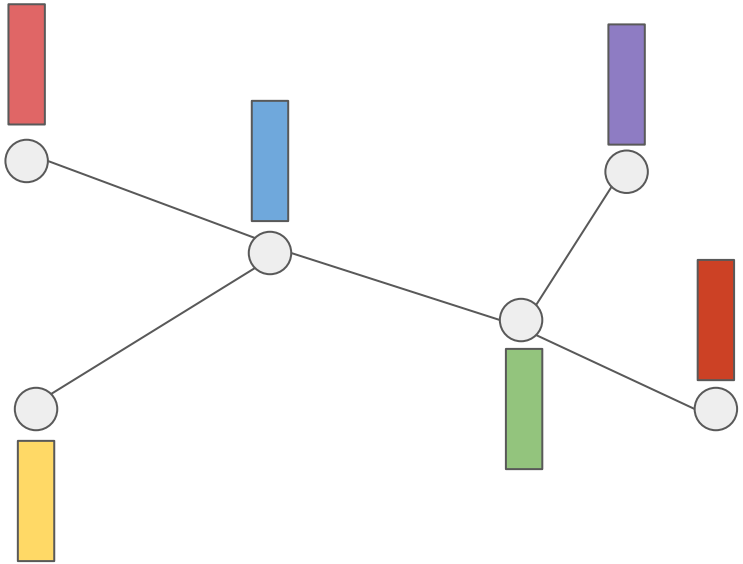
Edge-based Graph Convolution: Edge Embedding



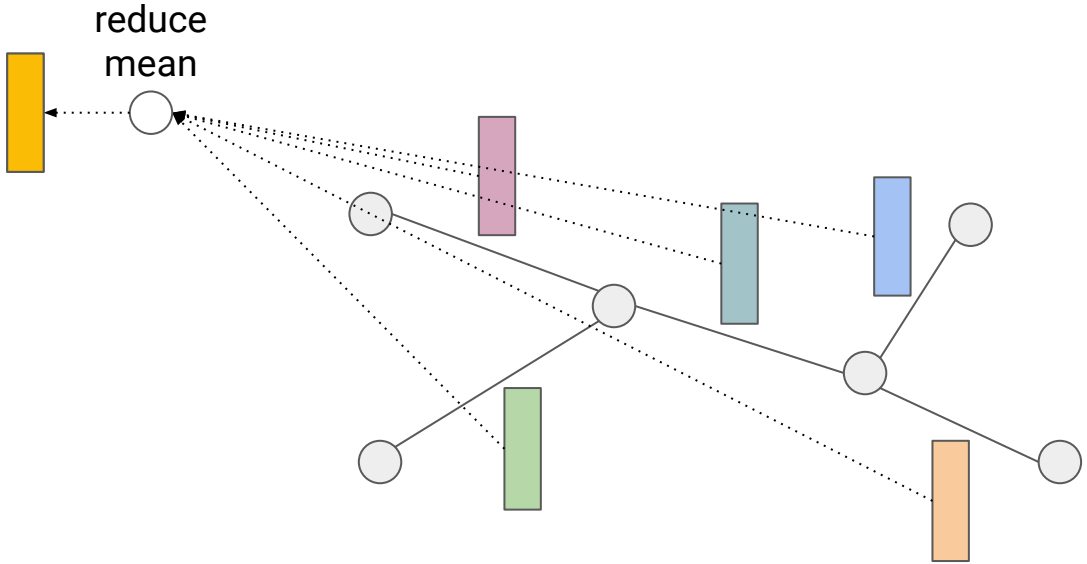
Edge-based Graph Convolution: Propagate



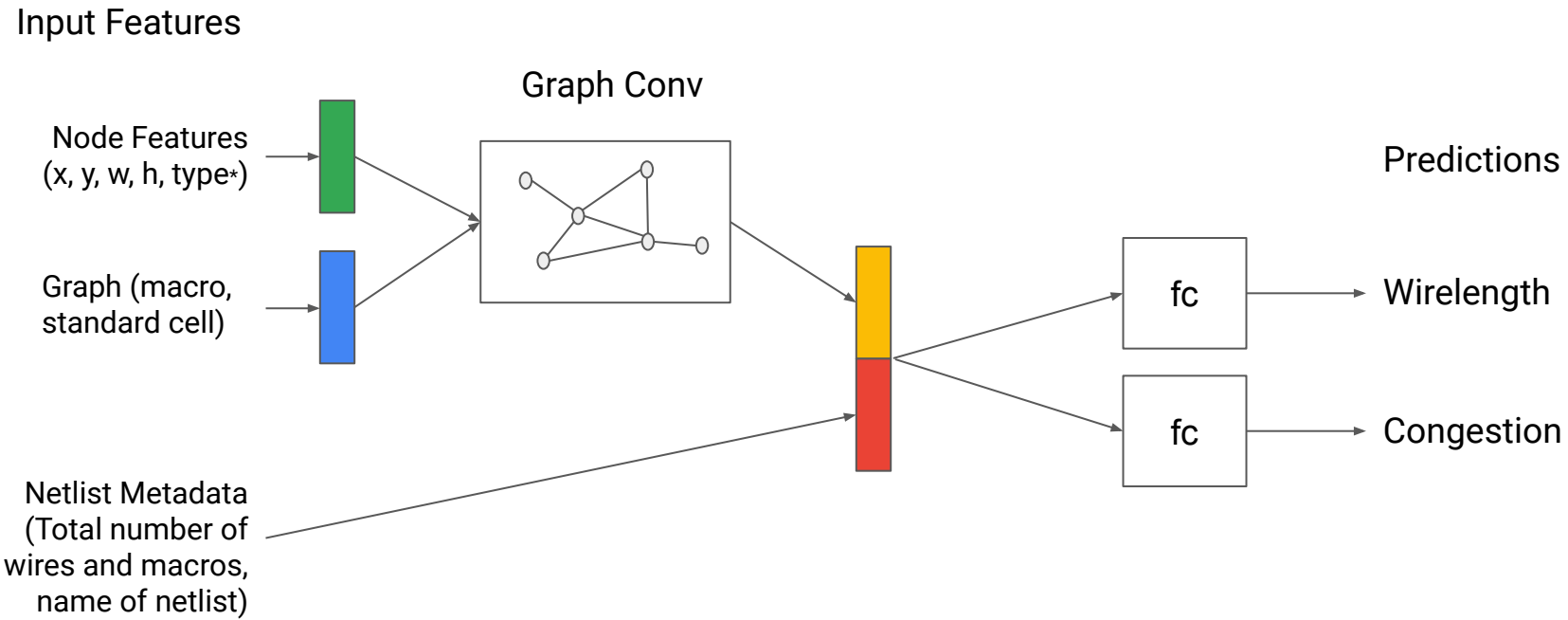
Edge-based Graph Convolution: Repeat



Final Step: Get Graph Embedding

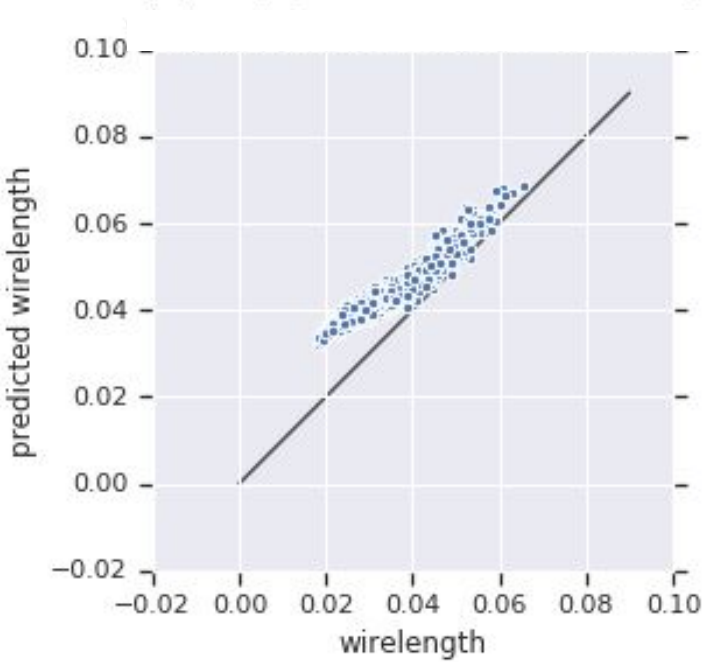


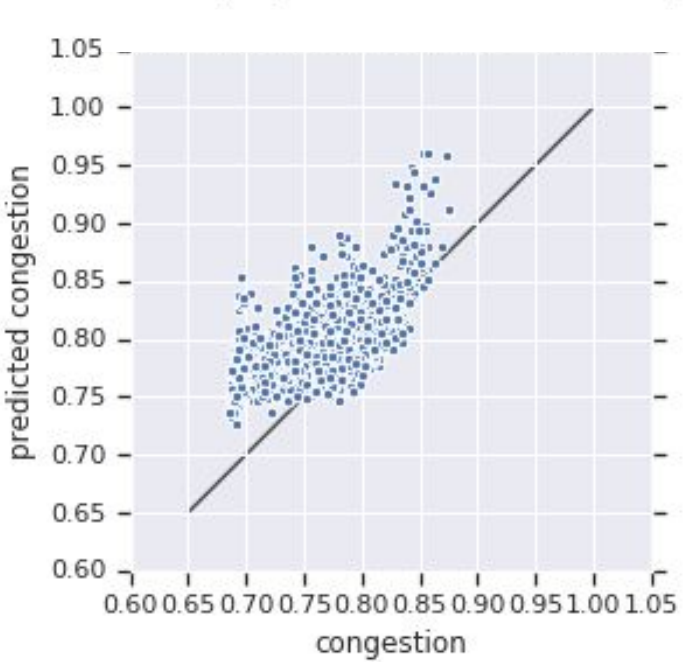
Reward Model Architecture and Features



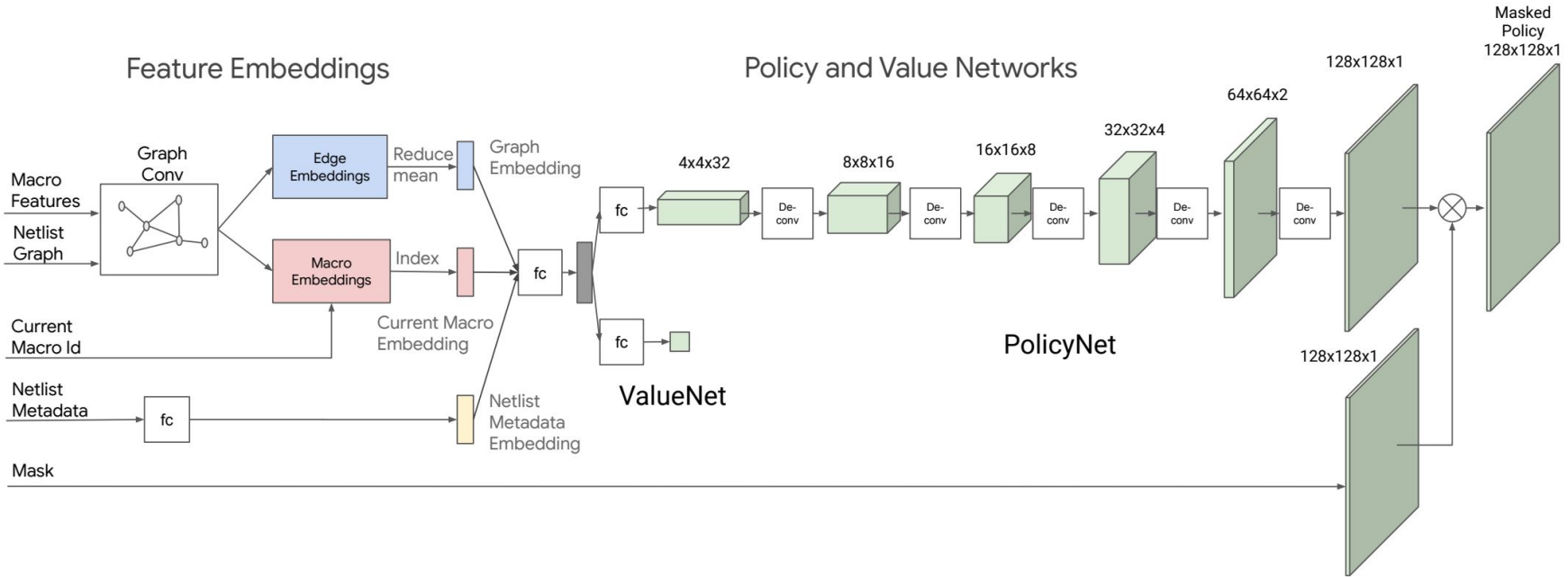
*Node type: One-hot category {Hard macro, soft macro}

Label Prediction Results on Test Chips



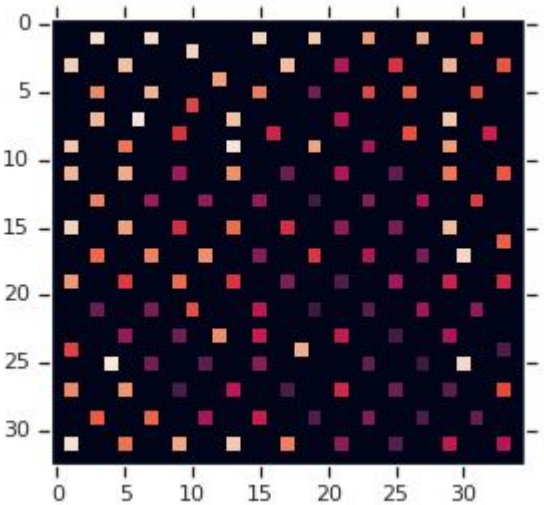


Policy/Value Model Architecture

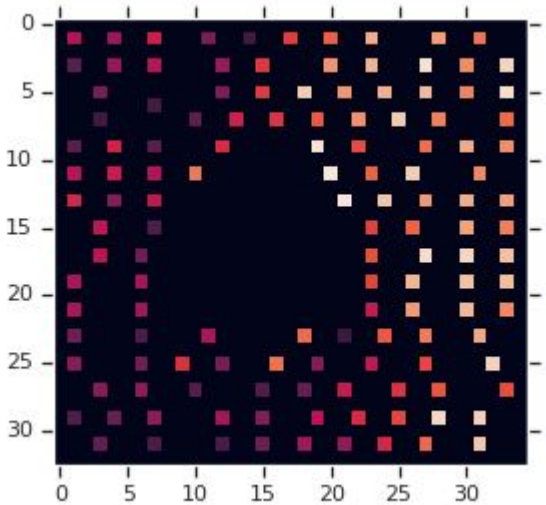


Ariane (RISC-V) Placement Visualization

Training policy from scratch



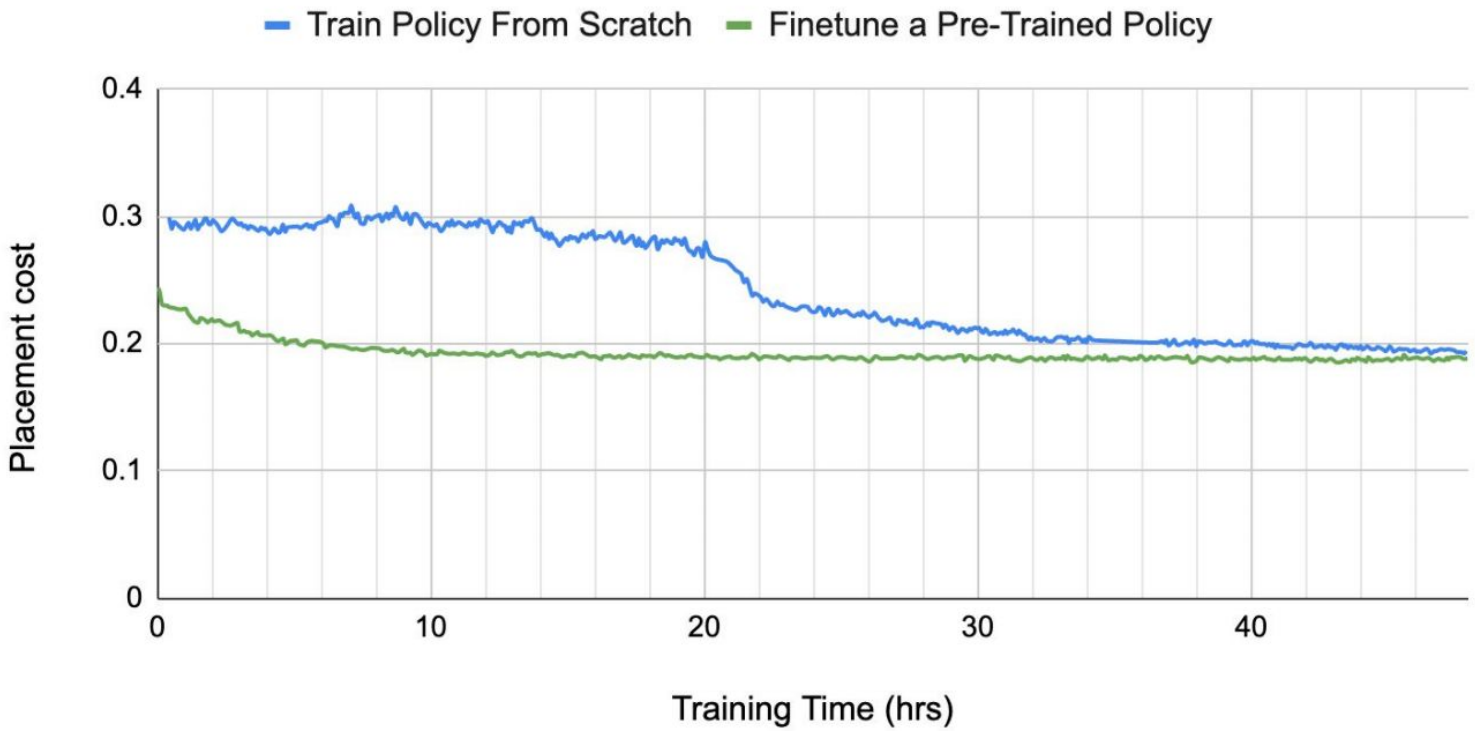
Finetuning a pre-trained policy



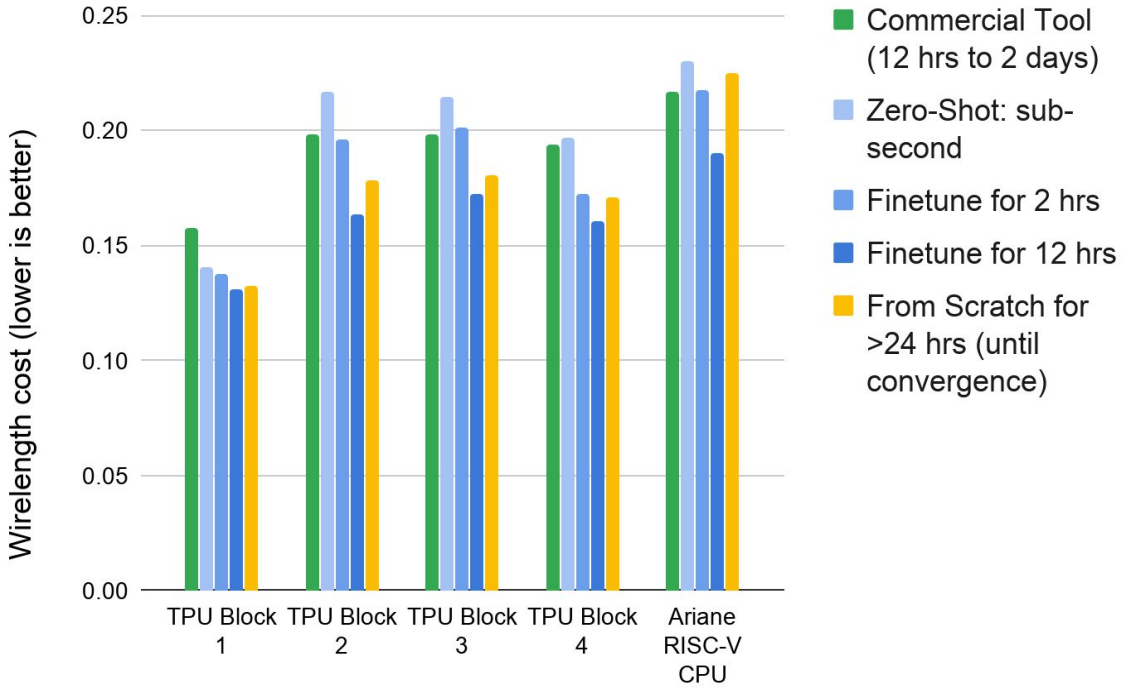
The animation shows the macro placements as the training progresses. Each square shows the center of a macro.

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

Convergence Curve: Training from Scratch vs. Finetuning

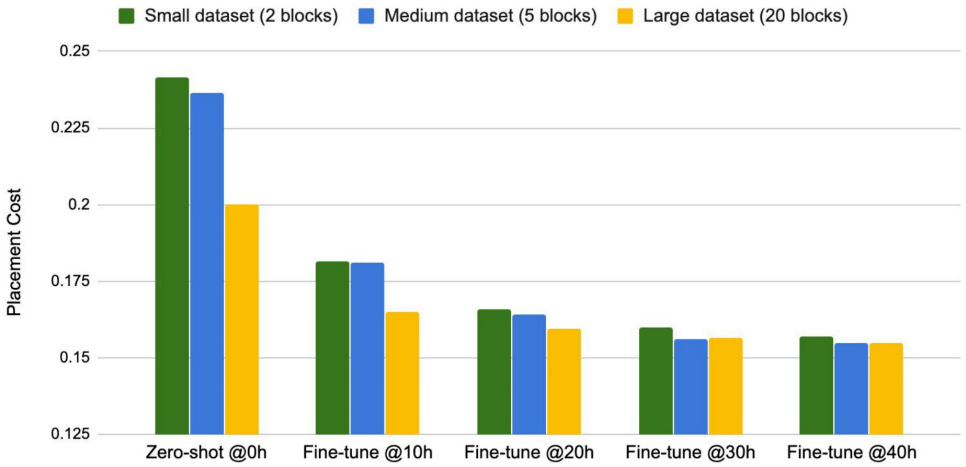


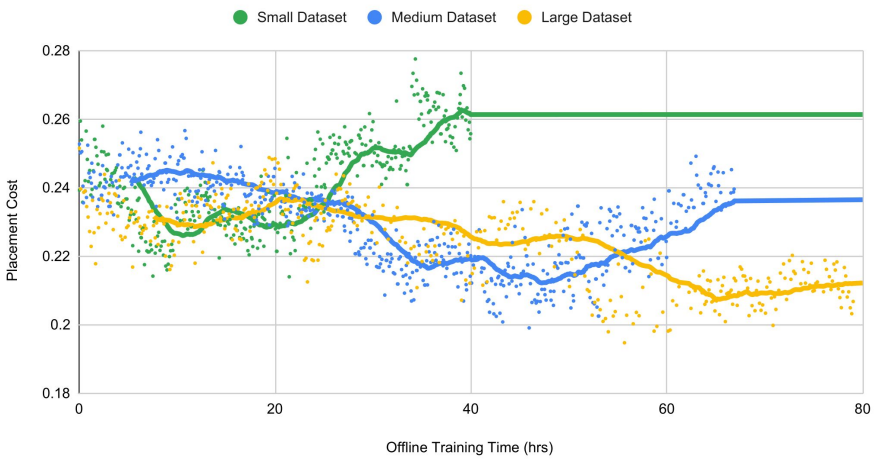
Generalization Results



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

Effects of Training Set Size on Convergence





Comparisons with Manual and SOTA Baselines

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 the 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

IEEE SPECTRUM

Google Invents AI That Learns a Key Part of Chip Design

AI helps designs AI chip that might help an AI design future AI chips

By **Samuel K. Moore**

MIT Technology Review

Google is using AI to design chips that will accelerate AI

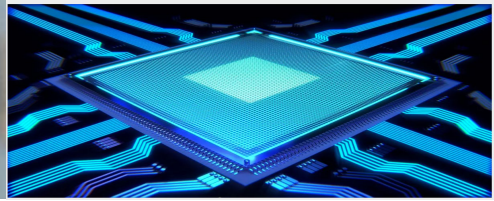
PC GAMER

Google is using AI to design AI processors much faster than humans can

By Paul Lilly 10 days ago

Chips making chips.

COMMENTS



Google Proposes AI as Solution for Speedier AI Chip Design

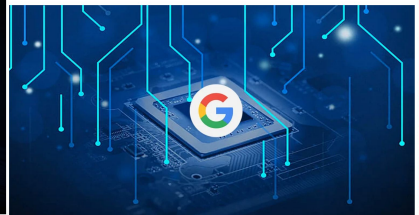
Google uses artificial intelligence to optimize AI chip production

By **Mario McKellop** - April 2, 2020 140 0

Google Hoping The Next AI Chips Will Be Designed By AI

Company researchers have come up with an AI system that can design other AI chips. The goal is to help improve AI with the help of AI.

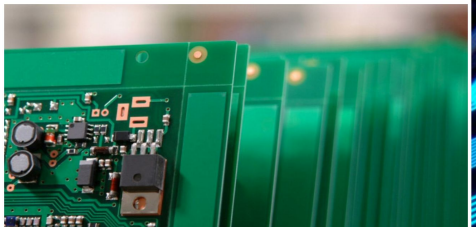
BY ASWIN NARAYANAN
MAR 31, 2020



Google Researchers Create AI-ception with an AI Chip That Speeds Up AI

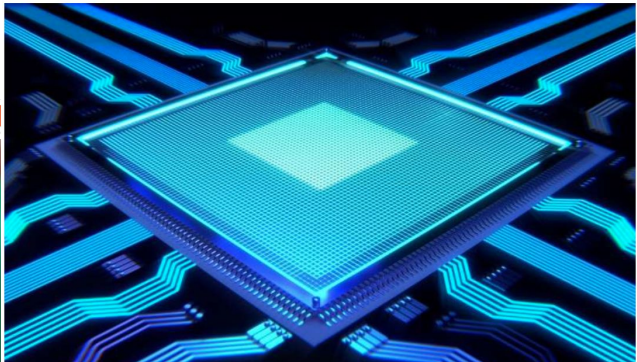
Using a reinforcement-learning algorithm, the AI has learnt to optimize the placement of components on a computer chip.

By Fabienne Lang
March 30, 2020



Google trains chips to design themselves

by Peter Grad , Tech Xplore



If you'd like to learn more:

Chip Placement is A Collaboration Between Google Research and CI2 Orgs

Paper: Chip Placement with Deep Reinforcement Learning, Azalia Mirhoseini*, Anna Goldie*, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Sungmin Bae, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Anand Babu, Quoc V Le, James Laudon, Richard Ho, Roger Carpenter, Jeff Dean

Google AI Blog: https://ai.googleblog.com/2020/04/chip-design-with-deep-reinforcement.html

Other work on placement/combinatorial optimization with ML by our group:

1. Placement Optimization with Deep Reinforcement Learning, Anna Goldie, Azalia Mirhoseini, ISPD, 2020.
2. GAP: Generalizable Approximate Graph Partitioning Framework. Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, Azalia Mirhoseini, ICLR workshop on graph representation, 2019.
3. Generalized Clustering by Learning to Optimize Expected Normalized Cuts, Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, Azalia Mirhoseini, Neurips workshop on sets and partitions, 2019.
4. A Reinforcement Learning Driven Heuristic Optimization Framework, Qingpeng Cai, Will Hang, Azalia Mirhoseini, George Tucker, Jingtao Wang, Wei Wei, KDD workshop on deep reinforcement learning for knowledge discovery, 2019.
5. GDP: generalized device placement for dataflow graphs, Yanqi Zhou, Sudip Roy, Amirali Abdolrashidi, Daniel Wong, Peter C. Ma, Qiumin Xu Ming Zhong, Hanxiao Liu, Anna Goldie, Azalia Mirhoseini, James Laudon, 2019.
6. A Hierarchical Model for Device Placement, Azalia Mirhoseini*, Anna Goldie*, Hieu Pham, Benoit Steiner, Quoc V. Le and Jeff Dean, ICLR, 2018.
7. Device Placement Optimization with Reinforcement Learning, Azalia Mirhoseini*, Hieu Pham*, Quoc V. Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, Jeff Dean, ICML, 2017.